WebSphere® Business Integration Server Foundation for z/OS V5.1

Resources

SA22-7979-00

Note
Before using this information, be sure to read the general information under "Notices" on page 369.

Compilation date: May 21, 2004

Contents

How to send your comments

Your feedback is important in helping to provide the most accurate and highest quality information.

- To send comments on articles in the WebSphere Application Server Information Center
  1. Display the article in your Web browser and scroll to the end of the article.
  2. Click on the **Feedback** link at the bottom of the article, and a separate window containing an e-mail form appears.
  3. Fill out the e-mail form as instructed, and click on **Submit feedback** .
- To send comments on PDF books, you can e-mail your comments to: **wasdoc@us.ibm.com** or fax them to 919-254-0206.

  Be sure to include the document name and number, the WebSphere Application Server version you are using, and, if applicable, the specific page, table, or figure number on which you are commenting.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Chapter 1. Welcome to Resources

The product supports all of the resources defined by the Java 2 Platform, Enterprise Edition (J2EE).

Data access (JDBC and J2C)

The J2EE Connector architecture defines a standard architecture that enables the integration of various enterprise information systems (EIS) with application servers and enterprise applications. It defines a standard resource adapter used by a Java application to connect to an EIS. This resource adapter can plug into the application server and, through the Common Client Interface (CCI), provide connectivity between the EIS, the application server, and the enterprise application.

Messaging

The product supports asynchronous messaging as a method of communication based on the Java Message Service (JMS) programming interface. The base JMS support enables IBM WebSphere Application Server applications to exchange messages asynchronously with other JMS clients by using JMS destinations (queues or topics).

Mail

Using JavaMail API, a code segment can be embedded in any Java 2 Enterprise Edition (J2EE) application component, such as an EJB or a servlet, allowing the application to send a message and save a copy of the mail to the Sent folder.

URLs

Java 2 Platform, Enterprise Edition (J2EE) applications can use URLs as resources in the same way other J2EE resources, such as JDBC and JavaMail, are used.

Resource environment entries

A resource environment reference maps a logical name used by the client application to the physical name of an object.

Chapter 2. Accessing data from applications

Note: WebSphere Application Server does not support JDBC 3.0.

Various enterprise information systems (EIS) use different methods for storing data. These *backend* data stores might be relational databases, procedural transaction programs, or object-oriented databases. IBM WebSphere Application Server provides several options for accessing an information system's backend data store:

- Programming directly to the database through the JDBC 2.0 Optional Package API.
- Programming to the procedural backend transaction through various J2EE Connector Architecture (JCA) 1.0 compliant connectors.
- Programming in the bean-managed persistence (BMP) bean or servlets indirectly accessing the backend store through either the JDBC API or JCA compliant connectors. The prerequisite Web site details which databases and drivers are currently supported.
- Using container-managed persistence (CMP) beans.
- Using embedded Structured Query Language in Java (SQLJ) support with applications that use DB2 as a backend database.
- Using the IBM data access beans, which also use the JDBC API, but give you additional ability to manipulate result sets.

1. Develop data access applications Develop your application to access data using the various ways available through the WebSphere Application Server. You can access data through APIs, container-managed persistence beans, bean-managed persistence beans, session beans, or Web components.
2. Assemble data access applications using the Assembly Toolkitor Application Assembly Tool (AAT). Assemble your application by creating and mapping resource references.
3. Deploy data access applications Ensure that the appropriate database objects are available. Create or configure any databases or tables required, set necessary configuration parameters to handle expected load, and configure any necessary JDBC providers and data source objects for servlets, enterprise beans, and client applications to use.

Resource adapter

A resource adapter is a system-level software driver that a Java application uses to connect to an enterprise information system (EIS).

A resource adapter plugs into an application server and provides connectivity between the EIS, the application server, and the enterprise application.

An application server vendor extends its system once to support the connector architecture and is then assured of seamless connectivity to multiple EISs. Likewise, an EIS vendor provides one standard resource adapter with the capability to plug into any application server that supports the connector architecture.

WebSphere Application Server provides the relational resource adapter (RRA) implementation in this release. This resource adapter provides data access through JDBC calls to access the database dynamically. The connection management is

based on the J2EE Connector Architecture (JCA) connection management architecture. It provides connection pooling, local transaction, and security support.

Container-managed persistence (CMP) beans data access is managed by the WebSphere Persistence Manager indirectly. The JCA Specification supports Persistence Manager delegation of the data access to the JCA resource adapter without knowing the specific backend store. For the relational database access, Persistence Manager utilizes the relational resource adapter to access the data from the database. You can find the supported database platforms for the JDBC API at the WebSphere Application Server prerequisite Web site.

J2EE Connector Architecture resource adapters

A J2EE Connector Architecture (JCA) resource adapter is any resource adapter conforming to the JCA Specification.

The product supports any resource adapter that implements version 1.0 of this specification. Although not part of WebSphere Application Server, IBM supplies resource adapters for enterprise systems such as: the Customer Information Control System (CICS), Host On-Demand (HOD), Information Management System (IMS), Systems, Applications, and Products (SAP) R/3, and Crossworlds, as separate products.

The general approach to writing an application that uses a JCA resource adapter is to develop enterprise bean session beans or services with tools such as WebSphere Studio Application Developer Integration Edition (WSADIE) or VisualAge for Java Enterprise Access Builder. The session bean uses the *javax.resource.cci* interfaces to communicate with an enterprise information system through the resource adapter.

Resource Recovery Services (RRS)

WebSphere Application Server for z/OS supports resource adapters that use Resource Recovery Services (RRS)to support global transaction processing. RRS is an z/OS extension to the JCA resource adapter specifications.

WebSphere Application Server for z/OS supports the J2EE Connector Architecture (JCA) 1.0., and because of this, any resource adapter that is designed to use the 1.0 level of the J2EE Connector Architecture (JCA) is supported.

In addition to the 3 types of transaction support defined by JCA, WebSphere Application Server for z/OS supports a fourth type, **RRSTransactional** support, which is a z/OS only extension to the architecture. Resource adapters that are capable of using RRS and that properly indicate to WAS z/OS they are RRSTransactional will be supported as RRS compliant resource adapters.

z/OS resource adapters that are capable of using RRS are:
- IMS Connector for Java
- CICS CTG ECI J2EE Connector
- IMS JDBC Connector
- DB2 for z/OS Local JDBC connector when used as aJDBC Provider under the WebSphere Relational Resource Adapter (RRA)

All RRS Compliant resource adapters are required to support the property **RRSTransactional** in their ManagedConnectionFactory and must support a getter method for the property.

```
java.lang.Boolean.RRSTransactional=true;

java.lang.Boolean getRRSTransactional(){
```

```
    // Determine if the adapter can run RRSTransactional based
   // on it's configuration, and set the RRSTransactional property
   // appropriately to true or false.
 return RRSTransactional;
}
```

RRS support is only applicable in a local environment, where the backend must reside on the system. CICS and IMS resources adapters may use **RRSTransactional** support only when these adapters are configured to use local interfaces to their backend resource manager, which as stated above must reside on the same system as the IBM WebSphere Application Server for z/OS. These adapters are also capable of being configured to a remote instance of their backend resource manager. In this case, the adapters will respond ″false″ when the getRRSTransactional() method is invoked and instead of running as RRSTransactional will use whichever one of the three types of J2EE Transaction support they have chosen to support.

WebSphere relational resource adapter settings

Use this page to view the default WebSphere relational resource adapter settings.

This is the WebSphere-provided relational resource adapter for handling data access to any relational data base. This adapter is preinstalled by the WebSphere Application Server. Although the default relational resource adapter settings are viewable, you cannot make changes to them.

To view this administrative console page, click **Resources** > **Resource Adapters** > **WebSphere Relational Resource Adapter**.

Scope

Specifies the level to which this resource definition is visible -- the cell, node, or server level.

Resources such as JDBC Providers, Namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

Note that no matter what the scope of a defined resource, the resource's properties only apply at an individual server level. For example, if you define the scope of a data source at the Cell level, all users in that Cell can look up and use that data source, which is unique within that Cell. However, resource property settings are local to each server in the Cell. For example, if you define *max connections* to 10, then each server in that Cell can have 10 connections.

Cell The most general scope. Resources defined at the Cell scope are visible from all Nodes and servers, unless they are overridden. To view resources defined in the cell scope, do not specify a server or a node name in the scope selection form.

Node The default scope for most resource types. Resources defined at the Node scope override any duplicates defined at the Cell scope and are visible to all servers on the same node, unless they are overridden at a server scope on that node. To view resources defined in a node scope, do not specify a server, but select a node name in the scope selection form.

Server The most specific scope for defining resources. Resources defined at the Server scope override any duplicate resource definitions defined at the Cell scope or parent Node scope and are visible only to a specific server. To view resources defined in a server scope, specify a server name as well as a node name in the scope selection form.

When resources are created, they are always created into the current scope selected in the panel. To view resources in other scopes, specify a different node or server in the scope selection form.

Data type String

Name
Specifies the name of the resource provider.

Data type String

Description
Specifies a description of the relational resource adapter.

Data type String

Archive path
Specifies the path to the Resource Adapter Archive (RAR) file containing the module for this resource adapter.

Data type String

Classpath
Specifies a list of paths or Java Archive (JAR) file names, which together form the location for the resource provider classes.

Data type String

Native path
Specifies a list of paths that forms the location for the resource provider native libraries.

Data type String

Data access portability features
The WebSphere Application Server relational resource adapter (RRA) provides a portability feature that enables applications to access data from different databases without changing the application. In addition, WebSphere Application Server enables you to plug in a data source that is not supported by WebSphere persistence. However, the data source *must* be implemented as either the *XADataSource* or the *ConnectionPoolDataSource*, and it must be in compliance with the JDBC 2.x specification.

You can achieve application portability through the following:

DataStoreHelper interface
With this interface, each data store platform can plug in its own private data store specific functions that the relational resource adapter run time uses. WebSphere Application Server provides an implementation for each supported JDBC provider.

In addition, the interface also provides a GenericDataStoreHelper class for unsupported data sources to use. You can subclass the GenericDataStoreHelper or other WebSphere provided helpers to support any new data source.

For more information, see the Javadoc **DataStoreHelper** in the Javadoc index.

The following code segment shows how a new data store helper is created to add two new error mappings for an unsupported data source.

```
public class NewDSHelper extends GenericDataStoreHelper
{
  public NewDSHelper()
  {
    super(null);
    java.util.Hashtable myErrorMap = null;
    myErrorMap = new java.util.Hashtable(2);
    myErrorMap.put(new Integer(-803), myDuplicateKeyException.class);
    myErrorMap.put(new Integer(-1015), myStaleConnectionException.class);
    myErrorMap.put("S1000", MyTableNotFoundException.class);
    setUserDefinedMap(myErrorMap);
    ...
  }
}
```

WSCallHelper class

With this class, applications can invoke any JDBC object proprietary methods that are not defined through the administrative console or standard APIs. This helper also enables applications to invoke many non-JDBC object methods.

All methods are static: see Javadoc **WSCallHelper** in the Javadoc index.

The following code segment illustrates using this helper class (with a DB2 data source):

```
Connection conn = ds.getConnection();
// get connection attribute
String connectionAttribute =(String) WSCallHelper.jdbcCall(DataSource.class, ds,
 "getConnectionAttribute", null, null);
// setAutoClose to false
WSCallHelper.jdbcCall(java.sql.Connection.class,
conn, "setAutoClose",
new Object[] { new Boolean(false)},
new Class[] { boolean.class });
// get data store helper
DataStoreHelper dshelper = WSCallHelper.getDataStoreHelper(ds);
```

Example: Developing your own DataStoreHelper class

The DataStoreHelper interface supports each data store platform plugging in its own private data store specific functions that are used by the Relational Resource Adapter run time.

```
package com.ibm.websphere.examples.adapter;

import java.sql.SQLException;
import javax.resource.ResourceException;

import com.ibm.websphere.appprofile.accessintent.AccessIntent;
import com.ibm.websphere.ce.cm.*;
import com.ibm.websphere.rsadapter.WSInteractionSpec;

/**
* Example DataStoreHelper class, demonstrating how to create a user-defined DataStoreHelper.
* Implementation for each method is provided only as an example.  More detail would likely be
* required for any custom DataStoreHelper created for use by a real application.
```

```
*/
public class ExampleDataStoreHelper extends com.ibm.websphere.rsadapter.GenericDataStoreHelper
{
    static final long serialVersionUID = 8788931090149908285L;


    public ExampleDataStoreHelper(java.util.Properties props)
    {
        super(props);

        // Update the DataStoreHelperMetaData values for this helper.
        getMetaData().setGetTypeMapSupport(false);

        // Update the exception mappings for this helper.
        java.util.Map xMap = new java.util.HashMap();

        // Add an Error Code mapping to StaleConnectionException.
        xMap.put(new Integer(2310),  StaleConnectionException.class);
        // Add an Error Code mapping to DuplicateKeyException.
        xMap.put(new Integer(1062),  DuplicateKeyException.class);
        // Add a SQL State mapping to the user-defined ColumnNotFoundException
        xMap.put("S0022",            ColumnNotFoundException.class);
        // Undo an inherited StaleConnection SQL State mapping.
        xMap.put("S1000",            Void.class);

        setUserDefinedMap(xMap);

        // Note: If you are extending a helper class, it is
        // normally not necessary to issue 'getMetaData().setHelperType(...)'
        // because your custom helper will inherit the helper type from its
        // parent class.  However, certain applications may need to differentiate
        // between a custom helper and an existing helper of the same type,
        // so WebSpehere has provided the value 'DataStoreHelper.CUSTOM_HELPER'
        // for this purpose.  If this functionality is needed by your application
        // insert the following line into your code:
        // getMetaData().setHelperType(DataStoreHelper.CUSTOM_HELPER);


    }


    public void doStatementCleanup(java.sql.PreparedStatement stmt) throws SQLException
    {
        // Clean up the statement so it may be cached and reused.

        stmt.setCursorName("");
        stmt.setEscapeProcessing(true);
        stmt.setFetchDirection(java.sql.ResultSet.FETCH_FORWARD);
        stmt.setMaxFieldSize(0);
        stmt.setMaxRows(0);
        stmt.setQueryTimeout(0);
    }


    public int getIsolationLevel(AccessIntent intent) throws ResourceException
    {
        // Determine an isolation level based on the AccessIntent.

        if (intent == null) return java.sql.Connection.TRANSACTION_SERIALIZABLE;

        return intent.getConcurrencyControl() == AccessIntent.CONCURRENCY_CONTROL_OPTIMISTIC ?
               java.sql.Connection.TRANSACTION_READ_COMMITTED :
               java.sql.Connection.TRANSACTION_REPEATABLE_READ;
    }
        public int getLockType(AccessIntent intent) {
           if ( intent.getConcurrencyControl() == AccessIntent.CONCURRENCY_CONTROL_PESSIMISTIC) {
              if ( intent.getAccessType() == AccessIntent.ACCESS_TYPE_READ ) {
```

```
                return WSInteractionSpec.LOCKTYPE_SELECT;
            }
            else {
                return WSInteractionSpec.LOCKTYPE_SELECT_FOR_UPDATE;
             }
        }
        return WSInteractionSpec.LOCKTYPE_SELECT;
    }


    public int getResultSetConcurrency(AccessIntent intent) throws ResourceException
    {
        // Determine a ResultSet concurrency based on the AccessIntent.

        return intent == null || intent.getAccessType() == AccessIntent.ACCESS_TYPE_READ ?
               java.sql.ResultSet.CONCUR_READ_ONLY :
               java.sql.ResultSet.CONCUR_UPDATABLE;
    }


    public int getResultSetType(AccessIntent intent) throws ResourceException
    {
        // Determine a ResultSet type based on the AccessIntent.

        if (intent == null) return java.sql.ResultSet.TYPE_SCROLL_INSENSITIVE;

        return intent.getCollectionAccess() == AccessIntent.COLLECTION_ACCESS_SERIAL ?
               java.sql.ResultSet.TYPE_FORWARD_ONLY :
               java.sql.ResultSet.TYPE_SCROLL_SENSITIVE;
    }
}
```

ColumnNotFoundException

```
package com.ibm.websphere.examples.adapter;

import java.sql.SQLException;
import com.ibm.websphere.ce.cm.PortableSQLException;

/**
* Example PortableSQLException subclass, which demonstrates how to create a user-defined
* exception for exception mapping.
*/
public class ColumnNotFoundException extends PortableSQLException
{
    public ColumnNotFoundException(SQLException sqlX)
    {
        super(sqlX);
    }
}
```

Connection factory

An application component uses a *connection factory* to access a connection instance, which the component then uses to connect to the underlying enterprise information system (EIS).

Examples of connections include database connections, Java Message Service connections, and SAP R/3 connections.

CMP Connection Factories collection

Use this page to view existing CMP connection factories settings.

These are the connection factories used by a container-managed persistence (CMP) bean to access any backend data store. A CMP Connection Factory is used by EJB model 2.0 Entities with CMP version 2.x. Connection factories listed on this page are created automatically under the WebSphere Relational Resource Adapter when you check the box *Use this DataSource in container managed persistence (CMP)* in the General Properties area on the Data Source page. You cannot modify or delete automatically created connection factories.

To view this administrative console page, click **Resources** >**Resource Adapters** >*WebSphere Relational Resource Adapter* > **CMP Connection Factories**.

Name
Specifies a list of the display names for the resources.

Data type String

JNDI Name
Specifies the JNDI name of the resource.

Data type String

Description
Specifies a description for the resource.

Data type String

Category
Specifies a category string which can be used to classify or group the resource.

Data type String

CMP connection factory settings
Use this page to view the settings of a connection factory that is used by a CMP bean to access any backend data store. This connection factory is only in ″read″ mode. It cannot be modified or deleted.

To view this administrative console page, click **Resources** >**Resource Adapters** > *WebSphere Relational Resource Adapter*> **CMP Connection Factories** > *connection_factory*

Scope:

Specifies the level to which this resource definition is visible -- the cell, node, or server level.

Resources such as JDBC Providers, Namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

Note that no matter what the scope of a defined resource, the resource's properties only apply at an individual server level. For example, if you define the scope of a data source at the Cell level, all users in that Cell can look up and use that data source, which is unique within that Cell. However, resource property settings are

local to each server in the Cell. For example, if you define *max connections* to 10, then each server in that Cell can have 10 connections.

Cell The most general scope. Resources defined at the Cell scope are visible from all Nodes and servers, unless they are overridden. To view resources defined in the cell scope, do not specify a server or a node name in the scope selection form.

Node The default scope for most resource types. Resources defined at the Node scope override any duplicates defined at the Cell scope and are visible to all servers on the same node, unless they are overridden at a server scope on that node. To view resources defined in a node scope, do not specify a server, but select a node name in the scope selection form.

Server The most specific scope for defining resources. Resources defined at the Server scope override any duplicate resource definitions defined at the Cell scope or parent Node scope and are visible only to a specific server. To view resources defined in a server scope, specify a server name as well as a node name in the scope selection form.

When resources are created, they are always created into the current scope selected in the panel. To view resources in other scopes, specify a different node or server in the scope selection form.

Data type String

Name:

Specifies the display name for the resource.

Data type String

JNDI name:

Specifies the JNDI name of the resource.

Data type String

Description:

Specifies a description for the resource.

Data type String

Category:

Specifies a category string which can be used to classify or group the resource.

Data type String

Authentication Preference:

Specifies which of the authentication mechanisms that are defined for the corresponding resource adapter applies to this connection factory.

For example, if two authentication mechanism entries are defined for a resource adapter (*KerbV5* and *Basic Password*), this specifies one of those two types. If the authentication mechanism preference specified is not an authentication mechanism available on the corresponding resource adapter, it is ignored.

Data type String

Component-managed authentication alias:

References authentication data for component-managed signon to the resource.

Data type Drop-down list

Container-managed authentication alias:

References authentication data for container-managed signon to the resource.

Data type Drop-down list

JDBC providers

Installed applications use JDBC providers to access data from databases.

The JDBC provider and data source together are functionally equivalent to the J2EE Connector Architecture (JCA) Connection Factory. The WebSphere Application Server prerequisite Web site has a current list of supported providers.

DB2 Universal JDBC Driver Support

Support for DB2 Universal JDBC Driver in WebSphere Application Server for z/OS release v5.0 and beyond.

WebSphere Application Server for z/OS supports the DB2 Universal JDBC Driver. The capabilities available depend on the DB2 Universal JDBC Driver that you installed as follows:

- The z/OS Application Connectivity to DB2 for z/OS feature that provides DB2 Universal JDBC Driver Type 4 connectivity. This DB2 Universal JDBC Drive can be invoked only as a type 4 driver for z/OS. As a type 4 driver, it uses a communication protocol to communicate requests from a z/OS application to a remote DB2 database.

  When you install and configure this driver for WebSphere Application Server for z/OS, it permits your applications to use JDBC or Container Managed Persistence (CMP) support to access backend DB2 databases (DB2 V7 and up) residing on z/OS at any location. All global transactions are handled as J2EE XA transactions.
- The DB2 Universal JDBC Driver in DB2 UDB for z/OS Version 8. This driver provides both Type 2 and Type 4 support.

  Type 4 driver support uses a communication protocol to communicate requests from a z/OS application to a remote DB2 database. This driver supports using J2EE XA transaction processing to process global transactions.

  Type 2 driver support uses local API protocol to communicate requests from a z/OS application to a target DB2 running on the same z/OS system image as the application. When the Type 2 driver is used under z/OS, the driver supports

the use of z/OS Resource Recovery Services (RRS) to coordinate global transactions across multiple resource manages using 2-phase commit processing.

When you install and configure this version of the driver, your applications can use JDBC or CMP support to access backend DB2 databases (DB2 V7 and up). These databases can reside on the same z/OS system image, or on a different z/OS system image. depending on the driver type used. Type 2 driver handles all global transactions as RRS-coordinated global transactions.

- The DB2 Universal JDBC Driver Provider by APAR PQ80841 on DB2 UDB for OS/390 and z/OS Version 7. This version provides both driver Type 2 and driver Type 4 support.

  Type 4 driver support uses a communication protocol to communicate requests from a z/OS application to a remote DB2 database. This driver supports using J2EE XA transaction processing to process global transactions.

  Type 2 driver support uses local API protocol to communicate requests from a z/OS application to a target DB2 running on the same z/OS system image as the application. When the Type 2 driver is used under z/OS, the driver supports the use of z/OS Resource Recovery Services (RRS) to coordinate global transactions across multiple resource manages using 2-phase commit processing.

  When you install and configure this version of the driver, your applications can use JDBC or CMP support to access backend DB2 databases (DB2 V7 and up). These databases can reside on the same z/OS system image, or on a different z/OS system image, depending on the driver type used.

WebSphere Application Server for z/OS DB2 JDBC Providers

The following JDBC provider is for use with the DB2 for OS/390 and z/OS Legacy JDBC Driver:

- DB2 for zOS Local JDBC Provider (RRS)

Following is a list of the JDBC providers for the DB2 Universal JDBC driver.

- DB2 Universal JDBC Driver Provider
- DB2 Universal JDBC Driver Provider (XA)

DB2 for zOS Local JDBC Provider (RRS)

The DB2 for zOS JDBC Provider (RRS) is for use with the DB2 for 390 and z/OS Legacy JDBC Driver. It can be used only with WebSphere Application Server for z/OS. This provider supports the creation of WebSphere Application Server for z/OS v5.0 and v4.0 datasources. Also, this provider uses z/OS Resource Recovery Services (RRS) to coordinate transactions across multiple resource managers using two-phase commit processing.

The DB2 for zOS Local JDBC Provider (RRS) allows applications to use both JDBC and Structured Query Language in Java (SQLJ) access to DB2 databases. Use of SQLJ with Container Managed Persistence (CMP) is not supported under this provider.

- **Provider Requirements**

  The configuration information is provided in a template for the DB2 for zOS Local JDBC Driver Provider (RRS) and is automatically filled in when you select this provider.

  The requirements for the DB2 for zOS Local JDBC Driver Provider (RRS) are:

  1. The following DB2 for z/OS Legacy JDBC Driver files on the CLASSPATH:

     ```
     ${DB2390_JDBC_DRIVER_PATH}/classes/db2j2classes.zip
     ```

2. The following LIBPATH, which is the fully-qualified path of the directory that contains the native files (.so type files) required by the DB2 for OS/390 and zOS Legacy JDBC Driver:

   ```
   ${DB2390_JDBC_DRIVER_PATH}/lib
   ```

3. The following DB2 datasource implementation class, which supports the use of RRS running under WebSphere Application Server for z/OS and allows WebSphere Application Server for z/OS to perform connection pooling:

   ```
   com.ibm.db2.jcc.DB2ConnectionPoolDataSource
   ```

- **DataSource Requirements**

  The minimum configuration requirements for each datasource defined by this provider are:

  1. A name for the datasource definition, which you must specify when you configure the datasource.
  2. The following DataStoreHelper class that is associated with the datasource.

     ```
     com.ibm.websphere.rsadapter.DB2DataStoreHelper
     ```

  3. A valid authentication alias. Where res-auth CONTAINER is used, it is permissible to not specify any authentication alias. In this case, the user identity associated with a connection created by the datasource is the user identity associated with the current thread at the time a request for a connection is issued.
  4. The following basic set of properties (at a minimum):

     databaseName
     The location name of the target database, used when establishing connections using this datasource.

DB2 Universal JDBC Driver Provider

The DB2 Universal JDBC Driver Provider is a non-XA JDBC provider that uses the DB2 Universal JDBC Driver to provide access to DB2 databases. The Universal JDBC Driver supports Java communication-based connectivity (driver Type 4), which allows distributed access to DB2. Additionally, the driver supports Java Native Interface (JNI) based connectivity (driver Type 2), which allows local access to DB2.

Note: Under WebSphere Application Server for z/OS, driver Type 4 access to DB2 is supported only for connectivity to DB2 for z/OS (Version 7 and up) databases on the z/OS platform.

The DB2 Universal JDBC Driver Provider allows applications to use both JDBC and Structured Query Language in Java (SQLJ) access to DB2 databases.

To use this provider, you must have the DB2 Universal JDBC Driver for DB2 Version 7 or DB2 Version 8 installed and configured for WebSphere Application Server for z/OS.

- **Provider Requirements**

  The configuration information is provided in a template for the DB2 Universal JDBC Driver Provider and is automatically filled in when you select this provider.

  The requirements for the DB2 Universal JDBC Driver Provider are:

  1. The following DB2 Universal JDBC Driver files on the CLASSPATH:

```
${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc.jar
${UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_License_cu.jar
${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_License_cisuz.jar
```

db2jcc.jar This is the DB2 Universal JDBC Driver jar file. After the DB2 installation, this jar file is located in DB2's install directory. The fully-qualified path of this jar must be specified as the value of the DB2UNIVERSAL_JDBC_DRIVER_PATH environment variable.

db2jcc_License_cu.jar This is the DB2 Universal JDBC driver license file that allows access to DB2 Universal databases under Cloudscape and workstations. It is not used for WebSphere Application Server for z/OS, but is included to make the provider definition common between WebSphere Application Server for z/OS and WebSphere Application Server Distributed.

db2jcc_License_cisuz.jar This is the DB2 Universal JDBC Driver license file that allows access to DB2 Universal databases under Cloudscape, workstations, and z/OS. After you install DB2, this jar file appears in the same DB2 directory as db2jcc.jar.

- The following LIBPATH, which is the fully-qualified path of the DB2 java directory that contains the native files (.so type files) needed by the DB2 Universal JDBC Driver in WebSphere Application Server for z/OS.

  ```
  ${DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH}
  ```

- The following DB2 datasource implementation class. This datasource implementation class performs only one-phase commit processing, except where driver Type 2 is specified in WebSphere Application Server for z/OS. In this case, RRS coordinates transaction processing and two-phase commit processing is performed for global transactions.

  ```
  com.ibm.db2.jcc.DB2ConnectionPoolDataSource
  ```

- **DataSource Requirements**

  The minimum configuration requirements for each datasource defined by this provider are:

  1. A name for the datasource definition.
  2. The following DataStoreHelper class that is associated with the datasource.

     ```
     com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper
     ```

  3. A valid authentication alias if the driverType property (see properties below) is set to 4. If the driverType property is set to 2, a component-managed authentication alias must be specified to use the datasource with res-auth APPLICATION. In the case where driverType 2 is specified and the datasource is used with res-auth CONTAINER, you can specify a container-managed authentication alias; however, it is not required. If you do not specify a container-managed alias, the user identity associated with a connection created by the datasource will be the user identity associated with the current thread at the time the connection is obtained.
  4. The following basic set of properties (at a minimum):

     databaseName
     The location name of the target database, used when establishing connections using this datasource.

     driverType
     The JDBC connectivity type used by the datasource. There are two permitted values: 2 and 4. If you want to use Universal JDBC Type 2 driver, set this value to 2. If you want to use Universal JDBC Type 4 driver, set this value to 4.

     ServerName
     The TCP/IP address or host name for the Distributed Relational

Database Architecture (DRDA) server. This is required only if your driverType is set to 4. This property is not required for a driverType of 2.

portNumber
The TCP/IP port number where the DRDA server resides. Specify a value only if your driverType is set to 4. This property is not required for a driverType of 2.

DB2 Universal JDBC Driver Provider (XA)

This provider is the XA DB2 Universal JDBC Driver Provider that uses the DB2 Universal JDBC Driver to provide access to DB2 databases. The Universal JDBC Driver supports Java communication-based connectivity (driver Type 4), which allows distributed access to DB2. The driver also supports Java Native Interface (JNI) based connectivity (driver Type 2), which allows local access to DB2; however, in the case of the XA support, driver Type 2 is not supported by the DB2 Universal JDBC Driver on WebSphere Application Server for z/OS. Driver Type 2 should not be used when defining a datasource under this provider.

Note: Under WebSphere Application Server for z/OS, driver Type 4 access to DB2 is supported only for connectivity to DB2 for z/OS (Version 7 and up) on the z/OS platform.

The J2EE XA transaction architecture is supported by the universal JDBC driver under this provider. This permits the coordination of global transactions across multiple resource managers using two-phase commit processing. The driver also supports one-phase transaction processing under this provider.

The DB2 Universal JDBC Driver Provider (XA) allows applications to use both JDBC and Structured Query Language in Java (SQLJ) access to DB2 databases. SQLJ use with CMP is also supported.

This provider supports only the creation of WebSphere Application Server for z/OS v5.0 datasources. The creation of 4.0 datasources is not supported.

To use this provider, you must have the DB2 Universal JDBC Driver for DB2 Version 7 or DB2 Version 8 installed and configured for WebSphere Application Server for z/OS, or you must have the z/OS Application Connectivity to DB2 for z/OS feature installed and configured for WebSphere Application Server for z/OS.

- **Provider Requirements**

  The configuration information is provided in a template for the DB2 Universal JDBC Driver Provider (XA) and is automatically filled in when you select this provider.

  The requirements for the DB2 Universal JDBC Driver Provider (XA) are:

  1. The following DB2 Universal JDBC Driver files on the CLASSPATH:

     ```
     ${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc.jar
     ${UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_License_cu.jar
     ${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_License_cisuz.jar
     ```

     db2jcc.jar This is the DB2 Universal JDBC Driver jar file. After the DB2 installation, this jar file is located in DB2's install directory. The fully-qualified path of this jar must be specified as the value of the DB2UNIVERSAL_JDBC_DRIVER_PATH environment variable.

     db2jcc_License_cu.jar This is the DB2 Universal JDBC driver license file that allows access to DB2 Universal databases under Cloudscape and

workstations. It is not used for WebSphere Application Server for z/OS, but is included to make the provider definition common between WebSphere Application Server for z/OS and WebSphere Application Server Distributed.

db2jcc_License_cisuz.jar This is the DB2 Universal JDBC Driver license file that allows access to DB2 Universal databases under Cloudscape, workstations, and z/OS. After you install DB2, this jar file appears in the same DB2 directory as db2jcc.jar.

2. The following LIBPATH, which is the fully-qualified path of the DB2 java directory that contains the native files (.so type files) needed by the DB2 Universal JDBC Driver in WebSphere Application Server for z/OS. If no native files are required, set the DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH to null.

   ```
   ${DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH}
   ```

3. The following DB2 datasource implementation class, which supports J2EE XA transaction processing.

   ```
   com.ibm.db2.jcc.DB2XADataSource
   ```

- **DataSource Requirements**

  The minimum configuration requirements for each datasource defined by this provider are:

  1. A name for the datasource definition.
  2. The following DataStoreHelper class that is associated with the datasource.

     ```
     com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper
     ```

  3. A valid authentication alias.
  4. The following basic set of properties (at a minimum):

     databaseName
     The location name of the target database, used when establishing connections using this datasource.

     driverType
     The JDBC connectivity type used by the datasource. For WebSphere Application Server for z/OS, use only driverType 4. Use of driverType 2 is not supported.

     ServerName
     The TCP/IP address or host name for the Distributed Relational Database Architecture (DRDA) server.

     portNumber
     The TCP/IP port number where the DRDA server resides.

For more information on DB2 for z/OS, visit the DB2 Web site at: http://www.ibm.com/software/data/db2/.

Provider coexistence considerations

Following are provider coexistence possibilities.

DB2 Legacy JDBC Providers and DB2 Universal JDBC Driver Providers

- Under WebSphere Application Server for z/OS, JDBC Provider definitions that use the Legacy DB2 for OS/390 and z/OS JDBC Driver (db2j2classes.zip) and JDBC Provider definitions that use the new DB2 Universal JDBC Driver (db2jcc.jar) must be carefully configured to ensure they never coexist on the same server. This is because some of the same class names are used in both drivers and these duplicate classes are functionally different. Adding jar files for

the two types of drivers to the same CLASSPATH causes unpredictable results, since incorrect classes will be used for the provider whose CLASSPATH definition is added last.

- Carefully define the scope in which the two different types of driver providers are used. If you define one provider type under one scope (cell, node, or server), and the other provider type under another scope, separation is not ensured if the two scopes include the same server.

  For example, if you define a DB2 for zOS Local JDBC Provider (RRS) at a node level, then define a DB2 Universal JDBC Driver Provider at a server level where the server is in the same node, the provider definition at the node level is propagated down to the server level. As a result, a conflict occurs between the JDBC drivers used by the two providers.

  To help you understand which WebSphere Application Server for z/OS providers cannot coexist together, the providers are listed below under the DB2 JDBC driver that it uses:

  1. Providers that use the DB2 for z/OS Legacy JDBC Driver
     - DB2 for zOS Local JDBC Provider (RRS)
  2. Providers that use the DB2 Universal JDBC Driver
     - DB2 Universal JDBC Driver Provider
     - DB2 Universal JDBC Driver Provider (XA)
     - Cloudscape Network Server Using Universal JDBC Driver Provider

DB2 Drivers and Cloudscape

- The Cloudscape Network Server Using Universal JDBC Driver Provider uses an embedded copy of the DB2 Universal JDBC Driver that is shipped with Cloudscape. This provider is configured to automatically use the new level of the DB2 Universal JDBC Driver when installing the DB2 Version 7, Version 8, or standalone Type 4 version of the DB2 Universal JDBC Driver on the system where WebSphere Application Server for z/OS is configured.

  The Cloudscape Network Server Using Universal JDBC Driver Provider can coexist on the same server as the DB2 for Universal JDBC Driver Providers since it uses the same DB2 JDBC driver.
- The Cloudscape Network Server Using Universal JDBC Driver Provider cannot coexist in the same server as the DB2 for zOS Local JDBC Provider (RRS), because the Cloudscape provider uses the DB2 Universal JDBC Driver and the DB2 for zOS Local JDBC Provider (RRS) uses the DB2 for OS/390 and z/OS Legacy JDBC Driver. These drivers conflict with one another.

Using a DB2 Universal JDBC Driver Provider with WebSphere Application Server for z/OS

Make sure that DB2 Universal JDBC Driver on z/OS and all the files required by WebSphere Application Server for z/OS are installed and available in an HFS directory before you continue with this configuration.

To use DB2 Universal JDBC Driver with WebSphere Application Server for z/OS, one of the following versions of the driver must be installed:

- The DB2 Universal JDBC Driver in DB2 UDB for z/OS Version 8. This version supports both driver Types 2 and 4.
- The DB2 Universal JDBC Driver Provider by APAR PQ80841 on DB2 UDB for OS/390 and z/OS Version 7. This version supports both driver Types 2 and 4.

- The z/OS Application Connectivity to DB2 for z/OS feature that provides DB2 Universal JDBC Driver Type 4 connectivity. This version supports only driver Type 4 connectivity to DB2 databases. If this driver is installed, only the DB2 Universal JDBC Driver Provider (XA) can be used on WebSphere Application Server for z/OS to access remote DB2 databases.

Refer to the APARs and DB2 product installation information for details regarding the installation of the above versions of the DB2 Universal JDBC Driver.

To use a DB2 Universal JDBC Driver with WebSphere Application Server for z/OS, you must:

- Configure the DB2 Universal JDBC Driver.
- Define a DB2 Universal JDBC Driver Provider for WebSphere Application Server for z/OS.
- Define a DB2 Universal JDBC Driver Provider DataSource.

1. Configure the DB2 Universal JDBC Driver for WebSphere Application Server for z/OS
   a. Define a DB2 Universal JDBC Driver Provider

      Before you create a JDBC provider for the DB2 Universal JDBC Driver for z/OS, WebSphere Application Server for z/OS must know the location of the installed DB2 Universal JDBC Driver and license file, and the location of any native files that might be required by the DB2 Universal JDBC Driver. To do this, from the WebSphere Application Server for z/OS Administrative Console, go to Environment > Manage WebSphere Variables, and update the values of the following environment variables:

      1) **DB2UNIVERSAL_JDBC_DRIVER_PATH**

         Specify the fully-qualified path of the directory that contains the DB2 Universal JDBC Driver. This must be the directory that contains the db2jcc.jar and the db2jcc_license_cisuz.jar.

         Example: If the fully-qualified path of the db2jcc.jar is /usr/lpp/db2810/jcc/classes/db2jcc.jar, specify /usr/lpp/db2810/jcc/classes as the value of the variable.

      2) **DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH**

         Specify the fully-qualified directory path of the directory that contains the DB2 Universal JDBC Driver native files, if necessary. This is the directory that contains the driver files that have a .so file type. If the driver version you are using does not require native files, leave this value at null.

         Example: If the fully-qualified path of the directory containing the native files is /usr/lpp/db2810/jcc/lib, specify /usr/lpp/db2810/jcc/lib as the value of the variable.

   b. Bind the required DB2 packages

      As with any application that executes SQL statements in DB2 for z/OS, the Universal JDBC driver must first bind with DB2 the packages that represent the SQL statements to be executed. The Universal JDBC Driver does not use the same packages used by the legacy JDBC driver, and uses a different process for binding its packages.

      The specific details of the bind utility and bind process are described by the README provided with the installed DB2 Universal JDBC Driver. Refer to this README for details on how to setup and perform the required binding.

Also note that the utility requires the server name (or IP address), the port number, and the database name (the database location on z/OS) for the target DB2. To get this information, issue a DB2 **-DISPLAY DDF** command on the target DB2 system. This displays the IPADDR (IP address), the SQL DOMAIN (server name), the TCPPORT number, and the LOCATION (database name/location) for you to use as input to the utility.

You must perform the bind process for each target DB2 that will be accessed using the DB2 Universal JDBC Driver.

c. Set up to handle in-doubt transactions

You must perform this setup once for each target DB2 for z/OS Version 7 location that is accessed using the DB2 Universal JDBC Driver Type 4 XA support.

Since DB2 for z/OS Version 7 does not implement J2EE XA support, the Type 4 driver XA processing uses DB2 V7 two-phase commit protocol and a table in each location (database) to store a list of global transactions that are in doubt (finished but not committed).

This table must be set up at each DB2 V7 location that is accessed. To do this, use the In-Doubt Utility, which is included as part of the installed DB2 Universal JDBC Driver. Use this utility to create the SYSIBM.INDOUBT Table that stores information about In-Doubt Global Transactions. This utility also binds the package T4XAIndbtPkg, which contains the SQL statements to insert and delete from the SYSIBM.INDOUBT Table. The T4XAIndbtPkg package is written with SQLJ.

The specific details of the In-Doubt utility are described by the README that is provided with the installed DB2 Universal JDBC Driver. Refer to this README for details on how to setup and perform the execution of the In-Doubt utility.

This installation process requires that the target DB2 subsystem be configured with DDF enabled for incoming TCP/IP connections.

1) To enable DDF on the target DB2, issue the DB2 **-START DDF** command on that system.
2) This utility requires the server name (or IP address) and the port number for the target DB2 V7. To obtain this information, issue a DB2 **-DISPLAY DDF** command on the target DB2 V7 system. This displays the IPADDR (IP address), the SQL DOMAIN (server name), and the TCPPORT number that can be used as input to the utility.

Note: This setup is ***not*** a requirement for DB2 FOR z/OS Version 8 servers because DB2 FOR z/OS Version 8 natively supports XA commands over DRDA and manages the In-Doubt Global Transactions internally.

d. Define a db2.jcc.propertiesFile

A db2.jcc.propertiesFile for use by DB2 Universal JDBC Driver Type 2 processing under WebSphere Application Server for z/OS can be created and specified as input to the driver. This runtime properties file is for use in specifying various runtime options that the DB2 Universal JDBC Driver uses for Type 2 connectivity. These options are specified as properties in the form of parameter=value. Refer to the README file packaged with the installed DB2 Universal JDBC Driver for a detailed description of each of the properties.

This file is not required; however, if it is not provided, universal driver default processing is performed.

Of specific interest is the db2.jcc.ssid property. This property specifies the DB2 subsystem identifier (not location name), to be used by the DB2

Universal JDBC Driver Type 2 processing as the local subsystem name to which it should connect. If this property is not provided, the driver uses the subsystem identifier that it finds in the DSNHDECP load module. If the installation wants to use the DSNHDECP load module to specify the subsystem identifier, this load module must be included in a steplib dataset in the servant region PROCs associated with each server that will use the DB2 identified by the subsystem ID. Refer to the README file packaged with the universal driver for more information on using this load module. If that DSNHDECP load module does not accurately reflect the desired subsystem, or if multiple subsystems might be using a generic DSNHDECP, the db2.jcc.ssid property must be specified.

Although the db2.jcc.propertiesFile is not required, if you choose to define the file, you must specify the fully qualified-hfs-filename. To do this, specify the file as a JVM System property as follows:

- **db2.jcc.propertiesFile = <fully-qualified-hfs-filename>**

Because the driver-general properties are typically specific to a driver load (for example, server) versus to all servers using the JDBC provider, it is best that this JVM property be set at the server level. To define the **db2.jcc.propertiesFile=** property to the server level using the WebSphere Application Server for z/OS Administrative Console:

1) Under the WebSphere Application Server for z/OS Administrative Console, go to Servers > Application Servers, then click the server to which you want to add the JVM property.
2) From the selected server page, go to Process Definition > Servant.
3) On the Servant page, scroll down to the Additional Properties at the bottom of the page, then click Java Virtual Machine.
4) On the Java Virtual Machine page, scroll down to Additional Properties at the bottom of the page, then click Custom Properties.
5) On the Custom Properties page, scroll down to New to configure a new JVM property for the selected server. The name of the property is **db2.jcc.propertiesFile**. The value of the property is the fully-qualified-hfs-filename that you created and initialized with the DB2 Universal JDBC Driver properties. These are the properties that you want the Type 2 driver to use for the selected server
6) Click OK.
7) Click Save to save the new JVM property.

2. Define the DB2 Universal JDBC Driver Provider.

   After the DB2 Universal JDBC Driver is configured for WebSphere Application Server for z/OS, configure a DB2 Universal JDBC Driver Provider, which is non-XA and/or a DB2 Universal JDBC Driver Provider (XA), which supports XA. In doing so, it is important to note that if you use both the DB2 Legacy JDBC Driver and the DB2 Universal JDBC Driver under WebSphere Application Server for z/OS, you must ensure that DB2 JDBC providers associated with these two drivers are not located on the same server. (Refer to Provider Coexistence Considerations).

   IBM suggests that you define your DB2 Universal JDBC Driver providers at a server level to reduce the chance of conflict with the DB2 for zOS Local JDBC Provider (RRS) that uses the DB2 Legacy JDBC Driver. Likewise, any DB2 for z/OS Local JDBC Provider you have defined must be defined at the server level to avoid conflict.

   To define a DB2 Universal JDBC Provider:

a. From the WebSphere Application Server for z/OS Administrative Console, click Resources > JDBC Providers.
b. On the JDBC Provider page, set the JDBC Provider scope to the server upon which you want to install the new provider.
c. Click Apply.
d. Click New.
e. In the list of JDBC providers that displays, choose the type of DB2 Universal JDBC Driver Provider as follows:
   - **DB2 Universal JDBC Driver Provider**

     This provider supports driver Types 2 and 4. It does not support J2EE XA transaction processing. In the case of driverType 2 processing, RRS is used to coordinate transaction processing and manage global transaction using two-phase commit. In the case of driverType 4, one-phase commit processing is used to manage transactions.

     Note: Do not select this provider if your installation has the z/OS Application Connectivity to DB2 for z/OS feature defined to WebSphere Application Server for z/OS. This feature provides only universal driver type 4 XA connectivity, which is not supported by the non-XA DB2 Universal JDBC Driver Provider.
   - **DB2 Universal JDBC Driver Provider (XA)**

     This provider supports only driverType 4. It uses J2EE XA to manage global transactions across multiple resource managers and to perform two-phase commit processing.
f. A configuration view of the selected provider displays, showing the default name of the provider, the classpath, the native library path, and the datasource implementation classname used by the provider. With the exception of the provider name, typically none of the information changes. If you choose, you can type your name for the provider in the Name field.
g. When the Provider definition is complete, click Apply.
h. Finally, click Save to save the new JDBC provider.

3. Defining a DB2 Universal JDBC Driver Provider datasource

   To specify a data source for the defined DB2 Universal JDBC Driver Provider:

   a. From the WebSphere Application Server for z/OS Administrative Console, click Resources > JDBC Providers. On the JDBC Providers page that displays, select the DB2 Universal JDBC Driver Provider for which you will define a datasource.
   b. On the DB2 Universal JDBC Driver Provider page that displays, in the Additional Properties section at the bottom of the page, make a choice as follows:
      - Choose **DataSources** if you want to define a datasource for a DB2 Universal JDBC Driver Provider (XA). In this case, DataSources (Version 4) is not supported.
      - Choose **DataSources** or **DataSources (Version 4)** if you want to define a datasource for a DB2 Universal JDBC Driver Provider. This choice depends on the type of datasource you want to define.
   c. On the DataSources page that displays, click New.
   d. On the New page, complete the following fields as required for your Datasource:
      1) Name
      2) JNDI name

3) Component-managed Authentication Alias (optional)
4) Container-managed Authentication Alias (optional)
5) Mapping-Configuration alias (optional)
6) Indicate if you want this DataSource to be used for container managed persistence.

Note: If you set the driverType property for the datasource to 4, an appropriate managed Authentication Alias must be specified. If you set the driverType property for the datasource to 2 and no managed Authentication Alias is specified, the user identity currently associated with the thread at the time of a getConnection request is used as the identity associated with the connection.

e. Click Apply.

f. A page with the name of the specified datasource displays. Under Additional Properties at the bottom of this page, click Custom Properties.

g. In the Custom Properties page, specify the property settings you desire. At a minimum, the following datasource properties must be specified.

databaseName
The location name of the target database used when establishing connections with this datasource

driverType
The JDBC connectivity type used by the datasource. If you want to use a driverType 4, set the value to 4. If you want to use a driverType 2 , set the value to 2. If the datasource is for the DB2 Universal JDBC Driver Provider (XA), specify only driverType 4. Specification of driverType 2 in the case of the DB2 Universal JDBC Driver Provider (XA) is not supported.

ServerName
The TCP/IP address or host name for the Distributed Relational Database Architecture (DRDA) server. This property is required only if driverType is set to 4. This property is not used if driverType is set to 2.

PortNumber
This is the TCP/IP port number where the DRDA server resides. Provide a value for this property only if driverType is set to 4. This property is not used if driverType is set to 2.

h. After you finish specifying the desired properties, click Apply.

i. Click Save to save the new datasource.

Data sources

An application uses a *data source* to access the data from the database.

A data source is associated with a JDBC provider that supplies the specific JDBC driver implementation class. The data source represents the J2EE Connector Architecture (JCA) connection factory for the relational resource adapter.

You can create multiple data sources associated with the same JDBC provider. Each JDBC provider supports the interfaces defined by Sun Microsystems listed below, with the exception of the DB2 for z/OS Local JDBC Provider (RRS), which only

provides support for the ConnectionPoolDataSource implementation. These interfaces enable the application to run in a single-phase or two-phase transaction protocol.

- *ConnectionPoolDataSource* - a data source that supports application participation in all transactions, including two-phase commit transactions. When this kind of data source is involved in a global transaction, transaction recovery is not provided by the transaction manager. The application is responsible for providing the backup recovery process if multiple resource managers are involved.
- *XADataSource* - a data source that supports application participation in a single-phase or a global (two-phase) transaction environment. When this data source is involved in a global transaction, the transaction manager provides transaction recovery.

Previously, the function of data access was provided by a single connection manager (CM) architecture. This connection manager architecture remains available to support J2EE 1.2 applications, but a new connection manager architecture is provided, based on the JCA architecture supporting the new J2EE 1.3 application style.

These two separate CM architectures are represented by two types of data sources. To choose the right data source, administrators must understand the nature of their applications, EJB modules, and enterprise beans.

- Data source (Version 4.0) - this data source runs under the CM architecture. Applications using this data source behave as if they were running in Version 4.0.
- Data source - this data source uses the JCA standard architecture to provide J2EE 1.3 support. It runs under the JCA connection manager and the relational resource adapter. Applications using this type of data source might behave differently because of the J2EE 1.3 architecture.

Choice of data source

- J2EE 1.2 application - all enterprise beans, JDBC applications, or Servlets 2.2 components must use the **4.0** data source.
- J2EE 1.3 application -
  - EJB 1.1 Module - all EJB 1.x beans must use the **4.0** data source.
  - EJB 2.0 Module - enterprise beans that include container-managed persistence (CMP) Version 2.0 and 1.x must use the **new** data source.
  - JDBC applications and Servlet 2.3 - must use the **new** data source.

Data access beans

Data access beans provide a rich set of features and function, while hiding much of the complexity associated with accessing relational databases.

They are Java classes written to the JavaBeans Specification.

You can use the data access beans in JavaBeans-compliant tools, such as the IBM *WebSphere Studio Application Developer (WSAD)*. Because the data access beans are also Java classes, you can use them like ordinary classes.

The data access beans (in the package *com.ibm.db*) offer the following capabilities:

Feature

Details

Caching query results
You can retrieve SQL query results all at once and place them in a cache. Programs using the result set can move forward and backward through the cache or jump directly to any result row in the cache.

For large result sets, the data access beans provide ways to retrieve and manage *packets*, subsets of the complete result set.

Updating through result cache
Programs can use standard Java statements (rather than SQL statements) to change, add, or delete rows in the result cache. You can propagate changes to the cache in the underlying relational table.

Querying parameter support
The base SQL query is defined as a Java String, with parameters replacing some of the actual values. When the query runs, the data access beans provide a way to replace the parameters with values made available at run time. Default mappings for common data types are provided, but you can specify whatever your Java program and database require.

Supporting metadata
A *StatementMetaData* object contains the base SQL query. Information about the query (*metadata*) enables the object to pass parameters into the query as Java data types.

Metadata in the object maps Java data types to SQL data types (as well as the reverse). When the query runs, the Java-datatyped parameters are automatically converted to SQL data types as specified in the metadata mapping.

When results return, the metadata object automatically converts SQL data types back into the Java data types specified in the metadata mapping.

Connection management architecture

Note: WebSphere Application Server does not support JDBC 3.0.

The connection management architecture for both relational and procedural access to enterprise information systems (EIS) is based on the J2EE Connector Architecture (JCA) specification. The Connection Manager (CM), which pools and manages connections within an application server, is capable of managing connections obtained through both resource adapters (RAs) defined by the JCA specification, and DataSources defined by the JDBC 2.0 Extensions Specification.

To make DataSource connections manageable by this CM that works only with RAs, WebSphere Application Server Version 5.0 provides its own resource adapter. From the CM point of view, JDBC DataSources and JCA connection factories look the same. Users of DataSources do not experience any programmatic or behavioral differences in their applications because of the underlying JCA architecture. JDBC users still configure and use DataSources according to the JDBC programming model.

Applications migrating from previous versions of WebSphere Application Server might experience some behavioral differences because of the change from J2EE 1.2 requirements to J2EE 1.3 requirements. These differences are not related to the adoption of the JCA architecture.

If you have J2EE 1.2 applications using the JDBC API that you wish to run in WebSphere Application Server 5.0, the JDBC CM from Version 4.0 is still provided as a configuration option. Using this configuration option enables J2EE 1.2

applications to run unaltered. If you migrate a Version 4.0 application to Version 5.0, using the Version 5.0 migration tools, the application automatically uses the Version 4.0 connection manager after migration. However, EJB 2.0 modules in J2EE 1.3 applications cannot use the JDBC CM from Version 4.0.

Connection pooling

When accessing any database, the initial database connection is an expensive operation. *Connection pooling* enables administrators to establish a pool of database connections that applications can share on an application server. When connection pooling capabilities are used, performance improvements up to 20 times the normal results are realized. **WebSphere Application Server does not support JDBC 3.0.**

Each time a resource attempts to access a backend store (such as a database), the resource must connect to that data store. A connection requires resources to create, maintain, and then release the connection when it is no longer required.

The total data store overhead for an application is particularly high for Web-based applications because Web users connect and disconnect more frequently. In addition, user interactions are typically shorter. Often, more effort is spent connecting and disconnecting than is spent during the interactions. Also, because Internet requests can arrive from virtually anywhere, you can find usage volumes large and difficult to predict.

To help lessen these overhead problems, the WebSphere Application Server enables administrators to establish a pool of backend connections that applications can share on an application server. Connection pooling spreads the connection overhead across several user requests, thereby conserving resources for future requests.

WebSphere Application Server supports JDBC 2.0 Standard Extension APIs to provide support for connection pooling and connection reuse. The connection pool is used to direct JDBC calls within the application, as well as for enterprise beans using the database.

Each entity EJB transaction requires an additional connection to the database specifically to handle the transaction. Take this into account when calculating the number of data source connections.

On UNIX platforms, a separate DB2 process is created for each connection and these processes quickly affect performance on systems with low memory and cause errors.

If clones are used, one data pool exists for each clone. This is important when configuring the database maximum connections.

Benefits of connection pooling

Connection pooling can improve the response time of any application that requires connections, especially Web-based applications. When a user makes a request over the Web to a resource, the resource accesses a data source. With connection pooling, most user requests do not incur the overhead of creating a new connection because the data source can locate and use an existing connection from the pool of connections. When the request is satisfied and the response is returned to the user, the resource returns the connection to the connection pool for reuse. The overhead of a disconnect is avoided. Each user request incurs a fraction of the

cost for connecting or disconnecting. After the initial resources are used to produce the connections in the pool, additional overhead is insignificant because the existing connections are reused.

When to use connection pooling

Use WebSphere connection pooling in an application that meets any of the following criteria:

- It cannot tolerate the overhead of obtaining and releasing connections whenever a connection is used.
- It requires Java Transaction API (JTA) transactions within WebSphere Application Server.
- It needs to share connections among multiple users within the same transaction.
- It needs to take advantage of product features for managing local transactions within the application server.
- It does not manage the pooling of its own connections.
- It does not manage the specifics of creating a connection, such as the database name, user name, or password

How connections are pooled together

Whenever you configure a unique data source or connection factory, you are required to give it a unique Java Naming and Directory Interface (JNDI) name. This JNDI name, along with its configuration information, is used to create a *connection pool*. A separate connection pool exists for each configured data source or connection factory.

A separate instance of a given configured connection pool is created on each application server that uses that data source or connection factory. For example, if you run a three server cluster in which all of the servers use *myDataSource*, and myDataSource has a *maximum connections* setting of 10, then you can generate up to 30 connections (three servers times 10 connections). Be sure to consider this fact when determining how many connections to your backend resource you can support.

It is also important to note that when using *connection sharing*, it is only possible to share connections obtained from the same connection pool.

Avoiding a deadlock: Deadlock can occur if the application requires more than one concurrent connection per thread, and the database connection pool is not large enough for the number of threads. Suppose each of the application threads requires two concurrent database connections and the number of threads is equal to the maximum connection pool size. Deadlock can occur when both of the following are true:

- Each thread has its first database connection, and all are in use.
- Each thread is waiting for a second database connection, and none would become available since all threads are blocked.

To prevent the deadlock in this case, the **Max Connections** value for the database connection pool should be increased by at least one. Doing this would allow for at least one of the waiting threads to obtain its second database connection and to avoid a deadlock.

To avoid deadlock, code the application to use, at most, one connection per thread. If the application is coded to require C concurrent database connections per thread, the connection pool must support at least the following number of connections, where T is the maximum number of threads.

```
T * (C - 1) + 1
```

The connection pool settings are directly related to the number of connections that the database server is configured to support. If the maximum number of connections in the pool is raised, and the corresponding settings in the database are not raised, the application fails and SQL exception errors are displayed in the SYSOUT of the application servant region.

Connection life cycle

A ManagedConnection object is always in one of three states: *DoesNotExist*, *InFreePool*, or *InUse*.

Before a connection is created, it must be in the DoesNotExist state. After a connection is created, it can be in either the InUse or the InFreePool state, depending on whether it is allocated to an application.

Between these three states are *transitions*. These transitions are controlled by *guarding conditions*. A guarding condition is one in which *true* indicates when you can take the transition into another legal state. For example, you can make the transition from the InFreePool to InUse state only if:

- the application has called the data source or connection factory *.getConnection* method (*getConnection*)
- a free connection is available in the pool with matching properties (*freeConnectionAvailable*)
- and one of the two following conditions are true:
  - the getConnection request is on behalf of a resource reference that is marked unsharable
  - the getConnection request is on behalf of a resource reference that is marked shareable but no shareable connection in use has the same properties.

This transition description follows:

InFreePool > InUse:
getConnection AND
freeConnectionAvailable AND
NOT(shareableConnectionAvailable)

Here is a list of guarding conditions and descriptions.

Condition	Description
ageTimeoutExpired	Connection is older then its ageTimeout value.
close	Application calls close method on the Connection object.
fatalErrorNotification	A connection has just experienced a fatal error.
freeConnectionAvailable	A connection with matching properties is available in the free pool.
getConnection	Application calls getConnection method on DataSource or ConnectionFactory object.
markedStale	Connection is marked as stale, typically in response to a FatalErrorNotification.
noOtherReferences	There is only one connection handle to the ManagedConnection, and the Transaction Service is not holding a reference to the ManagedConnection.

Condition	Description
noTx	No transaction is in force.
poolSizeGTMin	Connection pool size is greater than the minimum pool size (minimum number of connections)
poolSizeLTMax	Pool size is less than the maximum pool size (maximum number of connections)
shareableConnectionAvailable	The getConnection request was for a shareable connection and one with matching properties is in use and available to share.
TxEnds	The transaction has ended.
unshareableConnectionRequest	The getConnection request is for an unshareable connection.
unusedTimeoutExpired	Connection is in the free pool and not in use past its unused timeout value.

Getting connections

The first set of transitions covered are those in which the application requests a connection from either a data source or a connection factory. In some of these scenarios, a new connection to the database results. In others, the connection might be retrieved from the connection pool or shared with another request for a connection.

DoesNotExist

Every connection begins its life cycle in the DoesNotExist state. When an application server starts, the connection pool does not exist. Therefore, there are no connections. The first connection is not created until an application requests its first connection. Additional connections are created as needed, according to the guarding condition.

```
getConnection AND
NOT(freeConnectionAvailable) AND
poolSizeLTMax AND
(NOT(shareableConnectionAvailable) OR
unshareableConnectionRequest)
```

This transition specifies that a Connection object is not created unless the following conditions occur:

- The application calls the *getConnection()* method on the data source or connection factory
- No connections are available in the free pool (NOT(freeConnectionAvailable))
- The pool size is less than the maximum pool size (poolSizeLTMax)
- If the request is for a sharable connection and there is no sharable connection already in use with the same sharing properties (NOT(shareableConnectionAvailable)) OR the request is for an unsharable connection (unshareableConnectionRequest)

All connections begin in the DoesNotExist state and are only created when the application requests a connection. The pool grows from 0 to the maximum number of connections as applications request new connections. The pool is **not** created with the minimum number of connections when the server starts.

If the request is for a sharable connection and a connection with the same sharing properties is already in use by the application, the connection is shared by two or

more requests for a connection. In this case, a new connection is not created. For users of the JDBC API these sharing properties are most often *userid/password* and *transaction context*; for users of the Resource Adapter Common Client Interface (CCI) they are typically *ConnectionSpec*, *Subject*, and *transaction context*.

InFreePool

The transition from the InFreePool state to the InUse state is the most common transition when the application requests a connection from the pool.

```
InFreePool>InUse:
getConnection AND
freeConnectionAvailable AND
(unshareableConnectionRequest OR
NOT(shareableConnectionAvailable))
```

This transition states that a connection is placed in use from the free pool if:
- the application has issued a getConnection() call
- a connection is available for use in the connection pool (freeConnectionAvailable),
- and one of the following is true:
  - the request is for an unsharable connection (unsharableConnectionRequest)
  - no connection with the same sharing properties is already in use in the transaction. (NOT(sharableConnectionAvailable)).

Any connection request that a connection from the free pool can fulfill does not result in a new connection to the database. Therefore, if there is never more than one connection used at a time from the pool by any number of applications, the pool never grows beyond a size of one. This number can be less than the minimum number of connections specified for the pool. One way that a pool grows to the minimum number of connections is if the application has multiple concurrent requests for connections that must result in a newly created connection.

InUse

The idea of connection sharing is seen in the transition on the InUse state.

```
InUse>InUse:
getConnection AND
ShareableConnectionAvailable
```

This transition states that if an application requests a shareable connection (getConnection) with the **same** sharing properties as a connection that is already in use (ShareableConnectionAvailable), the existing connection is shared.

The same user (*user name* and *password*, or *subject*, depending on authentication choice) can share connections but only within the same transaction and only when all of the sharing properties match. For JDBC connections, these properties include the *isolationLevel* which is configurable on the resource-reference (IBM WebSphere extension) to data source default. For a resource adapter factory connection, these properties include those specified on the *ConnectionSpec*. Because a transaction is normally associated with a single thread, you should **never** share connections across threads.

Note: It is possible to see the same connection on multiple threads at the same time, but this situation is an error state usually caused by an application programming error.

Returning connections

All of the transitions so far have covered getting a connection for application use. From this point, the transitions result in a connection closing and either returning to the free pool or being destroyed. Applications should explicitly close connections (note: the connection that the user gets back is really a connection handle) by calling **close()** on the Connection object. In most cases, this action results in the following transition:

```
InUse>InFreePool:
(close AND
noOtherReferences AND
NoTx AND
UnshareableConnection)
OR
(ShareableConnection AND
TxEnds)
```

Conditions that cause the transition from the InUse state are:

- If the application (or the container) calls **close()** (close) and there are no references (noOtherReferences) either by the application (application sharing) or by the transaction manager (NoTx - who holds a reference when the connection is enlisted in a transaction), the Connection object returns to the free pool.
- If the connection was enlisted in a transaction but the transaction manager ends the transaction (txEnds), and the connection was a shareable connection (ShareableConnection), the connection closes and returns to the pool.

When the application calls close() on a connection, it is returning the connection to the pool of free connections; it is **not** closing the connection to the data store. When the application calls close() on a currently shared connection, the connection is *not returned* to the free pool. Only after the application drops the last reference to the connection, and the transaction is over, is the connection returned to the pool. Applications using unsharable connections must take care to close connections in a timely manner. Failure to do so can starve out the connection pool making it impossible for any application running on the server to get a connection.

When the application calls close() on a connection enlisted in a transaction, the connection is not returned to the free pool. Because the transaction manager must also hold a reference to the connection object, the connection cannot return to the free pool until the transaction ends. Once a connection is enlisted in a transaction, you cannot use it in any other transaction by any other application until after the transaction is complete.

There is a case where an application calling close() can result in the connection to the data store closing and bypassing the connection return to the pool. This situation happens if one of the connections in the pool is considered stale. A connection is considered stale if you can no longer use it to contact the data store. For example, a connection is marked stale if the data store server is shut down. When a connection is marked as stale, the entire pool is cleaned out by default because it is very likely that all of the connections are stale for the same reason (or you can set your configuration to clean just the failing connection). This cleansing includes marking all of the currently InUse connections as stale so they are destroyed upon closing. The following transition states the behavior on a call to close() when the connection is marked as stale:

```
InUse>DoesNotExist:
close AND
markedStale AND
NoTx AND
noOtherReferences
```

This transition states that if the application calls close() on the connection and the connection is marked as stale during the pool cleansing step (markedStale), the connection object closes to the data store and is not returned to the pool.

Finally, you can close connections to the data store and remove them from the pool.

This transition states that there are three cases in which a connection is removed from the free pool and destroyed.

1. If a fatal error notification is received from the resource adapter (or data source). A fatal error notification (FatalErrorNotification) is received from the resource adaptor when something happens to the connection to make it unusable. All connections currently in the free pool are destroyed.
2. If the connection is in the free pool for longer than the unused timeout period (UnusedTimeoutExpired) and the pool size is greater than the minimum number of connections (poolSizeGTMin), the connection is removed from the free pool and destroyed. This mechanism enables the pool to shrink back to its minimum size when the demand for connections decreases.
3. If an age timeout is configured and a given connection is older than the timeout. This mechanism provides a way to recycle connections based on age.

Unshareable and shareable connections

The product supports both *unshareable* and *shareable* connections. An unshareable connection is not shared with other components in the application. The component using this connection has full control of this connection.

You can share a shareable connection with other components within the same transaction as long as each *getConnection* request has the same connection properties. To enable connection sharing for data sources, the following connection properties must be the same:

- Java Naming and Directory Interface (JNDI) name. While not actually a *connection* property, this requirement simply means that you can only share connections from the same dataSource in the same server.
- Resource authentication
- In relational databases:
  - Isolation level (corresponds to Access Intent in the CMP bean)
  - Readonly
  - Catalog
  - TypeMap

To enable connection sharing for resource adapters within the same transaction, the following connection properties must be the same:

- JNDI name. While not actually a *connection* property, this requirement simply means that you can only share connections from the same resource adapter in the same server.
- Resource authentication

In addition, the *ConnectionSpec* used to get the connection must also be the same. For more information on sharing a connection with a CMP bean, see Sharing a connection with a CMP bean.

Access to a resource marked as **Unshareable** means that there is a one-to-one relationship between the connection handle a component is using and the physical connection the handle is associated with. This access implies that every call to *getConnection* returns a connection handle solely for the requesting user. Typically,

you must choose unshareable if you might do things to the connection that could result in unexpected behavior occurring to another application that is sharing the connection (for example, changing the isolation level).

Marking a resource as **Shareable** allows for greater scalability. Instead of creating new physical connections on every *getConnection* invocation, the physical connection (that is, managed connection) is shared through multiple connection handles, as long as each getConnection request has the same connection properties. But, sharing a connection means that each user must not do anything to the connection that could change its behavior and disrupt a sharing partner (for example, changing the isolation level). The user also cannot code an application *expecting* sharing to take place because it is up to the run time to decide whether or not to share a particular connection.

For WebSphere Application Server, all sharing of connections is relative to the current Unit of Work (UOW) boundary. Anyone within a specific transaction, when getting a connection from a specific connection pool, gets a handle to the same physical connection (if the sharing properties are the same).

Factors that determine sharing

The listing here is not an exhaustive one. The product might or might not share connections under different circumstances.

- Only connections acquired with the same resource reference (resource-ref), which specifies the res-sharing-scope as *Shareable*, are candidates for sharing. The resource-ref properties of res-sharing-scope and res-auth and the IBM extension isolationLevel help determine if it is possible to share a connection. IBM extension isolationLevel is stored in IBM deployment descriptor extension file; for example: ibm-ejb-jar-ext.xmi.
- You can only share connections that are requested with the same properties.
- Connection Sharing only occurs between different component instances if they are within a transaction (container- or user-initiated transaction).
- Connection Sharing only occurs within a sharing boundary. Current sharing boundaries include *Transactions* and *LocalTransactionContainment* (LTC) boundaries.
- Connection Sharing rules within an LTC Scope:
  - For shareable connections, only *Connection Reuse* is allowed within a single component instance. Connection reuse occurs when the following actions are taken with a connection: get, use, commit/rollback, close; get, use, commit/rollback, close. Note that if you use the LTC resolution-control of *ContainerAtBoundary* then no start/commit is needed because that action is handled by the container.

    The connection returned on the second *get* is the same connection as that returned on the first *get* (if the same properties are used). Because the connection use is serial, only one connection handle to the underlying physical connection is used at a time, so true *connection sharing* does not take place. The term *"reuse"* is more accurate.
- Shareable connections between transactions (either container-managed transactions (CMT), bean-managed transactions (BMT), or LTC transactions) follow these caching rules:
  - In general, setting properties on shareable connections is not allowed because a user of one connection handle might not anticipate a change made by another connection handle. This limitation is part of the J2EE 1.3 standard.
  - General users of resource adapters can set the connection properties on the connection factory *getConnection call* by passing them in a *ConnectionSpec*.

    However, the properties set on the connection during one transaction are not guaranteed to be the same when used in the next transaction. Because it is

not valid to share connections outside of a sharing scope, connection handles are moved off of the physical connection with which they are currently associated when a transaction ends. That physical connection is returned to the free connection pool. Connections are cleaned before going in the free pool. The next time the handle is used, it is automatically associated with an appropriate connection. The appropriateness is based on the security login information, connection properties, and (for the JDBC API) the *isolation level* specified in the extended resource reference, passed in on the original request that returned the current handle. Any properties set on the connection after it was retrieved are lost.

  – For JDBC users, WebSphere Application Server provides an extension to enable you to pass the connection properties through the ConnectionSpec.

    Use caution when setting properties and sharing connections in a local transaction scope. Ensure that other components with which the connection is shared are expecting the behavior resulting from your settings.

- You cannot set the *IsolationLevel* when using a shareable connection for the JDBC API using a relational resource adapter in a global transaction. The product provides an extension to the resource reference to enable you to specify the isolation level. If your application requires the use of multiple isolation levels, create multiple resource references and map them to the same data source or connection factory.

Sharing a connection with a CMP bean

WebSphere Application Server allows you to share a physical connection between a CMP bean, a BMP bean, and a JDBC application to reduce the resource allocation or deadlock scenarios. There are several ways to ensure that all these Entity beans and the JDBC applications are sharing the same physical connection.

- **Sharing a connection between CMP beans or methods**

  When all CMP bean methods use the same access intent, they all share the same physical connection. A different access intent policy triggers the allocation of a different physical connection. For example, a CMP bean has two methods; method 1 is associated with wsPessimisticUpdate intent, whereas method 2 has wsOptimisticUpdate access intent. Method 1 and method 2 cannot share the same physical connection within a transaction. In other words, an XA DataSource is required to run in a global transaction.

  You can experience some deadlocks from a database if both methods try to access the same table. Therefore, sharing a connection is determined by the access intents that are defined in the CMP methods.

- **Sharing a connection between CMP and BMP beans**

  There are two options to ensure that both CMP and BMP beans share the same physical connection:

  – Define the same access intent on both CMP and BMP bean methods. Because both use the same access intent, they share the same physical connection. The advantage to using this option is that the backend is transparent to a BMP bean; however, this BMP is not portable because it uses the WebSphere extended API to handle the isolation level. For more information, refer to the code example in Example: Accessing data using IBM extended APIs to share connections between container-managed and bean-managed persistence beans.

  – Determine the isolation level that the access intent uses on a CMP bean method, then use the corresponding isolation level that is specified on the resource reference to look up a data source and a connection. This option is more of a manual process, and the isolation level might be different from database to database. For more information refer to the isolation level and

access intent mapping table: Access intent isolation levels and update locks and the Isolation level and resource reference section.

- **Sharing a connection between CMP and a JDBC application that is used by a servlet or a Session Bean**

  Determine the isolation level that the access intent uses on a CMP bean method, then use the corresponding isolation level specified on the resource reference to look up a data source and a connection. For more information refer to Access intent isolation levels and update locks and Isolation level and resource reference.

Connection handles

A connection handle is a representation of a physical connection.

To use a backend resource (such as a relational database) in the WebSphere Application Server you must get a connection to that resource. When you call the *getConnection()* method, you get a *connection handle* returned. The handle is not the physical connection. The physical connection is managed by the connection manager.

There are two significant configurations or usage patterns that affect how connection handles are used and how they behave. The first is the *res-sharing-scope*, which is defined by the resource-reference used to look up the DataSource or Connection Factory. This property tells the connection manager whether or not you can share this connection.

The second factor that affects connection handle behavior is the *usage pattern*. There are essentially two usage patterns. The first is called the *get/use/close* pattern. It is used within a single method and without calling another method that might get a connection from the same data source or connection factory. An application using this pattern does the following:

1. gets a connection
2. does its work
3. commits (if appropriate)
4. closes the connection.

The second usage pattern is called the *cached handle* pattern. This is where an application:

1. gets a connection
2. begins a global transaction
3. does work on the connection
4. commits a global transaction
5. does work on the connection again

A cached handle is a connection handle that is held across transaction and method boundaries by an application. Keep in mind the following considerations for using cached handles:

- Cached handle support requires some additional connection handle management across these boundaries, which can impact performance. For example, in a JDBC application, *Statements*, *PreparedStatements*, and *ResultSets* are closed implicitly after a transaction ends, but the connection remains valid.
- You are encouraged **not** to cache the connection across the transaction boundary for shareable connections; the get/use/close pattern is preferred.
- **5.1 +** Caching of connection handles across servlet methods is limited to Java Database Connectivity (JDBC) and Java Message Service (JMS) resources. Other

non-relational resources, such as Customer Information Control System (CICS) or IMS, currently cannot have their connection handles cached in a servlet; you need to get, use, and close the connection handle within each method invocation. (This limitation only applies to single-threaded servlets because multithreaded servlets do not allow caching of connection handles.)

The following code segment shows the cached connection pattern.

```
Connection conn = ds.getConnection();
ut.begin();
conn.prepareStatement("....."); --> Connection runs in global transaction mode
...
ut.commit();
conn.prepareStatement(".....");   ---> Connection still valid but runs in autoCommit(True);
...
```

Unshareable connections

Some characteristics of connection handles retrieved with a *res-sharing-scope* of **unshareable** are described in the following sections.

The possible benefits of unshared connections

- Your application always maintains a direct link with a physical connection (managed connection).
- The connection always has a one-to-one relationship between the connection handle and the managed connection.
- In most cases, the connection does not close until the application closes it.
- You can use a cached unshared connection handle across multiple transactions.
- The connection can have a performance advantage in some cached handle situations. Because unshared connections do not have the overhead of moving connection handles off managed connections at the end of the transaction, there is less overhead in using a cached unshared connection.

The possible drawbacks of unshared connections

- Inefficient use of your connection resources. For example, if within a single transaction you get more than one connection (with the same properties) using the same data source or connection factory (same resource-ref) then you use multiple physical connections when you use unshareable connections.
- Wasted connections. It is important not to keep the connection handle open (that is, you have not called the *close()* method) any longer then it is needed. As long as you keep an unshareable connection open you tie up the physical connection, even if you currently are not using it.
- Deadlock considerations. Depending on how your components interact with the database within a transaction, using unshared connections can lead to deadlocks in the database. For example, within a transaction, component A gets a connection to data source X and updates table 1, and then calls component B. Component B gets another connection to data source X, and updates/reads table 1 (or even worse the same row as component A). In some circumstances, depending on the particular database, its locking scheme, and the transaction isolation level, a deadlock can occur.

  In the same scenario, but with a *shared* connection, a deadlock does not occur because all the work was done on the same connection. It is worth noting that when writing code which uses shared connections, it is important that the code be written in such a way that it expects other work to be done on the same connection, possibly within the same transaction. If you decide to use an unshareable connection, you must set the *maximum connections* property on the connection factory or data source correctly. An exception occurs if you try to exceed the maximum connections value.

Shareable connections

Some characteristics of connection handles retrieved with a *res-sharing-scope* of **shareable** are described in the following sections.

The possible benefits of shared connections

- They can share a managed connection with one or more connection handles within a sharing, boundary depending upon how the handle is retrieved and which connection properties are used.
- They can more efficiently use resources. Shareable connections are not valid outside of their sharing boundary. For this reason, at the end of a sharing boundary (such as transaction) the connection handle is no longer associated with the managed connection it was using within the sharing boundary (this applies only when using the cached handle pattern). The managed connection is returned to the free connection pool for reuse. Connection resources are not held longer than the end of the current sharing scope.

  If the cached handle pattern is used, then the next time the handle is used within a new sharing scope, the application server run time assures that the handle is reassociated with a managed connection appropriate for the current sharing scope and with the same properties with which the handle was originally retrieved. Remember that it is not appropriate to change properties on a shareable connection. If properties are changed, other components that share the same connection might experience unexpected behavior. Futhermore, when using cached handles, the value of the changed property might not be remembered across sharing scopes.

The possible drawbacks of shared connections

- Sharing within a single component (such as an enterprise bean and its related Java objects) is not always supported. The current specification allows resource adapters the choice of only allowing one active connection handle at a time.

  If a resource adapter chooses to implement this option then the following scenario results in an *invalid handle exception*: A component using shareable connections gets a connection and uses it. Without closing the connection, the component calls a utility class (Java object) which gets a connection (handle) to the same managed connection and uses it. Because the resource adapter only supports one active handle, the first connection handle is no longer valid. If the utility object returns without closing its handle, the first handle remains invalid and use of it causes an exception.

  Note: This exception occurs only when calling a utility object (a Java object).

  Not all resource adapters have this limitation, it depends on their implementation. The WebSphere Relational Resource Adapter (RRA) does not have this limitation. Any DataSource used through the RRA does not have this limitation. If you encounter a resource adapter with this limitation you can work around it by serializing your access to the managed connection. If you always close your connection handle before getting another, or close your handle before calling code which gets another handle, and you always close your handle before you return from the method, you can allow two pieces of code to share the same managed connection. You just cannot use the connection for both events at the same time.
- Trying to change the *isolation level* on a shareable JDBC based connection in a global transaction (those supported by the RRA) causes an exception. The correct way to get connections with different transaction isolation levels is by configuring the IBM extended resource-reference.

- Closing connection handles for shareable connections by an application is NOT supported and causes errors. However, you can avoid this limitation by using the Relational Resource Adapter.

Connections and transactions

All connection usage occurs within the scope of either a global transaction or a local transaction containment (LTC).

Connection behavior depends on your current operating scope. This article discusses some of the common characteristics you see when using connections in one of the transaction scopes of the product.

You can only share connections within a global transaction scope (assuming other sharing rules are met). However, you can *serially reuse* connections within an LTC scope. A a get/use/close connection pattern followed by another get/use/close (to the same data source or connection factory) enables you to reuse the same connection. See Unshareable and shareable connections for more details.

JDBC AutoCommit behavior

All JDBC connections, when first obtained through a *getConnection* call, have AutoCommit = TRUE by default.

- If you operate within an LTC and have its resolution-control set to *Application*, then AutoCommit remains *TRUE* unless changed by the application.
- If you operate within an LTC and have its resolution-control set to *ContainerAtBoundary*, then the application should **not** touch the AutoCommit setting. The WebSphere Application Server run time sets the AutoCommit value to *FALSE* before work begins, then commits or rolls back the work as appropriate at the end of the LTC scope.
- If you use a connection within a global transaction, then regardless of the user changing the AutoCommit setting, upon first use of the connection to do work the database ignores the AutoCommit setting so that the transaction service that controls the commit and rollback processing can manage the transaction . After the transaction completes, the AutoCommit value returns to the value it had before the first use of the connection. So even if the AutoCommit value is set to *TRUE* before the connection is used in a global transaction, you need not set the value to *FALSE* since the value is ignored by the database. In this example, after the transaction completes, the AutoCommit value of the connection returns to *TRUE*.
- If you use multiple distinct connections within a global transaction, all work is guaranteed to commit or roll back together. This is not the case for a local transaction containment (LTC scope). Within an LTC, work done on one connection commits or rolls back independently from work done on any other connection within the LTC.

One-phase commit and two-phase commit resources

One-phase commit resources are such that work being done on a one phase connection cannot mix with other connections and ensure that the work done on all of the connections completes or fails atomically . The product does not allow more than one one-phase commit connection in a global transaction. Futhermore, it does not allow a one-phase commit connection in a global transaction with one or more two-phase commit connections. You can coordinate only multiple two-phase commit connections within a global transaction.

Note that any time you do multiple *getConnection* calls using a resource reference that specifies *res-sharing-scope=Unshareable* , then you get multiple physical connections. This situation also occurs when *res-sharing-scope=Shareable* but the sharing rules are broken. In either case, if you run in a global transaction, ensure the resources involved are enabled for two-phase commit (also sometimes referred to as *JTA Enabled*). Failure to do so results in an XAException that logs the following message: WTRN0063E: An illegal attempt to enlist a one phase capable resource with existing two phase capable resources has occurred.

Developing data access applications

Note: WebSphere Application Server does not support JDBC 3.0.

You can access data in various ways:

- using standard or extended APIs
- using container-managed persistence beans
- using bean-managed persistence beans, session beans, or Web components.

1. Decide how to implement data access.

   The Enterprise JavaBeans (EJB) programming model provides several distinct server-side component types: entity, session, and message-driven beans, and servlets. Of these types, entity beans are typically used to model business components in an application. Entity beans have both *state* and *behavior*.

   The state of entity beans is persistent and is stored in a database. As changes are made to an entity bean, its state is kept in synchronization with the database record representing the bean. There are two types of entity beans provided by the EJB model and these two types differ in the mechanism used to provide persistence. These two types of entity beans are *container-managed persistence* (CMP) beans and *bean-managed persistence* (BMP) beans.

   With BMP beans, the developer manually produces code to manage the persistent state of the bean.

   With CMP beans, the EJB container manages the beans persistent state. Persistent state management is a complex and difficult task and using CMP beans allows the developer to concentrate on business logic by delegating persistence behavior to the container. Typical examples of CMP beans are *Customer*, *Account*, and so on. Because CMP beans are objects, their data (state) is accessed using field accessors. For example, a *Customer* entity bean is likely to have fields such as *name* and *phoneNumber*. These pieces of data are accessed using the accessor methods *getName()/setName()* and *getPhoneNumber()/setPhoneNumber()*. As a developer, you are not concerned with how this data is eventually stored and retrieved from the backend database and can assume that the integrity of the data is maintained by the container.

   Starting with WebSphere Application Server Version 5.0.1, you can use Structured Query Language in Java (SQLJ) support with both BMP and CMP beans when you are using the DB2 Universal JDBC driver provider with DB2 as your backend database. You can also use SQLJ support with BMP beans when you are using the DB2 for z/OS Local JDBC provider (RRS) with DB2 for z/OS as your backend database. DB2 for z/OS users who wish to use SQLJ support with CMP beans must use the DB2 Universal JDBC driver provider.

2. Create a JDBC provider and data source (Creating and configuring a JDBC provider and data source), or create a J2EE Connector Architecture (JCA) connection factory (Configuring Java 2 Connector connection factories in the administrative console). An application component uses a connection factory to access a connection instance, which the component then uses to connect to the underlying enterprise information system (EIS). A data source is associated

with a JDBC provider that supplies the specific JDBC driver implementation class. The data source represents the JCA connection factory for the relational resource adapter.

3. Look up a data source or connection factory using a resource reference (Looking up data sources with resource references for relational access). Using a resource reference to access your data source or connection factory is required when running in WebSphere Application Server.
4. Get a connection to a data source (Connection management architecture). The connection management architecture for both relational and procedural access to enterprise information systems (EIS) is based on the J2EE Connector Architecture (JCA) specification. The Connection Manager (CM), which pools and manages connections within an application server, is capable of managing connections obtained through both resource adapters (RAs) defined by the JCA specification, and DataSources defined by the JDBC 2.0 Extensions Specification.
5. Use thread identity to assign an owner to the connection.

Data access application programming interface support

Applications can access the backend data through the standard J2EE 1.3 defined application programming interfaces (APIs).

The standard APIs do not always provide a complete solution for an application that runs in an application server. For example, the JDBC programming model sometimes does not completely work with the J2EE Connector Architecture (JCA) Specification (even though the JCA architecture has explicitly specified that it integrates with the JDBC programming model). These gaps cause some incompatibility between the JDBC and JCA programming models.

When getting and using shareable connections in a global transaction, it is not valid to change a property on the connection after you obtain it. Changes can unknowingly affect other users who share the same connection.

The J2EE Connector Architecture (JCA) Specification supports telling the resource adapter the specific properties settings at the time you request the connection (using the *getConnection* method) by passing in a *ConnectionSpec*. The ConnectionSpec contains the necessary connection properties used to get a connection. After you obtain a connection from this environment, your application does not need to alter the properties.

The JDBC programming model does not have the same interface to specify the connection properties. Instead, it gets the connection first, then sets the properties on the connection. In the case of a shareable connection, changing the connection properties impacts all the connections shared with the same physical connection.

WebSphere Application Server provides the following extensions to fill in the gaps between these two specifications.

- **WSDataSource interface** - this interface extends *javax.sql.DataSource*, and enables a component or an application to specify the connection properties through the WebSphere Application Server *JDBCConnectionSpec* to get a specific connection.
  - *getConnection(JDBCConnectionSpec)* - this method returns a specific connection which has the JCA compliant connection behavior.
  - For more information see the Javadoc **wsdatasource** in the Javadoc index.
- **JDBCConnectionSpec interface** - this interface extends the *com.ibm.websphere.rsadapter.WSConnectionSpec*, which extends *javax.resources.cci.ConnectionSpec*. The standard *ConnectionSpec* interface provides

only the interface marker without any get and set methods. The *WSConnectionSpec* and *JDBCConnectionSpec* define a set of get and set methods used by the WebSphere Application Server run time. This interface enables the application to specify all the essential connection properties in order to get an appropriate connection. You can create this class from the WebSphere WSRRAFactory. For more information see the Javadoc **JDBCConnection** in the Javadoc index.

- **WSRRAFactory** - this is the Relational Resource Adapter Factory which allows the user to create a *JDBCConnectionSpec* or other resource adapter related object. For more information see the Javadoc **wsrrafactory** in the Javadoc index

Example: Accessing data using IBM extended APIs for connections

If your application runs with a shareable connection that might be shared with other container-managed persistence (CMP) beans within a transaction, it is recommended that you use the WebSphere Application Server extended APIs to get the connection. When you use these APIs, you cannot port your application to other application servers.

You can access an extended API in your JDBC application. Instead of using the DataSource interface, you use the WSDataSource interface. The following code segment illustrates how to get the connection.

```
import com.ibm.websphere.rsadapter.*;

...

// Create a JDBCConnectionSpec and set connection properties. If this connection is shared with
the CMP bean, make sure that the isolation level is the same as the isolation level
that is mapped by the Access Intent defined on the CMP bean.

JDBCConnectionSpec connSpec = WSRRAFactory.createJDBCConnectionSpec();

connSpec.setTransactionIsolation(CONNECTION.TRANSACTION_REPEATABLE_READ);

connSpec.setCatalog("DEPT407");


//Use WSDataSource to get the connection

Connection conn = ((WSDataSource)datasource).getConnection(connSpec);
```

Example: Accessing data using IBM extended APIs to share connections between container-managed and bean-managed persistence beans

If your application runs with a shareable connection that might be shared with other container-managed persistence (CMP) beans within a transaction, it is recommended that you use the WebSphere Application Server extended APIs to get the connection. When you use these APIs, you cannot port your application to other application servers.

You can access an extended API in your JDBC application. Instead of using the DataSource interface, you use the WSDataSource interface.

To ensure that both CMP and bean-managed persistence (BMP) beans are sharing the same physical connection, you can define the same Access Intent profile on both the CMP and BMP beans. Inside your BMP method, you can get the right isolation level from the relational resource adapter helper class.

```
package fvt.example;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import javax.ejb.CreateException;
import javax.ejb.DuplicateKeyException;
import javax.ejb.EJBException;
import javax.ejb.ObjectNotFoundException;
import javax.sql.DataSource;

// following imports are used by the IBM extended API
import com.ibm.websphere.appprofile.accessintent.AccessIntent;
import com.ibm.websphere.appprofile.accessintent.AccessIntentService;
import com.ibm.websphere.rsadapter.JDBCConnectionSpec;
import com.ibm.websphere.rsadapter.WSCallHelper;
import com.ibm.websphere.rsadapter.WSDataSource;
import com.ibm.websphere.rsadapter.WSRRAFactory;

/**
 * Bean implementation class for Enterprise Bean: Simple
 */

public class SimpleBean implements javax.ejb.EntityBean {
 private javax.ejb.EntityContext myEntityCtx;

 // Initial context used for lookup.

 private javax.naming.InitialContext ic = null;

 // define a JDBCConnectionSpec as instance variable

 private JDBCConnectionSpec connSpec;

 // define an AccessIntentService which is used to get
 // an AccessIntent object.

 private AccessIntentService aiService;

 // AccessIntent object used to get Isolation level

 private AccessIntent intent = null;

 // Persitence table name

 private String tableName = "cmtest";

 // DataSource JNDI name

 private String dsName = "java:comp/env/jdbc/SimpleDS";

 // DataSource

 private DataSource ds = null;

 // bean instance variables.

 private int id;
 private String name;

 /**
  * In setEntityContext method, you need to get the AccessIntentService
  * object in order for the subsequent methods to get the AccessIntent
  * object.
  * Other ejb methods will call the private getConnection() to get the
```

```
 * connection which has all specific connection properties
 */

public void setEntityContext(javax.ejb.EntityContext ctx) {
 myEntityCtx = ctx;

 try {
  aiService =
   (AccessIntentService) getInitialContext().lookup(
    "java:comp/websphere/AppProfile/AccessIntentService");
  ds = (DataSource) getInitialContext().lookup(dsName);
 }
 catch (javax.naming.NamingException ne) {
  throw new javax.ejb.EJBException(
   "Naming exception: " + ne.getMessage());
 }
}

 /**
 * ejbCreate
 */

public fvt.example.SimpleKey ejbCreate(int newID)
 throws javax.ejb.CreateException, javax.ejb.EJBException {
 Connection conn = null;
 PreparedStatement ps = null;

 // Insert SQL String

 String sql = "INSERT INTO " + tableName + " (id, name) VALUES (?, ?)";

 id = newID;
 name = "";

 try {
                   // call the common method to get the specific connection

  conn = getConnection();
 }
 catch (java.sql.SQLException sqle) {
  throw new EJBException("SQLException caught: " + sqle.getMessage());
 }
 catch (javax.resource.ResourceException re) {
  throw new EJBException(
   "ResourceException caught: " + re.getMessage());
 }

 try {
  ps = conn.prepareStatement(sql);
  ps.setInt(1, id);
  ps.setString(2, name);

  if (ps.executeUpdate() != 1) {
   throw new CreateException("Failed to add a row to the DB");
  }
 }
 catch (DuplicateKeyException dke) {
  throw new javax.ejb.DuplicateKeyException(
   id + "has already existed");
 }
 catch (SQLException sqle) {
  throw new javax.ejb.CreateException(sqle.getMessage());
 }
 catch (CreateException ce) {
  throw ce;
 }
 finally {
```

```
  if (ps != null) {
   try {
    ps.close();
   }
   catch (Exception e) {
   }
  }
 }
 return new SimpleKey(id);
}

 /**
 * ejbLoad
 */

public void ejbLoad() throws javax.ejb.EJBException {

 Connection conn = null;
 PreparedStatement ps = null;
 ResultSet rs = null;

 String loadSQL = null;

 try {
                   // call the common method to get the specific connection

  conn = getConnection();
 }
 catch (java.sql.SQLException sqle) {
  throw new EJBException("SQLException caught: " + sqle.getMessage());
 }
 catch (javax.resource.ResourceException re) {
  throw new EJBException(
   "ResourceException caught: " + re.getMessage());
 }

 // You need to determine which select statement to be used based on the
 // AccessIntent type:
 // If READ, then uses a normal SELECT statement. Otherwise uses a
 // SELECT...FORUPDATE statement
 // If your backend is SQLServer, then you can use different syntax for
 // the FOR UPDATE clause.

 if (intent.getAccessType() == AccessIntent.ACCESS_TYPE_READ) {
  loadSQL = "SELECT * FROM " + tableName + " WHERE id = ?";
 }
 else {
  loadSQL = "SELECT * FROM " + tableName + " WHERE id = ? FOR UPDATE";
 }

 SimpleKey key = (SimpleKey) getEntityContext().getPrimaryKey();

 try {
  ps = conn.prepareStatement(loadSQL);
  ps.setInt(1, key.id);
  rs = ps.executeQuery();
  if (rs.next()) {
   id = rs.getInt(1);
   name = rs.getString(2);
  }
  else {
   throw new EJBException("Cannot load id = " + key.id);
  }
 }
 catch (SQLException sqle) {
  throw new EJBException(sqle.getMessage());
 }
```

```
 finally {
  try {
   if (rs != null)
    rs.close();
  }
  catch (Exception e) {
  }
  try {
   if (ps != null)
    ps.close();
  }
  catch (Exception e) {
  }
  try {
   if (conn != null)
    conn.close();
  }
  catch (Exception e) {
  }
 }
}

       /**
        * This method will use the AccessIntentService to get the access intent;
        * then gets the isolation level from the DataStoreHelper
        * and sets it in the connection spec; then uses this connection
        * spec to get a connection which has the specific connection
        * properties.
        **/

private Connection getConnection()
 throws java.sql.SQLException, javax.resource.ResourceException, EJBException {

 // get current access intent object using EJB context
 intent = aiService.getAccessIntent(myEntityCtx);

 // Assume this bean only supports the pessimistic concurrency
 if (intent.getConcurrencyControl()
  != AccessIntent.CONCURRENCY_CONTROL_PESSIMISTIC) {
  throw new EJBException("Bean supports only pessimistic concurrency");
 }

 // determine correct isolation level for currently configured database
 // using DataStoreHelper
 int isoLevel =
  WSCallHelper.getDataStoreHelper(ds).getIsolationLevel(intent);
  connSpec = WSRRAFactory.createJDBCConnectionSpec();
 connSpec.setTransactionIsolation(isoLevel);

 // Get connection using connection spec
 Connection conn = ((WSDataSource) ds).getConnection(connSpec);
 return conn;
}
```

Container-managed persistence features

The container-managed persistence (CMP) features include those defined by the EJB 2.0 Specification, as well as capabilities that are beyond the specification.

EJB 2.0 specified capabilities

Container-Managed Relationships (CMR) is one of the most significant new features added by the EJB 2.0 Specification. Like *Inheritance*, relationships are a key component of object-oriented software development and non-trivial object models

can form complex networks with these relationships. The EJB 2.0 Specification adds relationships to the EJB programming model and requires that the container be responsible for their maintenance.

The container automatically manages the state of CMP entity beans. This management includes synchronizing the state of the bean with the underlying database when necessary and also managing any relationships (CMRs) with other entity beans. The bean developer is relieved of writing any database specific code and, instead, can focus on business logic.

Local interfaces are another feature introduced in the EJB 2.0 Specification. Local component interfaces allow co-located beans to interact without the overhead associated with remote access.

Value-add features

Several capabilities are provided to enhance the function of CMP entity beans that go beyond those capabilities defined by the specification. These include:

Entity bean inheritance

Inheritance is a key aspect of object-oriented software development and is a capability currently missing from the EJB 2.0 Specification.

The use of inheritance enables a developer to define fields, relationships, and business logic in a superclass entity bean that are inherited by all subclasses. See the section *EJB inheritance* of the WebSphere Studio Application Developer (WSAD) documentation for details on using inheritance with WebSphere Application Server and entity beans.

Access Intent Policies

Access intent policies provide J2EE application developers the mechanism by which they can indicate the intent of an application's interaction with the essential state for entity beans in order that the persistence mechanisms can make appropriate optimizations. For example, if it is known that an entity is not updated during the course of a transaction, then the persistence management is able to ease up on the concurrency control and still maintain data integrity by disallowing update operations on that bean for the duration of the transaction.

Caching data across transactions

Data caching across transactions is a configurable option set by the bean deployer that can greatly improve performance. Essentially, this is for data that changes infrequently. The option is known as *LifetimeInCache*. The data for an entity configured for lifetime in cache is stored in a cache until its specified lifetime expires. Requests on the entity during that configured lifetime use the cached data, and do not result in the execution of queries against the underlying data store. Lifetime can be expressed as time elapsed since the data was retrieved from the data store or until a specific time of day or week. See the *LifetimeInCache* help sections of the Assembly Toolkit (ATK) for more details.

Container-managed persistence restrictions and exceptions

The container-managed persistence (CMP) features have certain restrictions when used in specific ways.

Enterprise bean deployment and Sybase IMAGE type restriction

When deploying enterprise beans with container managed persistence (CMP) types that are non-primitive and do not have a natural Java Database Connectivity (JDBC) mapping, the deployment tool maps the CMP type to a binary type in the

database, where it is stored as a serialized instance. For Sybase, the tool uses the JDBC type *LONG VARBINARY*. The Sybase driver maps *LONG VARBINARY* to the native type *IMAGE*.

Although the type *VARBINARY* has fewer restrictions than *IMAGE* in Sybase, you cannot use it because it is limited to a size of 255 bytes, which is too small for typical serialized Java objects.

The specific restrictions on the IMAGE type are:

- You cannot use the *IMAGE* type in the *WHERE* clause of an SQL query. You can encounter this restriction whenever an enterprise bean contains an EJB-QL query that has a CMP type in the *WHERE* clause, which maps to the *IMAGE* type in the Sybase relational database (RDB).
- You cannot use *IMAGE* type in select queries marked *DISTINCT*. This situation arises in these user scenarios:
  - When the *DISTINCT* key word is specified in an EJB-QL select query having a Java type mapping to IMAGE.
  - When Enterprise beans have finder and ejbSelect() methods returning java.util. Set and have CMP types mapping to *IMAGE*.

To work around this restriction, edit the EJB mappings in the WebSphere Studio Application Developer toolset and do either of the following:

- If you are **sure** that the serialized instance of the CMP type is **never** larger than 255 bytes, you can change the CMP type mapping from *IMAGE* or *LONG VARBINARY* to *VARBINARY*.
- Map the CMP type to multiple RDB fields through a composer. For example, if the CMP type is a Java object X with an int field and a string field, then map X to two RDB fields *INTEGER* and *VARCHAR*, using a composer. Refer to the WebSphere Studio Application Developer documentation for more information about using composers.

A ClassCastException exception is thrown when running container managed persistence 1.1 beans

If you created your Enterprise JavaBeans (EJB) application using WebSphere Studio Application Developer or WebSphere Studio Application Developer Integration Edition, Version 4.0.x , and the application contains container managed persistence (CMP) 1.1 beans with associations (relationships), you might receive a *java.lang.ClassCastException* exception when you run your application on WebSphere Application Server Enterprise, Version 5.0.2 and WebSphere Business Integration Server Foundation, Version 5.1 or later

The cast operation generated by WebSphere Studio Application Developer or WebSphere Studio Application Developer Integration Edition, Version 4.0.x does not use the *javax.rmi.PortableRemoteObject.narrow(...)* object to convert the remote object to the remote interface of CMP beans in the *XToYLink.java* (or *YToXLink.java*) class where X and Y are CMP 1.1 beans.

Recommended response

1. Locate the following methods in all link classes, for example, XToYLink.java and YToXLink.javawhere X and Y are CMP 1.1 beans:

```
public void secondaryAddElementCounterLinkOf(javax.ejb.EJBObject anEJB)
public void secondaryRemoveElementCounterLinkOf(javax.ejb.EJBObject anEJB)
public void secondarySetCounterLinkOf(javax.ejb.EJBObject anEJB)
```

2. Add the javax.rmi.PortableRemoteObject.narrow(...) object to convert the remote object to the remote interface of CMP beans.

For example, change the following original method:

```
public void secondaryAddElementCounterLinkOf(javax.ejb.EJBObject anEJB) throws
  java.rmi.RemoteException {
        if (anEJB != null)
               ((X) anEJB).secondaryAddY((Y) getEntityContext().getEJBObject());
```

to:

```
public void secondaryAddElementCounterLinkOf(javax.ejb.EJBObject anEJB) throws
  java.rmi.RemoteException {
```

Note: The previous code should be entered on one line; it is split here for formatting purposes.

```
   if (anEJB != null)
             ((X) anEJB).secondaryAddY((Y)
javax.rmi.PortableRemoteObject.narrow(getEntityContext().getEJBObject(), Y.class));
```

Looking up data sources with resource references for relational access

Using a resource reference to access your data source or connection factory is required when running in WebSphere Application Server. Some of the reasons follow:

- If a data source is looked up directly, the connection gets all default properties for the missing resource reference. For example, the *sharing-scope* is a shareable connection resulting in the possibility that the physical connection is the same each time the connection is requested from the data source. This situation can cause a multitude of problems if you expect unshareable connections.
- It relieves the programmer from having to know the name of the actual data source at the target application server.
- You can set the default isolation level for the data source through resource references. With no resource reference you get the default for the JDBC driver you use.

Use a resource reference (resource-ref) for looking up a data source through the standard Java Naming and Directory Interface (JNDI) naming interface. The JNDI name defined in the resource-ref is a logical name of the data source. Have your application use this JNDI name to look up a data source instead of using the JNDI name that is defined on the data source.

Later, you can substitute the real name, either by using the Assembly Toolkit (ATK) or during installation of the application EAR file onto the server.

For example, assume that you use a DataSource *jdbc/Section* as illustrated in the code below.

```
javax.sql.DataSource specificDataSource  =
   (javax.sql.DataSource) (new InitialContext()).lookup("java:comp/env/jdbc/Section");
```

In the ATK, specify the name (*jdbc/Section*) as the resource reference. If you know the name of the DataSource, you specify it in the resource references Bindings page.

Isolation level and resource reference

In a J2EE 1.2 module, you can specify the isolation level at an enterprise bean method level, bean level, or module level. This capability has been removed from the J2EE 1.3 module. WebSphere Application Server Version 5.0 is compliant with the J2EE 1.3 specification; therefore you cannot specify isolation level on the EJB method level or bean level. Also, if a JDBC application, a bean-managed persistence (BMP) bean, or a servlet runs in a global transaction, and you are using shareable connections, you cannot set the isolation level on a connection.

When a container-managed persistence (CMP) bean uses a new data source to access a backend database, the isolation level is determined by the WebSphere Application Server run time based on the type of access intent that this method or the bean has chosen. All other connection users can also use the access intent and application profile support to manage their concurrency control.

For all JDBC connections (excluding those used by CMP beans), you can specify an isolation level default on the resource reference. For shareable connections that run in global transactions, this default is the only way to set the *isolationLevel* for connections. Trying to directly set the isolation level through the *setTransactionIsolation()* method on a shareable connection that runs in a global transaction is not allowed. To use a different isolation level on connections, you must provide a different resource reference. Set these defaults through the Assembly Toolkit (ATK).

Each resource reference associates with one isolation level. When your application uses this resource reference Java Naming and Directory Interface (JNDI) name to look up a data source, every connection returned from this data source using this resource reference has the same isolation level.

Components needing to use shareable connections with multiple isolation levels can create multiple resource references, giving them different JNDI names, and have their code look up the appropriate data source for the isolation level they need. In this way, you use separate connections with the different isolation levels enabled on them.

It is possible to map these multiple resource references to the same configured data source. The connections still come from the same underlying pool, however, the connection manager does not allow sharing of connections requested by resource references with different isolation levels.

For example, a data source is bound to two resource references: *jdbc/RRResRef* and *jdbc/RCResRef*. RRResRef has the *RepeatableRead* isolation level defined. RCResRef has the *ReadCommitted* isolation level defined. If your application wants to update the tables or a BMP bean updates some attributes, it can use the *jdbc/RRResRef* JNDI name to look up the data source instance. All connections returned from the data source instance have a RepeatableRead isolation level. If the application wants to perform a query for read only, then it is better to use the *jdbc/RCResRef* JNDI name to look up the data source.

Creating or changing a resource reference:

A resource reference supports application provider access to a resource (such as a data source, URL, or mail provider) using a logical name rather than the actual

name in the run time environment. This ability insulates the application provider from the run time configuration, and simplifies the process of changing the run time configuration.

Resource references are declared in the deployment descriptor by the application provider. At some point in the application deployment process, you must bind the resource reference to the actual name of the resource in the run time environment.

1. Start the Assembly Toolkit.
2. Import the enterprise application (EAR file) you want to change.
3. Display the resource references for the type of application component:
   - If an enterprise bean uses the resource reference:
     - Expand the name of the EAR file
     - Expand **EJB Modules**
     - Expand the EJB module wanted
     - Expand the section for the appropriate type of enterprise bean (**Session Beans** or **Entity Beans**)
     - Expand the enterprise bean
   - If a servlet uses the resource reference:
     - Expand the name of the EAR file
     - Expand **Web Modules**
     - Expand the Web module wanted
   - If an application client uses the resource reference:
     - Expand the name of the EAR file
     - Expand **Application Clients**
     - Expand the application client module wanted
4. Right-click the module whose resource references you want to change and click **Open With > Deployment Descriptor Editor**.
5. For servlets and application clients, click **Add**. For EJB modules, select the particular bean and click **Add**.
6. Select the resource reference option and click **Next**.
7. Specify the settings and click **Finish**.
8. Select the **References** tab and, under **WebSphere Extensions**, select an isolation level.
9. Under **WebSphere Bindings**, specify a JNDI name.
10. Close the deployment descriptor editor and save your changes.
11. Generate the code. (For details on this task, see the Information Center topic *Assembling > Applications > Assembly tools > Mapping enterprise beans to database tables > Generating code for EJB deployment*.)

Binding to a data source:

During either application assembly or deployment, you must bind the resource reference to the actual name of the resource in the run time environment. You can take this action in the Assembly Toolkit (ATK) or as one of the steps during installation of the application EAR file.

Bean-managed persistence bean: When developing your bean-managed persistence (BMP) bean you generally lack knowledge about the name of the data source on the target application server. In your code, do not look up the data source directly. Instead, you look up the resource reference from the *java:comp/env namespace* file. Let us assume that you look up the resource reference named *ref/ds* as illustrated in the code below.

```
javax.sql.DataSource dSource = (javax.sql.DataSource)((new InitialContext()).lookup
("java:/comp/env/ref/ds"));
```

Note: The previous code should be entered on one line; it is split for formatting purposes here.

In the ATK, you specify the name **ref/ds** in the Resource Reference page on the General Tab. If you know the name of the data source you can specify it in this Resource References page on the Bindings Tab. Note that if you do not specify it here , you must provide this Java Naming and Directory Interface (JNDI) name when you install the application EAR file.

Container-managed persistence bean: In a container-managed persistence (CMP) bean you do not specify the DataSource in the code. Instead, you specify the CMP Connection Factory JNDI name as a WebSphere binding property for each bean during application assembly.

Servlets and JavaServer Pages Files: In a servlet application, you look up the DataSource exactly as you look it up in the BMP bean case.

Access intent and isolation level:

The *access intent* service enables developers to precisely tune the management of application persistence.

Access intent enables developers to configure applications so that the EJB container and its agents can make performance optimizations for entity bean access. Entity beans and entity bean methods are configured with access intent policies. A policy is acted upon by either the combination of the WebSphere EJB container and Persistence Manager (for container-managed persistence (CMP) entities) or by bean-managed persistence (BMP) entities directly. Note that access intent policies apply to entity beans only.

Access intent -- isolation levels and update locks: The combination of concurrency and access type determines the isolation level for the persistence manager. The actual isolation level depends upon the particular database, as shown in the following table.

AccessIntent profile	**Isolation level**						**For Update**
	Sybase	**DB2**	**Oracle***	**Informix**	**Cloudscape**	**SQL Server**	
`wsPessimistic Update-Weakest LockAtLoad` (Default policy)	RR	RC	RR	RR	RR	RR	No (*Oracle, Yes)
`wsPessimistic Update`	RR	RC	RR	RR	RR	RR	Yes
`wsPessimistic Read`	RR	RC	RR	RR	RR	RR	No
`wsOptimistic Update`	RC	RC	RC	RC	RC	RC	No
`wsOptimistic Read`	RC	RC	RC	RC	RC	RC	No

AccessIntent profile	Isolation level						For Update
	Sybase	DB2	Oracle*	Informix	Cloudscape	SQL Server	
wsPessimisticUpdateNo-Collisions	RC	RC	RC	RC	RC	RC	No
wsPessimisticUpdate-Exclusive	S	S	S	S	S	S	Yes

- RC = JDBC Read committed
- RR = JDBC Repeatable read
- S = JDBC Serializable
- * Note: Oracle does not support JDBC Repeatable Read (RR). Therefore, wsPessimisticUpdate-weakestLockAtLoad and wsPessimisticUpdate behave the same way on Oracle as do wsPessismisticRead and wsOptimisticRead. Because of an Oracle restriction, the OracleXADataSource JDBC class **cannot** run with an S transaction isolation level. So you cannot use this class to run an application containing enterprise beans whose access intent policies are configured to cause the bean to load with RR isolation.

Custom finder SQL dynamic enhancement:

To ensure data integrity for applications using custom finders defined on Enterprise JavaBeans (EJB) version 1.1 home interfaces, WebSphere Application Server version 5.x uses custom finder Structured Query Language (SQL) dynamic enhancement to maintain correct SQL locking semantics.

WebSphere Application Server uses SQL clauses applied to the custom finder SQL statements for those custom finders defined with the *Update* attribute and certain method-level isolation level settings. These dynamic enhancements are applied only if the backend data store supports these clauses.

This support takes affect at run time when the run time attempts to execute Container Managed Persistence (CMP) persistence operations associated with the custom finders. To ensure that the SQL dynamic enhancements occur correctly for custom finders defined on an EJB version 1.1 home interface accessing a backend data store that requires the special SQL locking clauses, WebSphere Application Server provides new Java Virtual Machine (JVM) and bean (module) properties. These properties enable you to indicate which custom finders should be enhanced, provided the backend store supports the SQL clauses. For more information about these properties, see Custom finder SQL dynamic enhancement properties.

There are several important items to consider when using this functionality:

- this support **only** applies to EJB version 1.1 CMP Custom Finder methods
- Option A CMP beans and CMP beans involved in an inheritance relationship are not supported
- applications using this capability in WebSphere Application Server for z/OS Version 4.x continue to function, but you must address some compatibility issues:
  - the default behavior of the WebSphere Application Server Version 5.x product is the opposite of the Version 4.x product, that is, the default for 5.x is **not** to enhance custom finder SQL statements unless directed to by specific settings. If your WebSphere Application Server for z/OS installation relies on the automatic dynamic enhancement of all custom finders in all applications

installed, you must set the *com.ibm.websphere.ejbcontainer.customfinder.honorAccessIntent* indicator to **all**.

- If an application contains a bean which has the *com.ibm.websphere.persistence.bean.managed.custom.finder.access.intent* indicator set into its env-var settings, that indicator continues to be used, provided the dynamic SQL enhancement features of the Version 5.x product are enabled. For more information, see Custom finder SQL dynamic enhancement properties

Establishing custom finder SQL dynamic enhancement server-wide:

To establish this support on a server-wide basis, that is, dynamic SQL enhancement of all custom finders defined in all beans is enabled, use the following steps.

1. Open the administrative console.
2. Select **Servers**.
3. Select **Application Servers**.
4. Select the server you want to configure.
5. In the Additional Properties area, select **Process Definition**.
6. In the Additional Properties area, select **Control** or **Servant**. Select **Control** to define the property in the Control, **Servant** to define the property in the Servant.
7. In the Additional Properties area, select **Java Virtual Machine**.
8. Select **Custom Properties**.
9. Select **com.ibm.websphere.ejbcontainer.customfinder.honorAccessIntent** and enter a value of **all**. If the property is not present in the list, create a new property name, enter the name *com.ibm.websphere.ejbcontainer.customfinder.honorAccessIntent* and the value **all**.

Establishing custom finder SQL dynamic enhancement on a set of beans:

To establish this support for all custom finders defined on a set of beans use the following steps.

1. Open the administrative console.
2. Select **Servers**.
3. Select **Application Servers**.
4. Select the server you want to configure.
5. In the Additional Properties area, select **Process Definition**.
6. In the Additional Properties area, select **Control** or **Servant**. Select **Control** to define the property in the Control, **Servant** to define the property in the Servant.
7. In the Additional Properties area, select **Java Virtual Machine**.
8. Select **Custom Properties**.
9. Select **com.ibm.websphere.ejbcontainer.customfinder.honorAccessIntent** and enter a value that corresponds to a list of beans that need this support, with each bean's name separated from the others by a colon (:). For example, *beanA:beanB:beanC*.

   If the property is not present in the list, create a new property name, enter the name *com.ibm.websphere.ejbcontainer.customfinder.honorAccessIntent* and enter the list as the value.

Establishing custom finder SQL dynamic enhancement for specific custom finders:

To establish this support for specific custom finders use the following steps.

1. Start a J2EE application development environment of your choice.
2. Create or edit the application EAR file needing this support.
3. Check for an environmental variable called *com.ibm.websphere.ejbcontainer.customfinder.honorAccessIntent.methodLevel*. If the variable does not already exist, add it to the EAR file.
4. Give the variable a value that corresponds to a list of method names (including parameter lists) with each name separated from the others by a colon (:).
5. Deploy and install the application.

Disabling custom finder SQL dynamic enhancement for custom finders on a specific bean:

To disable this support for all custom finders defined on a specific bean, provided that the server wide support is enabled, follow these steps.

1. Start a J2EE application development environment of your choice.
2. Create or edit the application EAR file needing this support.
3. Check for an environmental variable called *com.ibm.websphere.persistence.bean.managed.custom.finder.access.intent* with a value of **true**. If the variable does not already exist, add it to the EAR file.
4. Ensure that the server-wide setting *com.ibm.websphere.ejbcontainer.customfinder.honorAccessIntent* is in place on the target server.
5. Deploy and install the application.

Custom finder SQL dynamic enhancement properties:

Use this page to modify custom finder SQL dynamic enhancement properties settings.

To ensure that the Structured Query Language (SQL) dynamic enhancements occur correctly for custom finders defined on an EJB 1.1 Home interface utilizing a backend data store that requires the special SQL locking clauses, the following Java Virtual Machine (JVM) and bean (module) properties are provided. These properties enable you to indicate which custom finders to enhance, provided the backend data store supports the SQL clauses.

To view this administrative console page, click **Servers** > **Application Servers** > *server* > **Process Definition** > **Control** (to define the property in the Control) or **Servant** (to define the property in the Servant) > **Java Virtual Machine** > **Custom Properties** .

com.ibm.websphere.ejbcontainer.customfinder.honorAccessIntent:

Used to indicate which Enterprise beans should have custom finder SQL dynamic enhancement enabled at runtime.

This property takes effect at the server level. Any EJB 1.1 home interface defined custom finder (prefix named *find*) that has *Update* as an access intent is a candidate for custom finder SQL dynamic enhancement based on its specified isolation level. If the backend data store requires special SQL semantics, they are applied. The particular SQL used varies according to the isolation level you choose for beans in the application, as well the backend data base being used. If set to **all**, custom finder SQL dynamic enhancement is enabled for all custom finders defined in any beans that are installed into the container. If set to **J2EENAME[:J2EENAME]**,

where *J2EENAME* is a fully qualified package or bean name, custom finder SQL dynamic enhancement is enabled for only the custom finders defined in the beans that are installed into the container and represented by the bean names denoted.

Note: Some of your applications might use custom finders that have been manually coded and already contain the SQL locking clauses, or keywords *ORDER BY* and *DISTINCT* on the *SELECT* operation. In these instances, if the runtime attempts SQL dynamic enhancement the possibility exists of introducing malformed SQL statements to the underlying backend data store. If an application contains these custom finders, then you must be careful when specifying the value for the JVM property *com.ibm.websphere.ejbcontainer.customfinder.honorAccessIntent*. A value of **all** causes custom finder SQL dynamic enhancement to occur for every custom finder method defined with an access intent of *Update* found in all beans installed in the application server, thus introducing malformed SQL for that subset of custom finders.

To prevent this from happening, ***do not*** set the server-wide setting to **all**. Instead, use the bean method level property, *com.ibm.websphere.ejbcontainer.customfinder.honorAccessIntent.methodLevel* to indicate on a per bean basis only those custom finder methods that should have the custom finder SQL dynamic enhancement executed on them at runtime.

com.ibm.websphere.ejbcontainer.customfinder.honorAccessIntent.methodLevel:

Used to indicate custom finder SQL dynamic enhancement be enabled at the method level on a particular bean.

When a bean is defined with this property set to a list of one or more custom finder methods, any custom finder (prefix named *find*) defined on the home interface that has a matching method name and parameter signature has SQL locking semantics applied at runtime. This occurs only if the custom finder method has an access intent of *Update* specified and the backend data store supports the SQL clauses. The particular SQL used varies according to the isolation level chosen for the application as well as the backend data store being used.

Data type String

com.ibm.websphere.persistence.bean.managed.custom.finder.access.intent:

Used by WebSphere Application Server for z/OS Version 4.x users to indicate that the SQL enhancement capability should **not be** applied to applications installed in the WebSphere Application Server for z/OS product.

The default behavior of the WebSphere Application Server for z/OS Version 4.x product is to perform the dynamic SQL enhancements. For those z/OS users choosing not to participate in dynamic SQL enhancement of custom finders in the Version 4.x product, this attribute is used to make this indication at both the bean and the server level.

At the bean level, a name/value pair consisting of this attribute name and a value of **true** disables the SQL enhancement of any custom finder defined on the given bean's home interface.

At the server level, an entry into the WebSphere Application Server for z/OS server property file with a value of **true** disables the SQL enhancement of all beans installed in the given server.

This custom finder enhancement attribute continues to be supported by the runtime at the bean level in the Version 5.x product. Its use as a server wide indicator has been deprecated by the fact that the default behavior of Version 5.x is to ***not*** dynamically enhance custom finder SQL.

Note: If your WebSphere Application Server for z/OS installation relies on the automatic dynamic enhancement of all custom finders in all applications installed, you should set the *com.ibm.websphere.ejbcontainer.customfinder.honorAccessIntent* indicator to **all**. If an application contains a bean that has the *com.ibm.websphere.persistence.bean.managed.custom.finder.access.intent* indicator set into its *env-var* settings, that indicator continues to be used, provided the dynamic SQL enhancement features of the Version 5.x product are enabled as described above.

Data type String

Some notes about precedence:

- The *com.ibm.websphere.ejbcontainer.customfinder.honorAccessIntent.methodLevel* attribute overrides any server-wide or bean level attribute setting
- Any bean listed through a *J2EE Name* in the *com.ibm.websphere.ejbcontainer.customfinder.honorAccessIntent* indicator causes dynamic enhancement to occur for custom finders defined for that bean, even if the default behavior is in effect for the server in question.
- The *com.ibm.websphere.persistence.bean.managed.custom.finder.access.intent* attribute disables a particular bean's use of this feature if the server-wide setting or bean setting is enabled and no method level settings are specified.

Data access from J2EE Connector Architecture applications

To access data from a J2EE Connector Architecture (JCA) compliant application in WebSphere Application Server, you can use encapsulated session beans and JCA connectors.

Use WebSphere Studio Application Developer Integration Edition (WSADIE) 4.1.1 to create a SessionBean that encapsulates the access to the backend. To find help on this subject, see the documentation for WSADIE.

Accessing data using J2EE Connector Architecture connectors

As indicated in the J2EE Connector Architecture (JCA) Specification, each enterprise information system (EIS) needs a resource adapter and a connection factory. This connection factory is then accessed through the following programming model. If you use WebSphere Studio Application Development (WSAD) tools, most of the following deployment descriptors and code are generated for you. This example shows the manual method of accessing an EIS resource.

For each EIS connection, do the following:

1. Declare a connection factory resource reference in your application component's deployment descriptors, as described in this example:

```
<resource-ref>
   <description>description</description>
   <res-ref-name>eis/myConnection</res-ref-name>
   <res-type>javax.resource.cci.ConnectionFactory</res-type>
   <res-auth>Application</res-auth>
</resource-ref>
```

2. Configure, during deployment, each resource adapter and associated connection factory through the console. See Configuring J2C resource adapters and Configuring J2C connection factories for more information.
3. Locate the corresponding connection factory for the EIS resource adapter using Java Naming and Directory Interface (JNDI) lookup in your application component, during run time.
4. Get the connection to the EIS from the connection factory.
5. Create an interaction from the Connection object.
6. Create an *InteractionSpec* object. Set the function to execute in the InteractionSpec object.
7. Create a Record instance for the input and output data used by function.
8. Execute the function through the Interaction object.
9. Process the record data from the function.
10. Close the connection.

The following code segment shows how an application component might create an interaction and execute it on the EIS:

```
javax.resource.cci.ConnectionFactory connectionFactory = null;
javax.resource.cci.Connection connection = null;
javax.resource.cci.Interaction interaction = null;
javax.resource.cci.InteractionSpec interactionSpec = null;
javax.resource.cci.Record inRec = null;
javax.resource.cci.Record outRec = null;

try {
// Locate the application component and perform a JNDI lookup
   javax.naming.InitialContext ctx = new javax.naming.InitialContext();
   connectionFactory = (javax.resource.cci.ConnectionFactory)
ctx.lookup("java:comp/env/eis/myConnection");

// create a connection
   connection = connectionFactory.getConnection();

// Create Interaction and an InteractionSpec
   interaction = connection.createInteraction();
   interactionSpec = new InteractionSpec();
   interactionSpec.setFunctionName("GET");

// Create input record
   inRec = new javax.resource.cci.Record();

// Execute an interaction
   interaction.execute(interactionSpec, inRec, outRec);

// Process the output...

} catch (Exception e) {
   // Exception Handling
}
finally {
    if (interaction != null) {
         try {
```

```
            interaction.close();
          }
          catch (Exception e) {/* ignore the exception*/}
    }
    if (connection != null) {
         try {
              connection.close();
         }
         catch (Exception e) {/* ignore the exception */}
    }
}
```

Example: Connection factory lookup

```
import javax.resource.cci.*;
import javax.resource.ResourceException;

import javax.naming.*;

import java.util.*;

/**
 * This class is used to look up a connection factory.
 */
public class ConnectionFactoryLookup {

   String jndiName = "java:comp/env/eis/SampleConnection";
   boolean verbose = false;

   /**
    * main method
    */
   public static void main(String[] args) {
      ConnectionFactoryLookup cfl = new ConnectionFactoryLookup();
      cfl.checkParam(args);

      try {
         cfl.lookupConnectionFactory();
      }
      catch(javax.naming.NamingException ne) {
   System.out.println("Caught this " + ne);
   ne.printStackTrace(System.out);
      }
      catch(javax.resource.ResourceException re) {
   System.out.println("Caught this " + re);
   re.printStackTrace(System.out);
      }
   }

 /**
  * This method does a simple Connection Factory lookup.
    *
    * After the Connection Factory is looked up, a connection is got from
  * the Connection Factory.  Then the Connection MetaData is retrieved
    * to verfiy the connection is workable.
  */
 public void lookupConnectionFactory()
  throws javax.naming.NamingException, javax.resource.ResourceException {

  javax.resource.cci.ConnectionFactory factory = null;
  javax.resource.cci.Connection conn = null;
  javax.resource.cci.ConnectionMetaData metaData = null;

  try {
   // lookup the connection factory
         if (verbose) System.out.println("Look up the connection factory...");

   InitialContext ic = new InitialContext();
```

```
 factory = (ConnectionFactory) ic.lookup(jndiName);

 // Get connection
       if (verbose) System.out.println("Get the connection...");
 conn = factory.getConnection();

 // Get ConnectionMetaData
 metaData = conn.getMetaData();

 // Print out the metadata Informatin.
 if (verbose) System.out.println("  **  EISProductName   :" + metaData.getEISProductName());
 if (verbose) System.out.println("      EISProductVersion:" + metaData.getEISProductName());
 if (verbose) System.out.println("      UserName         :" + metaData.getUserName());

       System.out.println("Connection factory "+jndiName+" is successfully looked up");
 }
 catch (javax.naming.NamingException ne) {
  // Connection factory cannot be looked up.
  throw ne;
 }
 catch (javax.resource.ResourceException re) {
  // Something wrong with connections.
  throw re;
 }
 finally {
  if (conn != null) {
   try {
    conn.close();
   }
   catch (javax.resource.ResourceException re) {
   }
  }
 }
}

 /**
  * Check and gather all the parameters.
  */
 private void checkParam(String args[]) {
int i = 0, j;
String arg;
char flag;
    boolean help = false;

// parse out the options
while (i < args.length && args[i].startsWith("-")) {
 arg = args[i++];

 // get the database name
 if (arg.equalsIgnoreCase("-jndiName")) {
  if (i < args.length)
   jndiName = args[i++];
  else {
   System.err.println("-jndiName requires a J2C Connection Factory JNDI name");
   break;
  }
 }
 else { // check for verbose, cmp , bmp
  for (j = 1; j < arg.length(); j++) {
   flag = arg.charAt(j);
   switch (flag) {
    case 'v' :
    case 'V' :
     verbose = true;
     break;
                case 'h' :
                case 'H' :
```

```
                    help = true;
                    break;
      default :
       System.err.println("illegal option " + flag);
       break;
     }
    }
   }
  }

  if ((i != args.length) || help) {
   System.err.println("Usage: java ConnectionFactoryLookup [-v] [-h]");
   System.err.println("    [-jndiName the J2C Connection Factory JNDI name]");
   System.err.println("-v=verbose");
   System.err.println("-h=this information");
   System.exit(1);
  }
   }

}
```

J2EE Connector Architecture migration tips

Previous WebSphere Application Server versions provided an initial implementation of the J2EE Connector Architecture (JCA) specification, Version 1.0. This implementation provided basic run time support based on the final JCA 1.0 Specification, but it was not a complete implementation.

The product now provides a complete implementation of the JCA 1.0 Specification, which supports:

- Connection sharing (*res-sharing-scope*).
- Get/use/close programming model for connection handles.
- Get/use/cache programming model for connection handles.
- *XA*, *Local*, and *No Transaction* models of resource adapters, including XA recovery.
- Security options A and C per the specification.

If you move from one of the earlier implementations of the J2EE Connector Architecture to the current implementation, be aware of the following:

- This version supports the *res-sharing-scope* tag within the resource reference (resource-ref) element. This tag was not available in previous versions and defaulted to *shareable* connections. Version 5.0 supports **both** shareable and unshareable connections.
- The current product supports the Web container. Both enterprise bean and Web components can utilize the J2EE Connector Architecture.
- Both connection handle usage patterns (get/use/close and get/use/cache) are supported. The get/use/close pattern indicates that a connection is retrieved, used, and closed all within the same transaction or method boundary. The get/use/cache pattern indicates that you can cache a connection across transaction or method boundaries.
- The current version supports additional authentication mechanisms. The capability to support Options A and C per the JCA specification is provided, as well as support for *res-auth* settings of either *Application* or *Container*. In previous versions, the *res-auth* setting was basically ignored, therefore it was treated as if *res-auth* was set to *Application*. If your existing applications had *res-auth* set to *Container*, they might behavior differently if you install them into a current environment without any changes.

- You can no longer specify pool and subpool names. The pool name is based on the data source or connection factory's Java Naming and Directory Interface (JNDI) name. Subpools were eliminated to provide better performance.

For applications that use WebServices and JCA Connectors, there are some additional points to be aware of:

- Applications generated on WebSphere Studio Application Developer -- Integration Edition Version 4.1.1 can run on WebSphere Application Server Version 4.0.4 and WebSphere Application Server Enterprise Version 4.1.
- Applications generated on WebSphere Studio Application Developer -- Integration Edition Version 5.0 can run on WebSphere Application Server Version 5.0 and WebSphere Application Server Enterprise Version 5.0.
- Applications generated on WebSphere Studio Application Developer -- Integration Edition Version 4.1.1 can run unchanged on WebSphere Application Server Enterprise Version 5.0, but can only run on WebSphere Application Server Version 5.0 if the applications are regenerated using WebSphere Studio Application Developer -- Integration Edition Version 5.0 tools.

This limitation is because of the *wsd14j.jar* file. As delivered in WebSphere Application Server Enterprise Version 4.1, the file is not fully compliant with JSR 110 (JSR 110 was not final at the time that 4.1 shipped). The *wsd14j.jar* file shipped with WebSphere Application Server Version 5.0 is compliant. However, because most of the classes have the same package names and interfaces, BUT NOT ALL, the two *wsd14j.jar* files cannot co-exist in the same WebSphere Application Server installation.

Data access from an enterprise entity bean

Container-managed persistence (CMP) developers can use *access intent* to provide hints on how the application server run time should manage the details of persistence without having to explicitly manage any of the persistence logic from within their application.

However, there are still situations where developers must develop bean-managed persistence (BMP) entity beans. Because the only meaningful difference between BMP and CMP components is who provides the persistence logic, BMP beans should leverage access intent hints just the same as the application server does on behalf of CMP beans. This ability becomes especially important when BMP entities and CMP entities want to share connections. BMP beans configured with the same concurrency as the CMP beans and implemented to the same isolation level mapping as the CMP can share connections.

Developers can apply access intent policies to BMP entity beans as well as to CMP entity beans. It is expected that BMP developers use only those access intent attributes that are important to a particular BMP bean. The access intent service interface is bound into the *java:comp namespace* for each particular BMP bean. The access intent policy retrieved from the access intent service is current from the time that the *ejbLoad* process is called until the time that the *ejbStore* process completes its invocation.

Data access bean types

Note: WebSphere Application Server does not support JDBC 3.0.

Data access beans are essentially a class library that makes it easier to access a database. The library contains a set of beans with methods that access the database through the JDBC API. There are several sets of classes referred to as data access beans. To make things clearer, you can refer to the classes by the name of the JAR file that contains them:

databeans.jar - This JAR file ships with the WebSphere Application Server. This file contains classes that enable you to access the database using the JDBC API.

ivjdab.jar - This JAR file ships with Visual Age for Java (VAJ). This file contains all of the classes in the *databeans.jar* file and classes that support easy use of the data access beans from the VAJ Visual Composition Editor.

dbbeans.jar - This JAR file ships with WebSphere Studio Site Developer (WSSD) and WebSphere Studio Application Developer (WSAD). This file contains a set of data access beans to more closely conform to the JDBC 2.0 *RowSet* standard.

For the current product, data access beans remain unchanged from WebSphere Application Server Version 4.0. The *com.ibm.db* package is provided to support existing applications that use data access beans.

IBM strongly suggests that any **new** applications using data access beans be developed using the *com.ibm.db.beans* package that is provided with WebSphere Studio Application Developer (WSAD).

If you want to continue using applications that use the *com.ibm.db* package, see the WebSphere Application Server Version 4.0 documentation concerning data access beans. An example is shown here: Example: Using data access beans in Version 4.0.

If you want to create new applications that use the *com.ibm.db.beans* package, see the WSAD documentation concerning data access beans. An example is shown here: Example: Using data access beans in Version 5.0

Example: Using data access beans in Version 4.0

```
package examples;

import com.ibm.db.uibeans.*;
import com.ibm.db.*;
/**
 * This type was created in VisualAge.
 */

public class SelectStatementExample {
/**
 * GenericTest constructor comment.
 */
public SelectStatementExample() {
 super();
}
/**
 * Starts the application.
 * @param args an array of command-line arguments
 */
public static void main(java.lang.String[] args) {
 // Objects

 SelectStatement stmt = new SelectStatement();
 DatabaseConnection conn = new DatabaseConnection();
 StatementMetaData metaData = new StatementMetaData();
 SelectResult result;
```

```
// Set properties for connection
conn.setDriverName("COM.ibm.db2.jdbc.app.DB2Driver");
conn.setDataSourceName("jdbc:db2:Sample");
conn.setUserID("userid");
conn.setPassword("password");

// Set SQL statement
metaData.setSQL("SELECT * FROM DEPARTMENT");

// Associate connection and metadata with stmt
stmt.setConnection(conn);
stmt.setMetaData(metaData);

try {
 // Execute SQL statement
 stmt.execute();

 // Process results
 result = stmt.getResult();
 for (int i = 1; i <= result.getNumRows(); i++) {
  System.out.println(result.getColumnValueToString(1));
  System.out.println(result.getColumnValueToString(2));
  result.nextRow();
 }

 // Release JDBC resources
 result.close();

 // Close the database connection
 conn.disconnect();

} catch (DataException ex) {
 ex.printStackTrace();
}
}
}
```

Example: Using data access beans in Version 5.0

```
package example;
import com.ibm.db.beans.*;
import java.sql.SQLException;

public class DBSelectExample {

 public static void main(String[] args) {

  DBSelect select = null;

  select = new DBSelect();
  try {

   // Set database connection information
   select.setDriverName("COM.ibm.db2.jdbc.app.DB2Driver");
   select.setUrl("jdbc:db2:SAMPLE");
   select.setUsername("userid");
   select.setPassword("password");

   // Specify the SQL statement to be executed
   select.setCommand("SELECT * FROM DEPARTMENT");

   // Execute the statement and retrieve the result set into the cache
   select.execute();

   // If result set is not empty
   if (select.onRow()) {
    do {
```

```
    // display first column of result set
    System.out.println(select.getColumnAsString(1));
    System.out.println(select.getColumnAsString(2));
   } while (select.next());
  }

  // Release the JDBC resources and close the connection
  select.close();

 } catch (SQLException ex) {
  ex.printStackTrace();
 }
}
}
```

Accessing data from application clients

To access a database directly from a J2EE application client, you retrieve a *javax.sql.DataSource* object from a resource reference configured in the client deployment descriptor. This resource reference is configured as part of the deployment descriptor for the client application, and provides a reference to a preconfigured data source object.

Note that data access from an application client uses the JDBC driver connection functionalities directly from the client side. It does not take advantage of the additional pooling support available in the application server run time. For this reason, your client application should utilize an enterprise bean running on the server side to perform data access. This enterprise bean can then take advantage of the connection reuse and additional added functionality provided by the product run time.

1. Import the appropriate JDBC API and naming packages:
   ```
   import java.sql.*;
   import javax.sql.*;
   import javax.naming.*;
   ```
2. Create the initial naming context:
   ```
   InitialContext ctx = new InitialContext();
   ```
3. Use the *InitialContext* to look up a data source object from a resource reference.
   ```
   javax.sql.DataSource ds = (DataSource)ctx.lookup("java:comp/env/jdbc/myDS");
   //where jdbc/myDS is the name of the resource reference
   ```
4. Get a *java.sql.Connection* from the data source.
   - If no user ID and password are required for the connection, or if you are going to use the *defaultUser* and *defaultPassword* that are specified when the data source is created in the Application Client Resource Configuration tool (ACRCT) in a future step:
     ```
     java.sql.Connection conn = ds.getConnection();
     ```
   - Otherwise, you should make the connection with a specific user ID and password:
     ```
     java.sql.Connection conn = ds.getConnection("user", "password");
     //where user and password are the user id and password for the connection
     ```
5. Run a database query using the *java.sql.Statement*, *java.sql.PreparedStatement*, or *java.sql.CallableStatement* interfaces as appropriate.
   ```
   Statement stmt = conn.createStatement();
   String query   = "Select FirstNme from " + owner.toUpperCase() + ".Employee
   where LASTNAME = '" + searchName + "'";
   ResultSet rs   = stmt.executeQuery(query);
   while (rs.next()) {    firstNameList.addElement(rs.getString(1));
   }
   ```

Note: The `String query` statement in the previous block of code should be entered on one line; it is split for formatting purposes here.

6. Close the database objects used in the previous step, including any *ResultSet*, *Statement*, *PreparedStatement*, or *CallableStatement* objects.
7. Close the connection. Ideally, you should close the connection in a *finally* block of the *try...catch* wrapped around the database operation. This action ensures that the connection gets closed, even in the case of an exception.

```
conn.close();
```

Connection thread identity

WebSphere Application Server for z/OS allows you to assign a thread identifier as an owner of a connection, when you first obtain the connection. The *thread identity* function only applies to J2EE Connector Architecture (JCA) resource adapters and Relational Resource Adapter (RRA) wrappered Java Database Connectivity (JDBC) providers that support the use of thread identity for connection ownership.

The following table lists the JCA resource adapter, the JDBC provider, and the WebSphere MQ JMS Provider configurations that support thread identity and OS thread security . It also provides the level of thread identity support:

Connectors	Thread identity support	thread security
IMS Connector - local `ConnectionFactory` configuration	ALLOWED	Not supported
IMS Connector - remote `ConnectionFactory` configuration	NOTALLOWED	Not supported
CTG CICSECIConnector - local `ConnectionFactory` configuration	ALLOWED	Not supported
CTG CICSECIConnector - remote `ConnectionFactory` configuration	NOTALLOWED	Not supported
IMS JDBC Connector - local `ConnectionFactory` configuration (By default, IMS JDBC only supports this type of configuration.)	REQUIRED	True
RRA DB2 for z/OS local JDBC provider - data sources configured to the local DB2	ALLOWED	True
RRA DB2 Universal JDBC Driver Provider using Type 2 connectivity	ALLOWED	True
RRA DB2 Universal JDBC Driver Provider using Type 4 connectivity	NOTALLOWED	Not supported

WebSphere Application Server for z/OS allows resource adapters and JDBC providers to define the level of thread identity support for the defined connection factories or data sources. The level of support can be:

- **`ALLOWED`**, which indicates thread identity for connection ownership is allowed for this configuration.
- **`NOTALLOWED`**, which indicates thread identity for connection ownership is not allowed for this configuration.
- **`REQUIRED`**, which indicates thread identity for connection ownership is required.

The thread identity function is only available in those server configurations where JCA connectors or JDBC providers access local z/OS resources through callable

(not TCP/IP) interfaces. So, for example, CICS and IMS provide thread identity support only if the target CICS or IMS is configured on the same system as the z/OS WebSphere Application Server.

To use thread identity when getting connections to a connection factory or JDBC data source for your application, you must specify **resauth=Container** for the connection factory or JDBC data source. Use the the Application Assembly Tool (AAT) or WebSphere Studio Application Developer Integration Edition (WSADIE) to indicate the **resauth=Container** setting.

When the level of thread identity support provided by the connector configuration is **ALLOWED**, if you want to use thread identity for the connections, you cannot specify a Container-managed alias when you define the connection factory or JDBC data source. If you specify a Container-managed alias, the userid defined by the alias is assigned as the owning id for the connections obtained by the application.

When the JDBC provider supports thread identity, the thread identity function is only used when data sources configured for that provider are used by Version 2.0 EJB modules and Version 2.3 servlets.

WebSphere Application Server for z/OS also allows supported resource adapters and JDBC providers to enable *OS thread security* in conjunction with thread identity support. You can use OS thread security when:

- The server configuration supports both thread identity and thread security.
- 5.1+ The **Connection Manager RunAs Identity Enabled** property is enabled.

  You can configure the server to allow **Connection Manager RunAs Identity Enabled** support by navigating through the following panels in the administrative console:

  Security > Global Security > z/OS Security Options

  On the **z/OS Security Options** panel, check the box entitled, **Connection Manager RunAs Identity Enabled**, and then select **Apply**.

If these conditions are met, the system creates an access control environment element (ACEE) for the user associated with the thread.

5.1+ Users of previous versions of WebSphere Application Server for z/OS will note that the instructions for enabling OS Thread Security have changed. Previously, OS Thread Security was enabled using a checkbox named Enable Synch to Thread. Users who wish to enable OS Thread Security must now use the checkbox named Connection Manager RunAs Identity Enabled

Using thread identity support

Perform the following steps to enable the thread identity function for the connection factories or JDBC provider data sources created with the supported JCA resource adapters and JDBC providers:

1. Define **resauth=Container** for the application resource reference (the equivalent for CMPs is `resourceAuthorization=Container`)
2. Ensure the JCA resource adapters, WebSphere MQ JMS Provider, or JDBC providers support the thread identity function.

   Review the supported resource adapters and datasource providers, and the level of support: REQUIRED, ALLOWED, and NOTALLOWED.

If the adapter or provider is not listed, then thread identity support is NOTALLOWED, by default.

3. Set the **Container-managed authentication alias** to NULL, if you configure the connector locally.

   When the connector is configured locally, the resource adapter determines the level of thread identity support as **ALLOWED**. If thread identity support is allowed and you specify **Container-managed authentication alias** as NULL, the connector uses the current thread identity as the owner for each connection that is created.

   When the resource adapter, WebSphere MQ JMS Provider, or JDBC provider determines that the level of thread identity support is **REQUIRED**, any specification for the **Container-managed authentication alias** is ignored. Thread identity support in this case always applies.

4. Determine connector behavior when global security is a factor.

   Note: With Bean-Managed Persistence (BMP) beans, if you obtain a connection under the `ejbLoad()` or `ejbStore()` functions during pre-invoke or post-invoke method processing, your thread identity support does not become the **RunAs** identity because the container during this processing is running under server identity. (For details on the **RunAs** identity, see the Information Center topic *Deploying > Security > Deploying secured applications > Delegations*.) With BMP beans, instead of using thread identity, specify a Container-managed alias to associate the user with the connection.

Security states with thread identity support

The combinations of global security, server configurations, connector configurations, and container-managed alias support determine the processing that results when you use the thread identity function. Thread identity support is only available with specific resource adapters and JDBC providers. Review the supported resource adapters and JDBC providers to determine if you can use thread identity. If your resource adapter or JDBC provider is in the supported list, use the following tables to determine the processing that occurs, based on the settings of the specified properties:

Table 1. Table 1. Security state

Global security enabled?	
Yes	No
Go to table 2.	Go to table 3.

Table 2. Table 2. Global security enabled

Container-managed alias specified?			
No		Yes	
Connector Allows or Requires Thread Identity?		Connector Requires Thread Identity?	
No	Yes	No	Yes

Table 2. Table 2. Global security enabled (continued)

Processing is dependent on connector: may throw exception may default to connector user/password custom properties	Connector requires OS thread security?			Use specified alias	Connector requires OS thread security?		
	No	Yes			No	Yes	
	Use identity associated with current thread	Server Sync-To-Thread enabled?			Use identity associated with current thread	Server Sync-To-Thread enabled?	
		No	Yes			No	Yes
		Use Server identity	Use identity associated with current thread			Use server identity	Use identity associated with current thread

Table 3. Table 3. Global Security is not enabled

Container-managed alias specified?			
No		Yes	
Connector ALLOWS or REQUIRES thread identity o be used when getting a connection		Connector REQUIRES thread identity to be used when getting a connection?	
No	Yes	No	Yes
Processing is dependent on connector: • May throw exception • May default to connector user/password custom properties	User server identity	Use specified alias	Use server identity

Exceptions pertaining to data access

All enterprise bean container-managed persistence (CMP) beans under the EJB 2.0 Specification receive a standard *EJBException* when an operation fails.

JDBC applications receive a standard *SQLException* if any JDBC operation fails.

The product provides special exceptions for its relational resource adapter (RRA), to indicate that the connection currently held is no longer valid. The *ConnectionWaitTimeoutException* indicates that the application timed out trying to get a connection. The *StaleConnectionException* indicates that the connection is no longer valid.

Connection wait timeout

The ConnectionWaitTimeout exception indicates that the application has waited for the number of seconds specified by the connection timeout setting and has not received a connection. This situation can occur when the pool is at maximum size and all of the connections are in use by other applications for the duration of the

wait. In addition, there are no connections currently in use that the application can share because either the connection properties do not match, or the connection is in a different transaction.

When using a Version 4.0 data source, the ConnectionWaitTimeout throws an exception whose class is *com.ibm.ejs.cm.pool.ConnectionWaitTimeoutException*.

For connection factories, the ConnectionWaitTimeout throws a ResourceException whose class is *com.ibm.websphere.ce.j2c.ConnectionWaitTimeoutException*.

Finally, Version 5.0 data sources throw an SQLException subclass called *com.ibm.websphere.ce.cm.ConnectionWaitTimeoutException*.

Example: Handling data access exception - ConnectionWaitTimeoutException (for the JDBC API): In all cases in which the *ConnectionWaitTimeoutException* is caught, there is very little to do for recovery.

The following code fragment shows how to use this exception in the JDBC API:

```
public void test1() {
 java.sql.Connection conn = null;
 java.sql.Statement stmt = null;
 java.sql.ResultSet rs = null;

 try {
  // Look for datasource
  java.util.Properties props = new java.util.Properties();
  props.put(
   javax.naming.Context.INITIAL_CONTEXT_FACTORY,
   "com.ibm.websphere.naming.WsnInitialContextFactory");
  ic = new javax.naming.InitialContext(props);
  javax.sql.DataSource ds1 =
   (javax.sql.DataSource) ic.lookup(jndiString);

  // Get Connection.
  conn = ds1.getConnection();
  stmt = conn.createStatement();
  rs = stmt.executeQuery("select * from mytable where this = 54");
 }
 catch (com.ibm.websphere.ce.cm.ConnectionWaitTimeoutException cwte) {
  //notify the user that the system could not provide a
  //connection to the database.  This usually happens when the
  //connection pool is full and there is no connection
  //available for to share.
 }
 catch (java.sql.SQLException sqle) {
  // handle other database problems.
 }
 finally {
  if (rs != null)
   try {
    rs.close();
   }
   catch (java.sql.SQLException sqle1) {
   }
  if (stmt != null)
   try {
    stmt.close();
   }
   catch (java.sql.SQLException sqle1) {
   }
  if (conn != null)
   try {
    conn.close();
```

```
    }
    catch (java.sql.SQLException sqle1) {
    }
  }
 }
```

Example: Handling data access exception - ConnectionWaitTimeoutException (for J2EE Connector Architecture): In all cases in which the *ConnectionWaitTimeoutException* is caught, there is very little to do for recovery.

The following code fragment shows how to use this exception in J2EE Connector Architecture (JCA):

```
/**
 * This method does a simple Connection test.
 */
public void testConnection()
   throws javax.naming.NamingException, javax.resource.ResourceException,
com.ibm.websphere.ce.j2c.ConnectionWaitTimeoutException {
```

Note: The previous line of code should be entered on one line; it is split here for formatting purposes.

```
   javax.resource.cci.ConnectionFactory factory = null;
   javax.resource.cci.Connection conn = null;
   javax.resource.cci.ConnectionMetaData metaData = null;
   try {
      // lookup the connection factory
      if (verbose) System.out.println("Look up the connection factory...");
try {
factory =
  (javax.resource.cci.ConnectionFactory) (new InitialContext()).
lookup("java:comp/env/eis/Sample");
```

Note: The previous line of code should be entered on one line; it is split here for formatting purposes.

```
}
   catch (javax.naming.NamingException ne) {
      // Connection factory cannot be looked up.
      throw ne;
   }
    // Get connection
      if (verbose) System.out.println("Get the connection...");
      conn = factory.getConnection();
      // Get ConnectionMetaData
      metaData = conn.getMetaData();
      // Print out the metadata Informatin.
      System.out.println("EISProductName is " + metaData.getEISProductName());
   }
   catch (com.ibm.websphere.ce.j2c.ConnectionWaitTimeoutException cwtoe) {
      // Connection Wait Timeout
      throw cwtoe;
   }
   catch (javax.resource.ResourceException re) {
      // Something wrong with connections.
      throw re;
   }
   finally {
      if (conn != null) {
         try {
            conn.close();
         }
     catch (javax.resource.ResourceException re) {
```

```
            }
        }
    }
}
```

Stale connections

The product provides a special subclass of *java.sql.SQLException* when using connection pooling to access a relational database. This *com.ibm.websphere.ce.cm.StaleConnectionException* subclass exists in both a WebSphere 4.0 data source and in the new data source using the relational resource adapter, and is used to indicate that the connection currently held is no longer valid. This situation can occur for many reasons, including the following:

- The application tries to get a connection and fails, as when the database is not started.
- A connection is no longer usable because of a database failure. When an application tries to use a previously obtained connection, the connection is no longer valid. In this case, all connections currently in use by the application can get this error when they try to use the connection.
- The connection is orphaned (because the application had not used it in at most two times the value of the *unused timeout* setting) and the application tries to use the orphaned connection. This case applies only to Version 4.0 data sources.
- The application tries to use a JDBC resource, such as a statement, obtained on a stale connection.
- A connection is closed by the Version 4.0 data source *auto connection cleanup* and is no longer usable. Auto connection cleanup is the standard mode in which connection management operates. This mode indicates that at the end of a transaction, the transaction manager closes all connections enlisted in that transaction. This enables the transaction manager to ensure that connections are not held for excessive periods of time and that the pool does not reach its maximum number of connections prematurely.

  One ramification of having the transaction manager close the connections and return the connection to the free pool after a transaction ends, is that an application cannot obtain a connection in one transaction and try to use it in another transaction. If the application tries this, a *StaleConnectionException* is thrown because the connection is already closed.

In the case of trying to use an orphaned connection or a connection cleaned up by auto connection cleanup, a *StaleConnectionException* indicates that the application has attempted to use a connection already returned to the connection pool. It does not indicate an actual problem with the connection. However, other cases of a *StaleConnectionException* indicate that the connection to the database has gone bad, or *stale*. Once a connection has gone stale, you cannot recover it, and you must completely close the connection rather than returning it to the pool.

Detecting stale connections

When a connection to the database becomes stale, operations on that connection result in an *SQLException* from the JDBC driver. Because an *SQLException* is a rather generic exception, it contains state and error code values that you can use to determine the meaning of the exception. However, the meanings of these states and error codes vary depending on the database vendor. The connection pooling run time maintains a mapping of which SQL state and error codes indicate a *StaleConnectionException* for each database vendor supported. When the connection pooling run time catches any *SQLException*, it checks to see if this *SQLException* is considered a *StaleConnectionException* for the database server in use.

Recovering from stale connections

Recovering from stale connections is a joint effort between the application server run time and the application developer. From an application server perspective, the connection pool is purged based on its *PurgePolicy* setting.

Explicitly catching a *StaleConnectionException* is not required in an application. Because applications are already required to catch *java.sql.SQLException*, and *StaleConnectionException* extends *SQLException*, *StaleConnectionException* can be thrown from any method that is declared to throw *SQLException*, and is caught automatically in the general catch-block. However, explicitly catching *StaleConnectionException* makes it possible for an application to recover from bad connections. When application code catches *StaleConnectionException*, it should take explicit steps to handle the exception.

Example: Handling data access exception - StaleConnectionException: When an application receives a *StaleConnectionException* on a database operation, it indicates that the connection currently held is no longer valid. While it is possible to get a *StaleConnectionException* on any database operation, the most common time to see a *StaleConnectionException* thrown is the first time that a connection is used, just after it is retrieved. Because connections are pooled, a database failure is not detected until the operation immediately following its retrieval from the pool, which is the first time communication to the database is attempted. It is only when a failure is detected that the connection is marked stale. *StaleConnectionException* occurs less often if each method that accesses the database gets a new connection from the pool.

Many *StaleConnectionException*s are caused by intermittent problems with the network of the database server. Obtaining a new connection and retrying the operation can result in successful completion without exceptions to the end user. In some cases it is advantageous to add a small wait time between the retries to give the database server more time to recover. However, applications should not retry operations indefinitely, in case the database is down for an extended period of time.

Before the application can obtain a new connection for a retry of the operation, roll back the transaction in which the original connection was involved and begin a new transaction. You can break down details on this action into two categories:
Objects operating in a bean-managed global transaction context begun in the same method as the database access.

A servlet or session bean with bean-managed transactions (BMT) can start a global transaction explicitly by calling *begin()* on a *javax.transaction.UserTransaction* object, which you can retrieve from naming or from the bean *EJBContext* object. To commit a bean-managed transaction, the application calls *commit()* on the *UserTransaction* object. To roll back the transaction, the application calls *rollback()*. Entity beans and non-BMT session beans cannot explicitly begin global transactions.

If an object that explicitly started a bean-managed transaction receives a *StaleConnectionException* on a database operation, close the connection and roll back the transaction. At this point, the application developer can decide to begin a new transaction, get a new connection, and retry the operation.

The following code fragment shows an example of handling *StaleConnectionException*s in this scenario:

```
//get a userTransaction
javax.transaction.UserTransaction tran = getSessionContext().getUserTransaction();
//retry indicates whether to retry or not
//numOfRetries states how many retries have
// been attempted
boolean retry = false;
int numOfRetries = 0;
java.sql.Connection conn = null;
java.sql.Statement stmt = null;
do {
  try {
    //begin a transaction
    tran.begin();
    //Assumes that a datasource has already been obtained
    //from JNDI
    conn = ds.getConnection();
    conn.setAutoCommit(false);
    stmt = conn.createStatement();
    stmt.execute("INSERT INTO EMPLOYEES VALUES
              (0101, 'Bill', 'R', 'Smith')");
    tran.commit();
    retry = false;
  } catch(com.ibm.websphere.ce.cm.StaleConnectionException
         sce)
  {
    //if a StaleConnectionException is caught
    // rollback and retry the action
    try {
      tran.rollback();
    } catch (java.lang.Exception e) {
      //deal with exception
      //in most cases, this can be ignored
    }
    if (numOfRetries < 2) {
      retry = true;
      numOfRetries++;
    } else {
      retry = false;
    }
  } catch (java.sql.SQLException sqle) {
    //deal with other database exception
    retry = false
  } finally {
    //always cleanup JDBC resources
    try {
      if(stmt != null) stmt.close();
    } catch (java.sql.SQLException sqle) {
      //usually can ignore
    }
    try {
      if(conn != null) conn.close();
    } catch (java.sql.SQLException sqle) {
      //usually can ignore
    }
  }
} while (retry) ;
```

Objects operating in a global transaction context and transaction not begun in the same method as the database access.

When the object which receives the StaleConnectionException does not have direct control over the transaction, such as in a container-managed transaction case, the object must mark the transaction for rollback, and then indicate to its caller to retry the transaction. In most cases, you can do this by throwing an application exception which indicates to retry that operation. However this action is not always allowed, and often a method is defined only to throw a particular exception. This is the case with the

ejbLoad and ejbStore methods on an enterprise bean. The next two examples explain each of these scenarios.

Example 1: Database access method can throw application exception.

When the method that accesses the database is free to throw whatever exception is required, the best practice is to catch *StaleConnectionException* and rethrow some application exception that yopu can interpret to retry the method. The following example shows an EJB client calling a method on an entity bean with transaction demarcation *TX_REQUIRED*, which means that the container begins a global transaction when *insertValue* is called:

```
public class MyEJBClient {
//... other methods here ...
public void myEJBClientMethod()
{
MyEJB myEJB = myEJBHome.findByPrimaryKey("myEJB");
boolean retry = false;
do {
try {
retry = false;
myEJB.insertValue();
}
catch(RetryableConnectionException retryable) {
retry = true;
}
catch(Exception e) { /* handle some other problem */ }
} while (retry);
}
}  //end MyEJBClient

public class MyEJB implements javax.ejb.EntityBean {
//... other methods here ...
public void insertValue() throws RetryableConnectionException,
java.rmi.RemoteException {
try
{
conn = ds.getConnection();
stmt = conn.createStatement();
stmt.execute("INSERT INTO my_table VALUES (1)");
}
catch(com.ibm.websphere.ce.cm.StaleConnectionException
sce) {
getSessionContext().setRollbackOnly();
throw new RetryableConnectionException();
}
catch(java.sql.SQLException sqle) {
//handle other database problem
}
finally {
21
//always cleanup JDBC resources
try {
if(stmt != null) stmt.close();
} catch (java.sql.SQLException sqle) {
//usually can ignore
}
try {
if(conn != null) conn.close();
} catch (java.sql.SQLException sqle) {
//usually can ignore
}
}
}
}  //end MyEJB
```

MyEJBClient first gets a *MyEJB* bean from the home interface, assumed to have been previously retrieved from the Java Naming and Directory Interface (JNDI). It then calls *insertValue()* on the bean. The method on the bean gets a connection and tries to insert a value into a table. If one of the methods fails with a *StaleConnectionException*, it marks the transaction for *rollbackOnly* (which forces the caller to roll back this transaction) and throws a new *RetryableConnectionException*, cleaning up the resources before the exception is thrown. The *RetryableConnectionException* is simply an application-defined exception that tells the caller to retry the method. The caller monitors *RetryableConnectionException* and, if it is caught, retries the method. In this example, because the container is beginning and ending the transaction, no transaction management is needed in the client or the server. Of course, the client could start a bean-managed transaction and the behavior would still be the same, provided that the client also committed or rolled back the transaction.

Example 2: Database access method can throw only RemoteException or EJBException.

Not all methods are allowed to throw exceptions defined by the application. If you use bean-managed persistence (BMP), use the *ejbLoad()* and *ejbStore()* methods to store the bean state. The only exceptions thrown from these methods are *java.rmi.RemoteException* or *javax.ejb.EJBException*, so you cannot use something similar to the previous example.

If you use container-managed persistence (CMP), the container persists the bean, and it is the container that sees *StaleConnectionException*. If a stale connection is detected, by the time the exception is returned to the client it is simply a *RemoteException*, and so a simple catch-block does not suffice. There is a way to determine if the root cause of a *RemoteException* is a *StaleConnectionException*. When *RemoteException* is thrown to wrap another exception, the original exception is usually retained. All *RemoteException* instances have a detail property, which is of type *java.lang.Throwable*. With this detail, you can trace back to the original exception and, if it is a *StaleConnectionException*, retry the transaction. In reality, when one of these RemoteExceptions flows from one Java Virtual Machine API to the next, the detail is lost, so it is better to start a transaction in the same server as the database access occurs. For this reason, the following example shows an entity bean accessed by a session bean with bean-managed transaction demarcation.

```
public class MySessionBean extends javax.ejb.SessionBean {
... other methods here ...
public void mySessionBMTMethod() throws
java.rmi.RemoteException
{
javax.transaction.UserTransaction tran =
getSessionContext().getUserTransaction();
boolean retry = false;
do {
try {
retry = false;
tran.begin();
// causes ejbLoad() to be invoked
myBMPBean.myMethod();
// causes ejbStore() to be invoked
tran.commit();
```

```
}
catch(java.rmi.RemoteException re) {
try { tran.rollback();
}
catch(Exception e) {
//can ignore
}
if (causedByStaleConnection(re))
retry = true;
else
throw re;
}
catch(Exception e) {
// handle some other problem
}
finally {
//always cleanup JDBC resources
try {
if(stmt != null) stmt.close();
} catch (java.sql.SQLException sqle) {
//usually can ignore
}
try {
if(conn != null) conn.close();
} catch (java.sql.SQLException sqle) {
//usually can ignore
}
}
} while (retry);
}

public boolean causedByStaleConnection(java.rmi.RemoteException
remoteException)
{
java.rmi.RemoteException re = remoteException;
Throwable t = null;
while (true) {
t = re.detail;
try { re = (java.rmi.RemoteException)t; }
catch(ClassCastException cce) {
return (t instanceof
com.ibm.websphere.ce.cm.StaleConnectionException);
}
}
}
}
public class MyEntityBean extends javax.ejb.EntityBean {
... other methods here ...
public void ejbStore() throws java.rmi.RemoteException
{
try {
conn = ds.getConnection();
stmt = conn.createStatement();
stmt.execute("UPDATE my_table SET value=1 WHERE
primaryKey=" + myPrimaryKey);
}
catch(com.ibm.websphere.ce.cm.StaleConnectionException
sce) {
//always cleanup JDBC resources
try {
if(stmt != null) stmt.close();
} catch (java.sql.SQLException sqle) {
//usually can ignore
}
try {
if(conn != null) conn.close();
} catch (java.sql.SQLException sqle) {
```

```
//usually can ignore
}
// rollback the tran when method returns
getEntityContext().setRollbackOnly();
throw new java.rmi.RemoteException("Exception occurred in
ejbStore", sce);
}
catch(java.sql.SQLException sqle) {
// handle some other problem
}
}
}
```

In *mySessionBMTMethod()*:

- the session bean first retrieves a *UserTransaction* object from the session context and then begins a global transaction.
- Next, it calls a method on the entity bean, which calls the *ejbLoad()* method. If *ejbLoad()* runs successfully, the client then commits the transaction, causing the *ejbStore()* method to be called.
- In *ejbStore()*, the entity bean gets a connection and writes its state to the database; if the connection retrieved is stale, the transaction is marked *rollbackOnly* and a new *RemoteException* that wraps the *StaleConnectionException* is thrown. That exception is then caught by the client, which cleans up the JDBC resources, rolls back the transaction, and calls *causedByStaleConnection()*, which determines if a *StaleConnectionException* is buried somewhere in the exception.
- If the method returns true, the retry flag is set and the transaction is retried; otherwise, the exception is rethrown to the caller.
- The *causedByStaleConnection()* method looks through the chain of detail attributes to find the original exception. Multiple wrapping of exceptions can occur by the time the exception finally gets back to the client, so the method keeps searching until it encounters a *non-RemoteException*. If this final exception is a *StaleConnectionException*, you found it and true is returned; otherwise, there is no *StaleConnectionException* in the list (because *StaleConnectionException* can never be cast to a *RemoteException*), and false is returned.
- If you are talking to a CMP bean instead of to a BMP bean, the session bean is exactly the same. The CMP bean's *ejbStore()* method would most likely be empty, and the container after calling it would persist the bean with generated code.
- If a stale connection exception occurs during persistence, it is wrapped with a *RemoteException* and returned to the caller. The *causedByStaleConnection()* method would again look through the exception chain and find the root exception, which would be *StaleConnectionException*.

Objects operating in a local transaction context.

When a database operation occurs outside of a global transaction context, a local transaction is implicitly begun by the container. This includes servlets or JSPs which do not begin transactions with the *UserTransaction* interface, as well as enterprise beans running in unspecified transaction contexts. As with global transactions, you must roll back the local transaction before the operation is retried. In these cases, the local transaction containment

usually ends when the business method ends. The one exception is if you are using activity sessions. In this case the activity session must end before attempting to get a new connection.

When the local transaction occurs in an enterprise bean running in an unspecified transaction context, the enterprise bean client object, outside of the local transaction containment, could use the method described in the previous bullet to retry the transaction. However, when the local transaction containment takes place as part of a servlet or JSP file, there is no client object available to retry the operation. For this reason, it is recommended to avoid database operations in servlets and JSP files unless they are a part of a user transaction.

Example: Developing servlet with user transaction:

```
//==================START_PROLOG=========================================
//
//   5630-A23, 5630-A22,
//   (C) COPYRIGHT International Business Machines Corp. 2002
//   All Rights Reserved
//   Licensed Materials - Property of IBM
//   US Government Users Restricted Rights - Use, duplication or
//   disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
//
//   IBM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING
//   ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
//   PURPOSE. IN NO EVENT SHALL IBM BE LIABLE FOR ANY SPECIAL, INDIRECT OR
//   CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF
//   USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR
//   OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE
//   OR PERFORMANCE OF THIS SOFTWARE.
//
//==================END_PROLOG===========================================

package WebSphereSamples.ConnPool;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
// Import JDBC packages and naming service packages.  Note the lack
// of an IBM Extensions package import.  This is no longer required.
import java.sql.*;
import javax.sql.*;
import javax.naming.*;
import javax.transaction.*;

public class EmployeeListTran extends HttpServlet {
    private static DataSource ds   = null;
    private UserTransaction ut = null;
    private static String title    = "Employee List";

// ****************************************************************
// * Initialize servlet when it is first loaded.                  *
// * Get information from the properties file, and look up the    *
// * DataSource object from JNDI to improve performance of the    *
// * the servlet's service methods.                               *
// ****************************************************************
    public void init(ServletConfig config)
    throws ServletException
    {
        super.init(config);
        getDS();
    }

// ****************************************************************
```

```
// * Perform the JNDI lookup for the DataSource and              *
// * User Transaction objects.                                   *
// * This method is invoked from init(), and from the service    *
// * method of the DataSource is null                            *
// *************************************************************
    private void getDS() {
        try {
            Hashtable parms = new Hashtable();
            parms.put(Context.INITIAL_CONTEXT_FACTORY,
 "com.ibm.websphere.naming.WsnInitialContextFactory");
            InitialContext ctx = new InitialContext(parms);
            // Perform a naming service lookup to get the DataSource object.
            ds = (DataSource)ctx.lookup("java:comp/env/jdbc/SampleDB");
            ut = (UserTransaction) ctx.lookup("java:comp/UserTransaction");
        } catch (Exception e) {
            System.out.println("Naming service exception: " + e.getMessage());
            e.printStackTrace();
        }
    }

// *************************************************************
// * Respond to user GET request                                 *
// *************************************************************
    public void doGet(HttpServletRequest req, HttpServletResponse res)
  throws ServletException, IOException
```

Note: The previous line of code should be entered on one line; it is split here for formatting purposes.

```
  {
        Connection conn  = null;
        Statement stmt = null;
        ResultSet rs = null;
        Vector employeeList = new Vector();
// Set retryCount to the number of times you would like to retry after a
// StaleConnectionException
        int retryCount = 5;
        // If the Database code processes successfully, we will set error = false
        boolean error = true;
        do {
            try {
                //Start a new Transaction
                ut.begin();
                // Get a Connection object conn using the DataSource factory.
        conn = ds.getConnection();
                // Run DB query using standard JDBC coding.
                stmt = conn.createStatement();
                String query   = "Select FirstNme, MidInit, LastName " +
                                 "from Employee ORDER BY LastName";
                rs   = stmt.executeQuery(query);
                while (rs.next()) {
                    employeeList.addElement(rs.getString(3) + ", " + rs.getString(1) +
         " " + rs.getString(2));
```

Note: The previous line of code (beginning `employeeList.addElement`) should be entered on one line; it is split here for formatting purposes.

```
          }
            //Set error to false to indicate successful completion of the database work
                error=false;
             } catch (com.ibm.ejs.cm.pool.ConnectionWaitTimeoutException cw) {

               // This exception is thrown if a connection can not be obtained from the
               // pool within a configurable amount of time.  Frequent occurrences of
```

```
            // this exception indicate an incorrectly tuned connection pool

  System.out.println("Connection Wait Timeout Exception during get connection or process SQL: "
  + c.getMessage());
```

Note: The previous line of code should be entered on one line; it is split here for formatting purposes.

```
//In general, we do not want to retry after this exception, so set retry count to 0
//and rollback the transaction
          try {
             ut.setRollbackOnly();
              }
          catch (SecurityException se) {

    //Thrown to indicate that the thread is not allowed to roll back the transaction.
          System.out.println("Security Exception setting rollback only! " + se.getMessage());
                }
      catch (IllegalStateException ise) {
    //Thrown if the current thread is not associated with a transaction.
          System.out.println("Illegal State Exception setting rollback only! "
     + ise.getMessage());
```

Note: The previous line of code should be entered on one line; it is split here for formatting purposes.

```
           }
          catch (SystemException sye) {
          //Thrown if the transaction manager encounters an unexpected error condition
          System.out.println("System Exception setting rollback only! " + sye.getMessage());
                }
          retryCount=0;
            }
            catch (com.ibm.websphere.ce.cm.StaleConnectionException sc) {

   // This exception indicates that the connection to the database is no longer valid.
   //Rollback the transaction, then retry several times to attempt to obtain a valid
 //connection, display an error message if the connection still can not be obtained.

          System.out.println("Stale Connection Exception during get connection or process SQL: "
     + sc.getMessage());
```

Note: The previous line of code should be entered on one line; it is split here for formatting purposes.

```
try {
                   ut.setRollbackOnly();
                }
                catch (SecurityException se) {

     //Thrown to indicate that the thread is not allowed to roll back the transaction.
          System.out.println("Security Exception setting rollback only! " + se.getMessage());
                }
                catch (IllegalStateException ise) {

  //Thrown if the current thread is not associated with a transaction.
          System.out.println("Illegal State Exception setting rollback only! "
     + ise.getMessage());
```

Note: The previous line of code should be entered on one line; it is split here for formatting purposes.

```
 }
                catch (SystemException sye) {
    //Thrown if the transaction manager encounters an unexpected error condition
          System.out.println("System Exception setting rollback only! " + sye.getMessage());
                 }
```

```
                if (--retryCount == 0) {
                   System.out.println("Five stale connection exceptions, displaying error page.");
                }
            }
            catch (SQLException sq) {
             System.out.println("SQL Exception during get connection or process SQL: "
    + sq.getMessage());
```

Note: The previous line of code should be entered on one line; it is split here for formatting purposes.

```
//In general, we do not want to retry after this exception, so set retry count to 0
//and rollback the transaction
                try {
                   ut.setRollbackOnly();
                }
                catch (SecurityException se) {

     //Thrown to indicate that the thread is not allowed to roll back the transaction.
     System.out.println("Security Exception setting rollback only! " + se.getMessage());
                 }
                catch (IllegalStateException ise) {
     //Thrown if the current thread is not associated with a transaction.
     System.out.println("Illegal State Exception setting rollback only! "+ ise.getMessage());
                 }
                 catch (SystemException sye) {
     //Thrown if the transaction manager encounters an unexpected error condition
     System.out.println("System Exception setting rollback only! " + sye.getMessage());
                 }
                 retryCount=0;
            }
            catch (NotSupportedException nse) {

  //Thrown by UserTransaction begin method if the thread is already associated with a
  //transaction and the Transaction Manager implementation does not support nested transactions.
     System.out.println("NotSupportedException on User Transaction begin: " + nse.getMessage());
       }
            catch (SystemException se) {

   //Thrown if the transaction manager encounters an unexpected error condition
    System.out.println("SystemException in User Transaction: " + se.getMessage());
            }
            catch (Exception e) {
    System.out.println("Exception in get connection or process SQL: " + e.getMessage());
//In general, we do not want to retry after this exception, so set retry count to 5
//and rollback the transaction
                try {
                   ut.setRollbackOnly();
                }
                catch (SecurityException se) {
  //Thrown to indicate that the thread is not allowed to roll back the transaction.
     System.out.println("Security Exception setting rollback only! " + se.getMessage());
}
                catch (IllegalStateException ise) {
  //Thrown if the current thread is not associated with a transaction.
     System.out.println("Illegal State Exception setting rollback only! " + ise.getMessage());
                 }
                 catch (SystemException sye) {
  //Thrown if the transaction manager encounters an unexpected error condition
     System.out.println("System Exception setting rollback only! " +  sye.getMessage());
                 }
                 retryCount=0;
            }
             finally {

                // Always close the connection in a finally statement to ensure proper
                // closure in all cases. Closing the connection does not close and
```

```
                // actual connection, but releases it back to the pool for reuse.

                if (rs != null) {
                   try {
                      rs.close();
                   }
                   catch (Exception e) {
                      System.out.println("Close Resultset Exception: " +  e.getMessage());
                   }
                }
                if (stmt != null) {
                   try {
  stmt.close();
                   }
                   catch (Exception e) {
                   System.out.println("Close Statement Exception: " + e.getMessage());
                   }
                }
                if (conn != null) {
                   try {
                      conn.close();
                   }
                   catch (Exception e) {
                      System.out.println("Close connection exception: " + e.getMessage());
                    }
                }
                try {
                    ut.commit();
                }
                catch (RollbackException re) {

 //Thrown to indicate that the transaction has been rolled back rather than committed.
                   System.out.println("User Transaction Rolled back! " + re.getMessage());
     }
                catch (SecurityException se) {
 //Thrown to indicate that the thread is not allowed to commit the transaction.
                System.out.println("Security Exception thrown on transaction commit: "
      + se.getMessage());
```

Note: The previous line of code should be entered on one line; it is split here for formatting purposes.

```
}
                catch (IllegalStateException ise) {
 //Thrown if the current thread is not associated with a transaction.
 System.out.println("Illegal State Exception thrown on transaction commit: " + ise.getMessage());
                }
                catch (SystemException sye) {
 //Thrown if the transaction manager encounters an unexpected error condition
 System.out.println("System Exception thrown on transaction commit: " + sye.getMessage());
                }
                catch (Exception e) {
 System.out.println("Exception thrown on transaction commit: " + e.getMessage());
                }
            }
       } while ( error==true && retryCount > 0 );

       // Prepare and return HTML response, prevent dynamic content from being cached
       // on browsers.
       res.setContentType("text/html");
       res.setHeader("Pragma", "no-cache");
       res.setHeader("Cache-Control", "no-cache");
       res.setDateHeader("Expires", 0);
       try {
           ServletOutputStream out = res.getOutputStream();
           out.println("<HTML>");
           out.println("<HEAD><TITLE>" + title + "</TITLE></HEAD>");
```

```
            out.println("<BODY>");
            if (error==true) {
                out.println("<H1>There was an error processing this request.</H1>
       Please try the request again, or contact " +
       " the <a href='mailto:sysadmin@my.com'>System Administrator</a>");
```

Note: The previous `if` statement should be entered on one line; it is split here for formatting purposes.

```
            }
            else if (employeeList.isEmpty()) {
               out.println("<H1>Employee List is Empty</H1>");
            }
            else {
               out.println("<H1>Employee List </H1>");
               for (int i = 0; i < employeeList.size(); i++) {
                  out.println(employeeList.elementAt(i) + "<BR>");
               }
            }
            out.println("</BODY></HTML>");
            out.close();
        }
        catch (IOException e) {
           System.out.println("HTML response exception: " + e.getMessage());
        }
    }
}
```

Example: Developing session bean with container managed transaction:

```
//==================START_PROLOG=========================================
//
//   5630-A23, 5630-A22,
//   (C) COPYRIGHT International Business Machines Corp. 2002
//   All Rights Reserved
//   Licensed Materials - Property of IBM
//   US Government Users Restricted Rights - Use, duplication or
//   disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
//
//   IBM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING
//   ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
//   PURPOSE. IN NO EVENT SHALL IBM BE LIABLE FOR ANY SPECIAL, INDIRECT OR
//   CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF
//   USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR
//   OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE
//   OR PERFORMANCE OF THIS SOFTWARE.
//
//==================END_PROLOG===========================================

package WebSphereSamples.ConnPool;

import java.rmi.RemoteException;
import java.util.*;
import java.sql.*;
import javax.sql.*;
import javax.ejb.*;
import javax.naming.*;

/*****************************************************************************
* This bean is designed to demonstrate Database Connections in a
* Container Managed Transaction Session Bean.  Its transaction attribute              *
* should be set to TX_REQUIRED or TX_REQUIRES_NEW.                                    *
*****************************************************************************
*/
public class ShowEmployeesCMTBean implements SessionBean {
 private javax.ejb.SessionContext mySessionCtx = null;
 final static long serialVersionUID = 3206093459760846163L;
```

```
 private javax.sql.DataSource ds;

//*****************************************************************************
//* ejbActivate calls the getDS method, which does the JNDI lookup for the DataSource.
//* Because the DataSource lookup is in a separate method, we can also invoke it from
//* the getEmployees method in the case where the DataSource field is null.
//*****************************************************************************
public void ejbActivate() throws java.rmi.RemoteException {
 getDS();
}
/**
 * ejbCreate method
 * @exception javax.ejb.CreateException
 * @exception java.rmi.RemoteException
 */
public void ejbCreate() throws javax.ejb.CreateException, java.rmi.RemoteException {}
/**
 * ejbPassivate method
 * @exception java.rmi.RemoteException
 */
public void ejbPassivate() throws java.rmi.RemoteException {}
/**
 * ejbRemove method
 * @exception java.rmi.RemoteException
 */
public void ejbRemove() throws java.rmi.RemoteException {}

//*****************************************************************************
//* The getEmployees method runs the database query to retrieve the employees.
//* The getDS method is only called if the DataSource variable is null.
//* Because this session bean uses Container Managed Transactions, it cannot retry the
//* transaction on a StaleConnectionException.  However, it can throw an exception to
//* its client indicating that the operation is retriable.    *
//*****************************************************************************

public Vector getEmployees() throws com.ibm.ejs.cm.pool.ConnectionWaitTimeoutException,
SQLException, RetryableConnectionException {
```

Note: The previous line of code should be entered on one line; here it is split for formatting purposes.

```
Connection conn  = null;
   Statement stmt = null;
   ResultSet rs = null;
   Vector employeeList = new Vector();

   if (ds == null) getDS();

   try {
      // Get a Connection object conn using the DataSource factory.
      conn = ds.getConnection();
      // Run DB query using standard JDBC coding.
      stmt = conn.createStatement();
      String query   = "Select FirstNme, MidInit, LastName " +
                       "from Employee ORDER BY LastName";
      rs   = stmt.executeQuery(query);
      while (rs.next()) {
         employeeList.addElement(rs.getString(3) + ", " + rs.getString(1) + " "
    + rs.getString(2));
```

Note: The previous line of code should be entered on one line; here it is split for formatting purposes.

```
      }
   }
catch (com.ibm.websphere.ce.cm.StaleConnectionException se) {
```

```
// This exception indicates that the connection to the database is no longer valid.
// Rollback the transaction, and throw an exception to the client indicating they
// can retry the transaction if desired.

System.out.println("Stale Connection Exception during get connection or process SQL: "
+ se.getMessage());
```

Note: The previous line of code should be entered on one line; here it is split for formatting purposes.

```
System.out.println("Rolling back transaction and throwing  RetryableConnectionException");

      mySessionCtx.setRollbackOnly();
      throw new RetryableConnectionException(se.toString());
      }
      catch (com.ibm.ejs.cm.pool.ConnectionWaitTimeoutException cw) {

// This exception is thrown if a connection can not be obtained from the
// pool within a configurable amount of time.  Frequent occurrences of
// this exception indicate an incorrectly tuned connection pool

System.out.println("Connection Wait Timeout Exception during get connection or process SQL:
  " + cw.getMessage());
```

Note: The previous line of code should be entered on one line; here it is split for formatting purposes.

```
throw cw;
      }
      catch (SQLException sq) {

//Throwing a remote exception will automatically roll back the container managed //transaction

      System.out.println("SQL Exception during get connection or process SQL: " +
    sq.getMessage());
```

Note: The previous line of code should be entered on one line; here it is split for formatting purposes.

```
throw sq;
      }
      finally {

         // Always close the connection in a finally statement to ensure proper
         // closure in all cases. Closing the connection does not close and
         // actual connection, but releases it back to the pool for reuse.

         if (rs != null) {
            try {
    rs.close();
            }
 catch (Exception e) {
               System.out.println("Close Resultset Exception: " + e.getMessage());
            }
         }
         if (stmt != null) {
            try {
               stmt.close();
            }
            catch (Exception e) {
               System.out.println("Close Statement Exception: " + e.getMessage());
            }
         }
         if (conn != null) {
            try {
               conn.close();
            }
```

```
            catch (Exception e) {
               System.out.println("Close connection exception: " + e.getMessage());
            }
         }
      }
      return employeeList;
}
/**
 * getSessionContext method
 * @return javax.ejb.SessionContext
 */
public javax.ejb.SessionContext getSessionContext() {
 return mySessionCtx;
}
//*****************************************************************************
//* The getDS method performs the JNDI lookup for the DataSource.        *
//* This method is called from ejbActivate, and from getEmployees if the DataSource
//* object is null.                          *
//*****************************************************************************

private void getDS() {
 try {
  Hashtable parms = new Hashtable();
  parms.put(Context.INITIAL_CONTEXT_FACTORY,
    "com.ibm.websphere.naming.WsnInitialContextFactory");
  InitialContext ctx = new InitialContext(parms);
  // Perform a naming service lookup to get the DataSource object.
  ds = (DataSource)ctx.lookup("java:comp/env/jdbc/SampleDB");
 }
       catch (Exception e) {
  System.out.println("Naming service exception: " + e.getMessage());
  e.printStackTrace();
 }
}
/**
 * setSessionContext method
 * @param ctx javax.ejb.SessionContext
 * @exception java.rmi.RemoteException
 */
public void setSessionContext(javax.ejb.SessionContext ctx) throws java.rmi.RemoteException {
 mySessionCtx = ctx;
}
}
//==================START_PROLOG=========================================
//
//   5630-A23, 5630-A22,
//   (C) COPYRIGHT International Business Machines Corp. 2002
//   All Rights Reserved
//   Licensed Materials - Property of IBM
//   US Government Users Restricted Rights - Use, duplication or
//   disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
//
//   IBM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING
//   ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
//   PURPOSE. IN NO EVENT SHALL IBM BE LIABLE FOR ANY SPECIAL, INDIRECT OR
//   CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF
//   USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR
//   OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE
//   OR PERFORMANCE OF THIS SOFTWARE.
//
//==================END_PROLOG===========================================

package WebSphereSamples.ConnPool;

/**
 * This is a Home interface for the Session Bean
 */
```

```
public interface ShowEmployeesCMTHome extends javax.ejb.EJBHome {

/**
 * create method for a session bean
* @return WebSphereSamples.ConnPool.ShowEmployeesCMT
 * @exception javax.ejb.CreateException
 * @exception java.rmi.RemoteException
 */
WebSphereSamples.ConnPool.ShowEmployeesCMT create() throws javax.ejb.CreateException,
 java.rmi.RemoteException;
```

Note: The previous line of code should be entered on one line; here it is split for formatting purposes.

```
}
//==================START_PROLOG=======================================
//
//   5630-A23, 5630-A22,
//   (C) COPYRIGHT International Business Machines Corp. 2002
//   All Rights Reserved
//   Licensed Materials - Property of IBM
//   US Government Users Restricted Rights - Use, duplication or
//   disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
//
//   IBM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING
//   ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
//   PURPOSE. IN NO EVENT SHALL IBM BE LIABLE FOR ANY SPECIAL, INDIRECT OR
//   CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF
//   USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR
//   OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE
//   OR PERFORMANCE OF THIS SOFTWARE.
//
//==================END_PROLOG=========================================

package WebSphereSamples.ConnPool;

/**
 * This is an Enterprise Java Bean Remote Interface
 */
public interface ShowEmployeesCMT extends javax.ejb.EJBObject {

/**
 *
 * @return java.util.Vector
 */
java.util.Vector getEmployees() throws java.sql.SQLException, java.rmi.RemoteException,
 com.ibm.ejs.cm.pool.ConnectionWaitTimeoutException,
 WebSphereSamples.ConnPool.RetryableConnectionException;
```

Note: The previous line of code (beginning `java.util.Vector`) should be entered on one line; here it is split for formatting purposes.

```
}
//==================START_PROLOG=======================================
//
//   5630-A23, 5630-A22,
//   (C) COPYRIGHT International Business Machines Corp. 2002
//   All Rights Reserved
//   Licensed Materials - Property of IBM
//   US Government Users Restricted Rights - Use, duplication or
//   disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
//
//   IBM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING
//   ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
//   PURPOSE. IN NO EVENT SHALL IBM BE LIABLE FOR ANY SPECIAL, INDIRECT OR
//   CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF
```

```
//   USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR
//   OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE
//   OR PERFORMANCE OF THIS SOFTWARE.
//
//==================END_PROLOG=========================================

package WebSphereSamples.ConnPool;

/**
 * Exception indicating that the operation can be retried
 * Creation date: (4/2/2001 10:48:08 AM)
 * @author: Administrator
 */
public class RetryableConnectionException extends Exception {
/**
 * RetryableConnectionException constructor.
 */
public RetryableConnectionException() {
 super();
}
/**
 * RetryableConnectionException constructor.
 * @param s java.lang.String
 */
public RetryableConnectionException(String s) {
 super(s);
}
}
```

Example: Developing session bean with bean managed transaction:

```
//==================START_PROLOG=========================================
//
//   5630-A23, 5630-A22,
//   (C) COPYRIGHT International Business Machines Corp. 2002
//   All Rights Reserved
//   Licensed Materials - Property of IBM
//   US Government Users Restricted Rights - Use, duplication or
//   disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
//
//   IBM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING
//   ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
//   PURPOSE. IN NO EVENT SHALL IBM BE LIABLE FOR ANY SPECIAL, INDIRECT OR
//   CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF
//   USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR
//   OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE
//   OR PERFORMANCE OF THIS SOFTWARE.
//
//==================END_PROLOG=========================================

package WebSphereSamples.ConnPool;

import java.rmi.RemoteException;
import java.util.*;
import java.sql.*;
import javax.sql.*;
import javax.ejb.*;
import javax.naming.*;
import javax.transaction.*;

/*******************************************************************************
* This bean is designed to demonstrate Database Connections in a
* Bean-Managed Transaction Session Bean.  Its transaction attribute
* should be set to TX_BEANMANAGED.
*******************************************************************************/
public class ShowEmployeesBMTBean implements SessionBean {
 private javax.ejb.SessionContext mySessionCtx = null;
```

```
 final static long serialVersionUID = 3206093459760846163L;

 private javax.sql.DataSource ds;

 private javax.transaction.UserTransaction userTran;

//*******************************************************************************
//* ejbActivate calls the getDS method, which makes the JNDI lookup for the DataSource
//* Because the DataSource lookup is in a separate method, we can also invoke it from
//* the getEmployees method in the case where the DataSource field is null.
//*******************************************************************************
public void ejbActivate() throws java.rmi.RemoteException {
 getDS();
}
/**
 * ejbCreate method
 * @exception javax.ejb.CreateException
 * @exception java.rmi.RemoteException
 */
public void ejbCreate() throws javax.ejb.CreateException, java.rmi.RemoteException {}
/**
 * ejbPassivate method
 * @exception java.rmi.RemoteException
 */
public void ejbPassivate() throws java.rmi.RemoteException {}
/**
 * ejbRemove method
 * @exception java.rmi.RemoteException
 */
public void ejbRemove() throws java.rmi.RemoteException {}

//*******************************************************************************
//* The getEmployees method runs the database query to retrieve the employees.
//* The getDS method is only called if the DataSource or userTran variables are null.
//* If a StaleConnectionException occurs, the bean retries the transaction 5 times,
//* then throws an EJBException.       *
//*******************************************************************************

public Vector getEmployees() throws EJBException {
   Connection conn  = null;
   Statement stmt = null;
   ResultSet rs = null;
   Vector employeeList = new Vector();

   // Set retryCount to the number of times you would like to retry after a
   //StaleConnectionException
   int retryCount = 5;

   // If the Database code processes successfully, we will set error = false
   boolean error = true;

   if (ds == null || userTran == null) getDS();
      do {
         try {
              //try/catch block for UserTransaction work
              //Begin the transaction
            userTran.begin();
try {
                 //try/catch block for database work
                 //Get a Connection object conn using the DataSource factory.
               conn = ds.getConnection();
               // Run DB query using standard JDBC coding.
               stmt = conn.createStatement();
               String query   = "Select FirstNme, MidInit, LastName " +
        "from Employee ORDER BY LastName";
               rs   = stmt.executeQuery(query);
               while (rs.next()) {
```

```
                employeeList.addElement(rs.getString(3) + ", " +
             }
//Set error to false, as all database operations are successfully completed
             error = false;
          }
          catch (com.ibm.websphere.ce.cm.StaleConnectionException se) {

// This exception indicates that the connection to the database is no longer valid.
// Rollback the transaction, and throw an exception to the client indicating they
// can retry the transaction if desired.

System.out.println("Stale Connection Exception during get connection or process SQL: "
 + se.getMessage());
```

Note: The previous line of code must be entered on one line; here it is split for formatting purposes.

```
     userTran.rollback();
          if (--retryCount == 0) {
//If we have already retried the requested number of times, throw an EJBException.
             throw new EJBException("Transaction Failure: " + se.toString());
          }
          else {
             System.out.println("Retrying transaction, retryCount = " + retryCount);
          }
       }
       catch (com.ibm.ejs.cm.pool.ConnectionWaitTimeoutException cw) {

// This exception is thrown if a connection can not be obtained from the
// pool within a configurable amount of time.  Frequent occurrences of
// this exception indicate an incorrectly tuned connection pool

          System.out.println("Connection Wait Timeout Exception during get connection
      or process SQL: " + cw.getMessage());
```

Note: The previous line of code must be entered on one line; here it is split for formatting purposes.

```
     userTran.rollback();
          throw new EJBException("Transaction failure: " + cw.getMessage());
       }
       catch (SQLException sq) {
   // This catch handles all other SQL Exceptions
   System.out.println("SQL Exception during get connection or process SQL: " +
    sq.getMessage());
          userTran.rollback();
          throw new EJBException("Transaction failure: " + sq.getMessage());
       }
       finally {
// Always close the connection in a finally statement to ensure proper
// closure in all cases. Closing the connection does not close and
// actual connection, but releases it back to the pool for reuse.

          if (rs != null) {
             try {
                rs.close();
             }
             catch (Exception e) {
                System.out.println("Close Resultset Exception: " + e.getMessage());
             }
          }
          if (stmt != null) {
             try {
                stmt.close();
 }
          catch (Exception e) {
             System.out.println("Close Statement Exception: " + e.getMessage());
```

```
            }
         }
         if (conn != null) {
            try {
               conn.close();
            }
            catch (Exception e) {
               System.out.println("Close connection exception: " + e.getMessage());
            }
         }
      }
      if (!error) {
         //Database work completed successfully, commit the transaction
         userTran.commit();
      }
      //Catch UserTransaction exceptions
      }
      catch (NotSupportedException nse) {

//Thrown by UserTransaction begin method if the thread is already associated with a
//transaction and the Transaction Manager implementation does not support nested transactions.
 System.out.println("NotSupportedException on User Transaction begin: " + nse.getMessage());
         throw new EJBException("Transaction failure: " + nse.getMessage());
      }
      catch (RollbackException re) {
//Thrown to indicate that the transaction has been rolled back rather than committed.
         System.out.println("User Transaction Rolled back! " + re.getMessage());
         throw new EJBException("Transaction failure: " + re.getMessage());
      }
      catch (SystemException se) {
   //Thrown if the transaction manager encounters an unexpected error condition
         System.out.println("SystemException in User Transaction: "+ se.getMessage());
         throw new EJBException("Transaction failure: " + se.getMessage());
      }
      catch (Exception e) {
         //Handle any generic or unexpected Exceptions
         System.out.println("Exception in User Transaction: " + e.getMessage());
         throw new EJBException("Transaction failure: " + e.getMessage());
      }
   }
   while (error);
   return employeeList;
}
/**
 * getSessionContext method comment
 * @return javax.ejb.SessionContext
 */
public javax.ejb.SessionContext getSessionContext() {
 return mySessionCtx;
}

//*****************************************************************************
//* The getDS method performs the JNDI lookup for the DataSource.
//* This method is called from ejbActivate, and from getEmployees if the DataSource
//* object is null.
//*****************************************************************************
private void getDS() {
   try {
      Hashtable parms = new Hashtable();
parms.put(Context.INITIAL_CONTEXT_FACTORY,
         "com.ibm.websphere.naming.WsnInitialContextFactory");
      InitialContext ctx = new InitialContext(parms);

      // Perform a naming service lookup to get the DataSource object.
      ds = (DataSource)ctx.lookup("java:comp/env/jdbc/SampleDB");
      //Create the UserTransaction object
      userTran = mySessionCtx.getUserTransaction();
```

```
   }
   catch (Exception e) {
      System.out.println("Naming service exception: " + e.getMessage());
      e.printStackTrace();
}
}
/**
 * setSessionContext method
 * @param ctx javax.ejb.SessionContext
 * @exception java.rmi.RemoteException
 */
public void setSessionContext(javax.ejb.SessionContext ctx) throws java.rmi.RemoteException {
 mySessionCtx = ctx;
}
}
//==================START_PROLOG=======================================
//
//   5630-A23, 5630-A22,
//   (C) COPYRIGHT International Business Machines Corp. 2002
//   All Rights Reserved
//   Licensed Materials - Property of IBM
//   US Government Users Restricted Rights - Use, duplication or
//   disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
//
//   IBM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING
//   ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
//   PURPOSE. IN NO EVENT SHALL IBM BE LIABLE FOR ANY SPECIAL, INDIRECT OR
//   CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF
//   USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR
//   OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE
//   OR PERFORMANCE OF THIS SOFTWARE.
//
//==================END_PROLOG=========================================

package WebSphereSamples.ConnPool;

/**
 * This is a Home interface for the Session Bean
 */
public interface ShowEmployeesBMTHome extends javax.ejb.EJBHome {

/**
 * create method for a session bean
 * @return WebSphereSamples.ConnPool.ShowEmployeesBMT
 * @exception javax.ejb.CreateException
 * @exception java.rmi.RemoteException
 */
WebSphereSamples.ConnPool.ShowEmployeesBMT create() throws javax.ejb.CreateException,
 java.rmi.RemoteException;
```

Note: The previous line of code must be entered on one line; here it is split for formatting purposes.

```
}
//==================START_PROLOG=======================================
//
//   5630-A23, 5630-A22,
//   (C) COPYRIGHT International Business Machines Corp. 2002
//   All Rights Reserved
//   Licensed Materials - Property of IBM
//   US Government Users Restricted Rights - Use, duplication or
//   disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
//
//   IBM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING
//   ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
//   PURPOSE. IN NO EVENT SHALL IBM BE LIABLE FOR ANY SPECIAL, INDIRECT OR
```

```
//   CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF
//   USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR
//   OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE
//   OR PERFORMANCE OF THIS SOFTWARE.
//
//==================END_PROLOG=========================================

package WebSphereSamples.ConnPool;

/**
 * This is an Enterprise Java Bean Remote Interface
 */
public interface ShowEmployeesBMT extends javax.ejb.EJBObject {

/**
 *
 * @return java.util.Vector
 */
java.util.Vector getEmployees() throws java.rmi.RemoteException, javax.ejb.EJBException;
}
```

Example: Developing entity bean with bean managed persistence (container managed transaction):

```
//==================START_PROLOG=======================================
//
//   5630-A23, 5630-A22,
//   (C) COPYRIGHT International Business Machines Corp. 2002
//   All Rights Reserved
//   Licensed Materials - Property of IBM
//   US Government Users Restricted Rights - Use, duplication or
//   disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
//
//   IBM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING
//   ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
//   PURPOSE. IN NO EVENT SHALL IBM BE LIABLE FOR ANY SPECIAL, INDIRECT OR
//   CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF
//   USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR
//   OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE
//   OR PERFORMANCE OF THIS SOFTWARE.
//
//==================END_PROLOG=========================================

package WebSphereSamples.ConnPool;

import java.rmi.RemoteException;
import java.util.*;
import javax.ejb.*;
import java.sql.*;
import javax.sql.*;
import javax.ejb.*;
import javax.naming.*;

/**
 * This is an Entity Bean class with five BMP fields
 * String firstName, String lastName, String middleInit
 * String empNo, int edLevel
 */
public class EmployeeBMPBean implements EntityBean {
 private javax.ejb.EntityContext entityContext = null;
 final static long serialVersionUID = 3206093459760846163L;

 private java.lang.String firstName;
 private java.lang.String lastName;
 private String middleInit;
 private javax.sql.DataSource ds;
 private java.lang.String empNo;
```

```
 private int edLevel;
/**
 * ejbActivate method
 * @exception java.rmi.RemoteException
 * ejbActivate calls getDS(), which perfoms the
 * JNDI lookup for the datasource.
 */
public void ejbActivate() throws java.rmi.RemoteException {
 getDS();
}
/**
 * ejbCreate method for a BMP entity bean
 * @return WebSphereSamples.ConnPool.EmployeeBMPKey
 * @param key WebSphereSamples.ConnPool.EmployeeBMPKey
 * @exception javax.ejb.CreateException
 * @exception java.rmi.RemoteException
 */
public WebSphereSamples.ConnPool.EmployeeBMPKey ejbCreate(String empNo, String firstName,
 String lastName, String middleInit, int edLevel) throws javax.ejb.CreateException,
 java.rmi.RemoteException {
```

Note: The previous line of code (beginning `public WebSphereSamples`) should be entered on one line; it is split here for formatting purposes.

```
Connection conn = null;
   PreparedStatement ps = null;

   if (ds == null) getDS();

   this.empNo = empNo;
   this.firstName = firstName;
   this.lastName = lastName;
   this.middleInit = middleInit;
   this.edLevel = edLevel;

   String sql = "insert into Employee (empNo, firstnme, midinit, lastname, edlevel)
  values (?,?,?,?,?)";
```

Note: The previous line of code should be entered on one line; it is split here for formatting purposes.

```
try {
      conn = ds.getConnection();
      ps = conn.prepareStatement(sql);
      ps.setString(1, empNo);
      ps.setString(2, firstName);
      ps.setString(3, middleInit);
      ps.setString(4, lastName);
      ps.setInt(5, edLevel);

 if (ps.executeUpdate() != 1){
         System.out.println("ejbCreate Failed to add user.");
         throw new CreateException("Failed to add user.");
      }
   }
   catch (com.ibm.websphere.ce.cm.StaleConnectionException se) {

// This exception indicates that the connection to the database is no longer valid.
// Rollback the transaction, and throw an exception to the client indicating they
// can retry the transaction if desired.

System.out.println("Stale Connection Exception during get connection or process SQL: "
  +  se.getMessage());
```

Note: The previous line of code should be entered on one line; it is split here for formatting purposes.

```
throw new CreateException(se.getMessage());
   }
   catch (SQLException sq) {
      System.out.println("SQL Exception during get connection or process SQL: " +
                         sq.getMessage());
   throw new CreateException(sq.getMessage());
   }
   finally {
      // Always close the connection in a finally statement to ensure proper
      // closure in all cases. Closing the connection does not close and
      // actual connection, but releases it back to the pool for reuse.
  if (ps != null) {
         try {
            ps.close();
         }
         catch (Exception e) {
            System.out.println("Close Statement Exception: " + e.getMessage());
         }
      }
      if (conn != null) {
         try {
            conn.close();
         }
         catch (Exception e) {
            System.out.println("Close connection exception: " + e.getMessage());
         }
      }
   }
   return new EmployeeBMPKey(this.empNo);
}
/**
 * ejbFindByPrimaryKey method
 * @return WebSphereSamples.ConnPool.EmployeeBMPKey
 * @param primaryKey WebSphereSamples.ConnPool.EmployeeBMPKey
 * @exception java.rmi.RemoteException
 * @exception javax.ejb.FinderException
 */
public WebSphereSamples.ConnPool.EmployeeBMPKey ejbFindByPrimaryKey
 (WebSphereSamples.ConnPool.EmployeeBMPKey primaryKey)
 throws java.rmi.RemoteException, javax.ejb.FinderException {
```

Note: The previous line of code (beginning `public WebSphereSamples.ConnPool...`) should be entered on one line; it is split here for formatting purposes.

```
loadByEmpNo(primaryKey.empNo);
 return primaryKey;
}
/**
 * ejbLoad method
 * @exception java.rmi.RemoteException
 */
public void ejbLoad() throws java.rmi.RemoteException {
 try {
  EmployeeBMPKey pk = (EmployeeBMPKey) entityContext.getPrimaryKey();
  loadByEmpNo(pk.empNo);
 } catch (FinderException fe) {
  throw new RemoteException("Cannot load Employee state from database.");
 }
}
/**
 * ejbPassivate method
 * @exception java.rmi.RemoteException
 */
public void ejbPassivate() throws java.rmi.RemoteException {}
/**
 * ejbPostCreate method for a BMP entity bean
 * @param key WebSphereSamples.ConnPool.EmployeeBMPKey
```

```
 * @exception java.rmi.RemoteException
 */
public void ejbPostCreate(String empNo, String firstName, String lastName, String middleInit,
 int edLevel) throws java.rmi.RemoteException {}
```

Note: The previous line of code should be entered on one line; it is split here for formatting purposes.

```
/**
 * ejbRemove method
 * @exception java.rmi.RemoteException
 * @exception javax.ejb.RemoveException
 */
public void ejbRemove() throws java.rmi.RemoteException, javax.ejb.RemoveException {

   if (ds == null)
      GetDS();

   String sql = "delete from Employee where empNo=?";
   Connection con = null;
   PreparedStatement ps = null;
 try {
      con = ds.getConnection();
      ps = con.prepareStatement(sql);
      ps.setString(1, empNo);
      if (ps.executeUpdate() != 1){
         throw new RemoteException("Cannot remove employee: " + empNo);
      }
   }
   catch (com.ibm.websphere.ce.cm.StaleConnectionException se) {

// This exception indicates that the connection to the database is no longer valid.
// Rollback the transaction, and throw an exception to the client indicating they
// can retry the transaction if desired.

System.out.println("Stale Connection Exception during get connection or process SQL: "
 + se.getMessage());
```

Note: The previous line of code should be entered on one line; it is split here for formatting purposes.

```
 throw new RemoteException(se.getMessage());
   }
   catch (SQLException sq) {
      System.out.println("SQL Exception during get connection or process SQL: " +
                         sq.getMessage());
      throw new RemoteException(sq.getMessage());
   }
   finally {
      // Always close the connection in a finally statement to ensure proper
      // closure in all cases. Closing the connection does not close and
      // actual connection, but releases it back to the pool for reuse.
      if (ps != null) {
         try {
            ps.close();
         }
         catch (Exception e) {
            System.out.println("Close Statement Exception: " + e.getMessage());
         }
      }
      if (con != null) {
         try {
            con.close();
         }
         catch (Exception e) {
            System.out.println("Close connection exception: " + e.getMessage());
         }
```

```
      }
   }
}
 try {
      con = ds.getConnection();
      ps = con.prepareStatement(sql);
      ps.setString(1, empNo);
      if (ps.executeUpdate() != 1){
         throw new RemoteException("Cannot remove employee: " + empNo);
      }
   }
   catch (com.ibm.websphere.ce.cm.StaleConnectionException se) {

// This exception indicates that the connection to the database is no longer valid.
// Rollback the transaction, and throw an exception to the client indicating they
// can retry the transaction if desired.

System.out.println("Stale Connection Exception during get connection or process SQL: "
 + se.getMessage());
```

Note: The previous line of code should be entered on one line; it is split here for formatting purposes.

```
 throw new RemoteException(se.getMessage());
   }
   catch (SQLException sq) {
      System.out.println("SQL Exception during get connection or process SQL: " +
                         sq.getMessage());
      throw new RemoteException(sq.getMessage());
   }
   finally {
      // Always close the connection in a finally statement to ensure proper
      // closure in all cases. Closing the connection does not close and
      // actual connection, but releases it back to the pool for reuse.
      if (ps != null) {
         try {
            ps.close();
         }
         catch (Exception e) {
            System.out.println("Close Statement Exception: " + e.getMessage());
         }
      }
      if (con != null) {
         try {
            con.close();
         }
         catch (Exception e) {
            System.out.println("Close connection exception: " + e.getMessage());
         }
      }
   }
}
}
   catch (com.ibm.websphere.ce.cm.StaleConnectionException se) {

// This exception indicates that the connection to the database is no longer valid.
// Rollback the transaction, and throw an exception to the client indicating they
// can retry the transaction if desired.

System.out.println("Stale Connection Exception during get connection or process SQL: "
 + se.getMessage());
```

Note: The previous line of code should be entered on one line; it is split here for formatting purposes.

```
 throw new RemoteException(se.getMessage());
   }
   catch (SQLException sq) {

      System.out.println("SQL Exception during get connection or process SQL: " +
                         sq.getMessage());

      throw new RemoteException(sq.getMessage());
   }
   finally {
      // Always close the connection in a finally statement to ensure proper
      // closure in all cases. Closing the connection does not close and
      // actual connection, but releases it back to the pool for reuse.
      if (ps != null) {
         try {
            ps.close();
         }
         catch (Exception e) {
            System.out.println("Close Statement Exception: " + e.getMessage());
         }
      }
      if (con != null) {
         try {
            con.close();
         }
         catch (Exception e) {
            System.out.println("Close connection exception: " + e.getMessage());
         }
      }
   }
}
/**
 * Get the employee's edLevel
 * Creation date: (4/20/2001 3:46:22 PM)
 * @return int
 */
public int getEdLevel() {
 return edLevel;
}
/**
 * getEntityContext method
 * @return javax.ejb.EntityContext
 */
public javax.ejb.EntityContext getEntityContext() {
 return entityContext;
}
/**
 * Get the employee's first name
 * Creation date: (4/19/2001 1:34:47 PM)
 * @return java.lang.String
 */
public java.lang.String getFirstName() {
 return firstName;
}
/**
 * Get the employee's last name
 * Creation date: (4/19/2001 1:35:41 PM)
 * @return java.lang.String
 */
public java.lang.String getLastName() {
 return lastName;
}
/**
* get the employee's middle initial
 * Creation date: (4/19/2001 1:36:15 PM)
 * @return char
 */
```

```
public String getMiddleInit() {
 return middleInit;
}
/**
 * Lookup the DataSource from JNDI
 * Creation date: (4/19/2001 3:28:15 PM)
 */
private void getDS() {
 try {
  Hashtable parms = new Hashtable();
  parms.put(Context.INITIAL_CONTEXT_FACTORY,
    "com.ibm.websphere.naming.WsnInitialContextFactory");
  InitialContext ctx = new InitialContext(parms);
  // Perform a naming service lookup to get the DataSource object.
  ds = (DataSource)ctx.lookup("java:comp/env/jdbc/SampleDB");
 }
       catch (Exception e) {
  System.out.println("Naming service exception: " + e.getMessage());
  e.printStackTrace();
 }
}
/**
 * Load the employee from the database
 * Creation date: (4/19/2001 3:44:07 PM)
 * @param empNo java.lang.String
 */
private void loadByEmpNo(String empNoKey) throws javax.ejb.FinderException{

 String sql = "select empno, firstnme, midinit, lastname, edLevel from employee where empno = ?";
 Connection conn = null;
 PreparedStatement ps = null;
 ResultSet rs = null;

 if (ds == null) getDS();

 try {
// Get a Connection object conn using the DataSource factory.
     conn = ds.getConnection();
    // Run DB query using standard JDBC coding.
  ps = conn.prepareStatement(sql);
  ps.setString(1, empNoKey);
  rs = ps.executeQuery();
  if (rs.next()) {
   empNo= rs.getString(1);
   firstName=rs.getString(2);
   middleInit=rs.getString(3);
   lastName=rs.getString(4);
   edLevel=rs.getInt(5);
  }
             else {
                throw new ObjectNotFoundException("Cannot find employee number " +
                                                  empNoKey);
             }
      }
      catch (com.ibm.websphere.ce.cm.StaleConnectionException se) {

// This exception indicates that the connection to the database is no longer valid.
// Rollback the transaction, and throw an exception to the client indicating they
// can retry the transaction if desired.

System.out.println("Stale Connection Exception during get connection or process SQL: "
 + se.getMessage());
```

Note: The previous line of code should be entered on one line; it is split here for formatting purposes.

```
 throw new FinderException(se.getMessage());
      }
      catch (SQLException sq) {
System.out.println("SQL Exception during get connection or process SQL: " +
       sq.getMessage());
         throw new FinderException(sq.getMessage());
      }
      finally {
  // Always close the connection in a finally statement to ensure proper
  // closure in all cases. Closing the connection does not close and
  // actual connection, but releases it back to the pool for reuse.
         if (rs != null) {
            try {
               Rs.close();
            }
            catch (Exception e) {
               System.out.println("Close Resultset Exception: " + e.getMessage());
            }
         }
         if (ps != null) {
            try {
               ps.close();
            }
            catch (Exception e) {
               System.out.println("Close Statement Exception: " + e.getMessage());
            }
         }
         if (conn != null) {
            try {
               conn.close();
            }
            catch (Exception e) {
               System.out.println("Close connection exception: " + e.getMessage());
            }
        }
     }
}
/**
 * set the employee's education level
 * Creation date: (4/20/2001 3:46:22 PM)
 * @param newEdLevel int
 */
public void setEdLevel(int newEdLevel) {
 edLevel = newEdLevel;
}
/**
 * setEntityContext method
 * @param ctx javax.ejb.EntityContext
 * @exception java.rmi.RemoteException
 */
public void setEntityContext(javax.ejb.EntityContext ctx) throws java.rmi.RemoteException {
 entityContext = ctx;
}
/**
 * set the employee's first name
 * Creation date: (4/19/2001 1:34:47 PM)
 * @param newFirstName java.lang.String
 */
public void setFirstName(java.lang.String newFirstName) {
 firstName = newFirstName;
}
/**
 * set the employee's last name
 * Creation date: (4/19/2001 1:35:41 PM)
 * @param newLastName java.lang.String
 */
public void setLastName(java.lang.String newLastName) {
```

```
 lastName = newLastName;
}
/**
 * set the employee's middle initial
 * Creation date: (4/19/2001 1:36:15 PM)
 * @param newMiddleInit char
 */
public void setMiddleInit(String newMiddleInit) {
 middleInit = newMiddleInit;
}
/**
 * unsetEntityContext method
 * @exception java.rmi.RemoteException
 */
public void unsetEntityContext() throws java.rmi.RemoteException {
 entityContext = null;
}
}
//==================START_PROLOG=======================================
//
//   5630-A23, 5630-A22,
//   (C) COPYRIGHT International Business Machines Corp. 2002
//   All Rights Reserved
//   Licensed Materials - Property of IBM
//   US Government Users Restricted Rights - Use, duplication or
//   disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
//
//   IBM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING
//   ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
//   PURPOSE. IN NO EVENT SHALL IBM BE LIABLE FOR ANY SPECIAL, INDIRECT OR
//   CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF
//   USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR
//   OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE
//   OR PERFORMANCE OF THIS SOFTWARE.
//
//==================END_PROLOG=========================================

package WebSphereSamples.ConnPool;

/**
 * This is a Home interface for the Entity Bean
 */
public interface EmployeeBMPHome extends javax.ejb.EJBHome {

/**
 *
 * @return WebSphereSamples.ConnPool.EmployeeBMP
 * @param empNo java.lang.String
 * @param firstName java.lang.String
 * @param lastName java.lang.String
 * @param middleInit java.lang.String
 * @param edLevel int
 */
WebSphereSamples.ConnPool.EmployeeBMP create(java.lang.String empNo, java.lang.String
 firstName, java.lang.String lastName, java.lang.String middleInit, int edLevel)
 throws javax.ejb.CreateException, java.rmi.RemoteException;
```

Note: The previous line of code (beginning `WebSphereSamples.ConnPool...`) should be entered on one line; it is split here for formatting purposes.

```
/**
 * findByPrimaryKey method comment
 * @return WebSphereSamples.ConnPool.EmployeeBMP
 * @param key WebSphereSamples.ConnPool.EmployeeBMPKey
 * @exception java.rmi.RemoteException
 * @exception javax.ejb.FinderException
```

```
 */
WebSphereSamples.ConnPool.EmployeeBMP findByPrimaryKey(WebSphereSamples.ConnPool.
 EmployeeBMPKey key) throws java.rmi.RemoteException, javax.ejb.FinderException;
```

Note: The previous line of code should be entered on one line; it is split here for formatting purposes.

```
}
//==================START_PROLOG=======================================
//
//   5630-A23, 5630-A22,
//   (C) COPYRIGHT International Business Machines Corp. 2002
//   All Rights Reserved
//   Licensed Materials - Property of IBM
//   US Government Users Restricted Rights - Use, duplication or
//   disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
//
//   IBM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING
//   ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
//   PURPOSE. IN NO EVENT SHALL IBM BE LIABLE FOR ANY SPECIAL, INDIRECT OR
//   CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF
//   USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR
//   OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE
//   OR PERFORMANCE OF THIS SOFTWARE.
//
//==================END_PROLOG=========================================

package WebSphereSamples.ConnPool;

/**
 * This is an Enterprise Java Bean Remote Interface
 */
public interface EmployeeBMP extends javax.ejb.EJBObject {

/**
 *
 * @return int
  */
int getEdLevel() throws java.rmi.RemoteException;
/**
 *
 * @return java.lang.String
 */
java.lang.String getFirstName() throws java.rmi.RemoteException;
/**
 *
 * @return java.lang.String
 */
java.lang.String getLastName() throws java.rmi.RemoteException;
/**
 *
 * @return java.lang.String
 */
java.lang.String getMiddleInit() throws java.rmi.RemoteException;
/**
 *
 * @return void
 * @param newEdLevel int
 */
void setEdLevel(int newEdLevel) throws java.rmi.RemoteException;
/**
 *
 * @return void
 * @param newFirstName java.lang.String
 */
void setFirstName(java.lang.String newFirstName) throws java.rmi.RemoteException;
/**
```

```
 *
 * @return void
 * @param newLastName java.lang.String
 */
void setLastName(java.lang.String newLastName) throws java.rmi.RemoteException;
/**
 *
 * @return void
 * @param newMiddleInit java.lang.String
 */
void setMiddleInit(java.lang.String newMiddleInit) throws java.rmi.RemoteException;
}
//==================START_PROLOG=====================================
//
//   5630-A23, 5630-A22,
//   (C) COPYRIGHT International Business Machines Corp. 2002
//   All Rights Reserved
//   Licensed Materials - Property of IBM
//   US Government Users Restricted Rights - Use, duplication or
//   disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
//
//   IBM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING
//   ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
//   PURPOSE. IN NO EVENT SHALL IBM BE LIABLE FOR ANY SPECIAL, INDIRECT OR
//   CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF
//   USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR
//   OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE
//   OR PERFORMANCE OF THIS SOFTWARE.
//
//==================END_PROLOG=======================================

package WebSphereSamples.ConnPool;

/**
* This is a Primary Key Class for the Entity Bean
**/
public class EmployeeBMPKey implements java.io.Serializable {
 public String empNo;
 final static long serialVersionUID = 3206093459760846163L;

/**
* EmployeeBMPKey() constructor
*/
public EmployeeBMPKey()  {
}
/**
* EmployeeBMPKey(String key) constructor
*/
public EmployeeBMPKey(String key)  {
 empNo = key;
}
/**
* equals method
* - user must provide a proper implementation for the equal method. The generated
*   method assumes the key is a String object.
*/
public boolean equals (Object o)  {
 if (o instanceof EmployeeBMPKey)
  return empNo.equals(((EmployeeBMPKey)o).empNo);
 else
  return false;
}
/**
* hashCode method
* - user must provide a proper implementation for the hashCode method. The generated
```

```
*    method assumes the key is a String object.
*/
public int hashCode ()  {
 return empNo.hashCode();
```

Example: Handling data access exception - error mapping in DataStoreHelper

Error mapping is necessary because various database vendors can provide differing SQL errors and codes that might mean the same things. For example, the *StaleConnectionException* has different codes in different databases. The **DB2** SQLCODEs of *1015, 1034, 1036* and so on, indicate that the connection is no longer available because of a temporary database problem. The **Oracle** SQLCODEs of *28, 3113, 3114* and so on indicate the same situation.

To provide portability for applications, WebSphere Application Server provides a *DataStoreHelper* interface to enable mapping of these codes to the WebSphere Application Server exceptions. The following code segment illustrates how to add two error codes into the error map:

```
public class NewDSHelper extends GenericDataStoreHelper
{
  public NewDSHelper()
  {
    super(null);
    java.util.Hashtable myErrorMap = null;
    myErrorMap = new java.util.Hashtable(2);
    myErrorMap.put(new Integer(-803), myDuplicateKeyException.class);
    myErrorMap.put(new Integer(-1015), myStaleConnectionException.class);
    myErrorMap.put("S1000", MyTableNotFoundException.class);
    setUserDefinedMap(myErrorMap);
    ...
  }
}
```

Using embedded Structured Query Language in Java (SQLJ) support

WebSphere Application Server for z/OS Version 5.1 provides support for applications that use Structured Query Language in Java (SQLJ) to connect to a DB2 for z/OS backend database. The level of support provided depends on the driver that is used to connect to DB2 for z/OS such as:

- The DB2 for z/OS Legacy Driver in conjunction with the DB2 for z/OS Local JDBC Provider (RRS) which supports the use of SQLJ backed servlets, session beans, and entity beans using Bean Managed Persistence
- The DB2 Universal JDBC Driver in conjunction with the DB2 Universal JDBC Driver Provider which supports the use of SQLJ backed servlets, session beans, and entity beans using Bean Managed Persistence or Container Managed Persistence

Assembling data access applications

1. Define the resource reference attributes through the Assembly Toolkit.
2. Bind this resource reference to a resource like a J2EE Connector Architecture (JCA) connection factory or a data source.
3. Configure isolation level, access intent assembly settings.
   a. Right-click your EJB module in a J2EE Hierarchy view and click **Open With > Deployment Descriptor Editor**.
   b. In an EJB Deployment Descriptor editor, select the **Access** tab.

c. Under **Isolation Level**, click **Add**.
d. Select the isolation level, enterprise beans, and method elements. For information on isolation levels, press F1.
e. Click **Finish**.

4. Map enterprise beans to database tables.

Resource adapter archive file

A Resource Adapter Archive (RAR) file is a Java archive (JAR) file used to package a resource adapter for the Java 2 Connector (J2C) Architecture for WebSphere Application Server.

A RAR file can contain the following:
- Enterprise information system (EIS) supplied resource adapter implementation code in the form of JAR files or other executables, such as dynamic link lists (DLL).
- Utility classes.
- Static documents, such as HTML files, images, and sound files.
- J2C common client interfaces, such as *cci.jar*.

The standard file extension of a RAR file is *.rar*.

In the Assembly Toolkit, RAR files are called *connectors*.

Assembling resource adapter (connector) modules

A resource adapter archive (RAR) file contains code that implements a library for connecting with a backend Enterprise Information System (EIS). You might see the terms resource adapter *modules*, resource adapter *connectors* and resource adapter *archive files* used interchangeably.

Use the Assembly Toolkit to assemble a *connector* in any of the following ways:
- Import an existing RAR file.
- Create a new connector module.

1. Start the Assembly Toolkit.
2. Optional: Open the J2EE perspective to work with J2EE projects. Click **Window > Open Perspective > Other > J2EE**.
3. Optional: Open the J2EE Hierarchy view. Click **Window > Show View > J2EE Hierarchy**. Other helpful views include the Project Navigator view (**Window > Show View > Other > J2EE > Project Navigator**) and the Navigator view (**Window > Show View > Navigator**).
4. Migrate RAR files created with the Application Assembly Tool (AAT) or a different tool to the Assembly Toolkit. To migrate files, import your RAR files or connector modules to the Assembly Toolkit.
5. Create a new connector module.
6. Verify the contents of the new connector module in either of the following ways:
   - In the J2EE Hierarchy view, expand **Conector Modules** and view the new module.
   - Click **Window > Show View > Navigator** to see the associated files for the connector module in a Navigator view.

Deploying data access applications

Before installing a data access application into the WebSphere Application Server environment, you must first ensure that the appropriate database objects are available. This action includes creating and configuring any databases or tables required, setting necessary configuration parameters to handle expected load, and configuring any necessary JDBC providers and data source objects for servlets, enterprise beans, and client applications to use.

1. If your database configuration does not already exist:
   a. Create a database to hold the data.
   b. Create tables required by your application.
      If your application uses entity enterprise beans to access the data
      You can create the tables using the data definition language (DDL) generated from the enterprise bean configuration. For more information, see Recreating database tables from the exported table data definition language.
      If your application does not use entity beans
      You must use your database server interfaces to create the tables.
   c. See Minimum required properties for vendor-specific data sources for certain vendor's databases requirements.
2.
3. See Creating a JDBC provider using the administrative console if your enterprise application contains a Web application or an EJB application that uses connection pooling to access a relational database.
4. See Configuring data access for application clients if your enterprise application contains an application client that accesses a relational database.
5. Consider the security of lookups with component managed authentication. See Security of lookups with component managed authentication for more information.

Installing Java 2 Connector resource adapters

1. Click **Resources**.
2. Click **Resource Adapters**.
3. Click **Install RAR**. The Install RAR button opens a dialog that enables you to install a J2EE Connector Architecture (JCA) connector and create a resource adapter for it. You can also use the **New** button, but the New button creates only a new resource adapter (the JCA connector must already be installed on the system).

   Note: When installing a RAR file using this dialog, the scope you define on the Resource Adapters page has no effect on where the RAR file is installed. You can install RAR files only at the *node* level. The node on which the file is installed is determined by the scope on the **Install RAR** page. (The scope you set on the Resource Adapters page determines the scope of the new resource adapters, which you can install at the server, node, or cell level.)
4. Browse to find the appropriate RAR file.
   - If your RAR file is located on your local workstation, select **Local path** and browse to find the file.
   - If your RAR file is located on your server, select **Server path** and specify the fully qualified path to the file.

5. Click **Next**.
6. Enter the resource adapter name and any other properties needed under *General Properties*. If you install a J2C Resource Adapter that includes *Native path* elements, consider the following: If you have more than one native path element, and one of the native libraries (native library A) is dependent on another library (native library B), then you must copy native library B to a *system* directory. Because of limitations on Windows NT and most Unix platforms, an attempt to load a native library does not look in the current directory.
7. Click **OK**.

Installing resource adapters within applications

1. Assemble an application with resource adapter archive (RAR) modules in it. See Assembling applications.
2. Install the application following the steps in Installing a new application. In the **Map modules to application servers** step, specify target servers or clusters for each RAR file. Be sure to map all other modules that use the resource adapters defined in the RAR modules to the same targets.
3. Click **Finish** > **Save** to save the changes.
4. Create connection factories for the newly installed application.
   a. Open the administrative console.
   b. Click **Applications** > **Enterprise Applications** > *application name*.
   c. Click **Connector Modules** in the Related Items section of the page.
   d. Click *filename.rar*.
   e. Click **Resource adapter** in the Additional Properties section of the page.
   f. Click **J2C Connection Factories** in the Additional Properties section of the page.
   g. Click on an existing connection factory to update it, or **New** to create a new one.

      If you install a J2C Resource Adapter that includes *Native path* elements, consider the following: If you have more than one native path element, and one of the native libraries (native library A) is dependent on another library (native library B), then you must copy native library B to a *system* directory. Because of limitations on Windows NT and most Unix platforms, an attempt to load a native library does not look in the current directory.

      After you create and save the connection factories, you can modify the resource references defined in various modules of the application and specify the Java Naming and Directory Interface (JNDI) names of the connection factories wherever appropriate.

   Note: A given native library can only be loaded one time for each instance of the Java virtual machine (JVM). Because each application has its own classloader, separate applications with embedded RAR files cannot both use the same native library. The second application receives an exception when it tries to load the library.

   If any application deployed on the application server uses an embedded RAR file that includes native path elements, then you must always ensure that you shut down the application server cleanly, with no outstanding transactions. If the application server does not shut down cleanly it performs *recovery* upon server restart and loads any required RAR files and native libraries. On completion of recovery, do not attempt

any application-related work. Shut down the server and restart it. No further recovery is attempted by the application server on this restart, and normal application processing can proceed.

Resource Adapters collection

Use this page to select a Resource Adapter, which represents an archive file containing code that implements a library for connecting with some EIS (Enterprise Information System) backend such as CICS, SAP, or PeopleSoft.

To view this administrative console page, click **Resources** >**Resource Adapters**.

Name:

Specifies a list of the available resource adapters.

These resource adapters can be supplied by a third party vendor other than IBM. Typically, a single resource adapter can only connect to one type of backend system (EIS) but it can support many different configurations for connections to that EIS. The resource adapter has many configuration properties that are defined in the J2C specification and set by the vendor who supplies the code.

Data type String

Resource adapter settings:

Use this page to specify settings for a Resource Adapter.

This resource adapter can be supplied by a third party vendor other than IBM. Typically, a single resource adapter can only connect to one type of backend system (EIS) but it can support many different configurations for connections to that EIS. The resource adapter has many configuration properties that are defined in the J2C specification and set by the vendor who supplies the code.

To view this administrative console page, click **Resources** >**Resource Adapters** > *resource_adapter*.

Scope:

Specifies the level to which this resource definition is visible -- the cell, node, or server level.

WebSphere 5.0 allows resources such as JDBCProviders, Namespace Bindings, or Shared Libraries at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

Cell The most general scope. Resources defined at the Cell scope are visible from all Nodes and servers, unless they have been overridden. To view resources defined in the cell scope, do not specify a server or a node name in the scope selection form.

Node The default scope for most resource types. Resources defined at the Node scope override any duplicates defined at the Cell scope and are visible to all servers on the same node, unless they have been overridden at a server scope on that node. To view resources defined in a node scope, do not specify a server, but select a node name in the scope selection form.

Server The most specific scope for defining resources. Resources defined at the Server scope override any duplicate resource definitions defined at the Cell scope or parent Node scope and are visible only to a specific server. To

view resources defined in a server scope, specify a server name as well as a node name in the scope selection form.

When resources are created, they are always created into the current scope that is selected in the panel. To view resources in other scopes, specify a different node or server in the scope selection form.

Data type String

Name:

Specifies the name of the resource provider.

A string with no spaces meant to be a meaningful text identifier for the resource provider.

Data type String

Description:

Specifies a text description of the resource factory.

A free-form text string to describe the resource factory and its purpose.

Data type String

Archive path:

Specifies the path to the .RAR file containing the module for this resource adapter.

Data type String

Classpath:

Specifies a list of paths or JAR file names which together form the location for the resource provider classes.

This includes any additional libraries needed by the resource adapter. The resource adapter code base itself is automatically added to the classpath, but if anything outside the .RAR is needed it can be specified here.

Data type String

Native path:

Specifies a list of paths which forms the location for the resource provider native libraries.

The resource adapter code base itself is automatically added to the classpath, but if anything outside the .RAR is needed it can be specified here.

Data type String

Ensuring applications obtain valid connections

If your application is accessing pooled connections, you can enable connection pretesting to help prevent it from obtaining connections that are no longer valid. Connection pretesting is a way to test connections from the free pool before giving them to the client.

When a backend resource, such as a database, fails, pooled connections that are not valid might exist in the free pool. This is likely when the purge policy is *failingConnectionOnly*, meaning only the failing connection is removed from the pool. Depending on the failure, the remaining connections in the pool might not be valid.

1. Enable and configure connection pretesting.
   a. Open the j2c.properties file.

      The file is located in install_root/properties/j2c.properties.
   b. Using the example in the file, create an <advanced connection properties> entry corresponding to your datasource.
   c. Enable connection pretesting.

      Set `<testConnection>` to true.
   d. Specify how often to retry, should the pretest fail.

      Set `<testConnectionRetryInterval>` to the number of seconds for the pretest connection thread to wait between attempts to create and pretest a connection. It will do so until it is successful, at which point the connection pool starts processing getConnection() requests and the pretest connection thread ends.

      If the pretest fails, new connection requests are rejected with a ResourceAllocationException indicating Failed preTestConnection. Pool requests are blocked until the test connection thread is successful.

   For example, this j2c.properties configuration enables connection pretesting to occur every 5 seconds until it is successful:

```
<advanced-connection-properties connectionFactoryJNDIName=jdbc/DataSource>
   <testConnection>true</testConnection>
   <testConnectionRetryInterval>5</testConnectionRetryInterval>
</advanced-connection-properties>
```

2. Specify the SQL statement to use when connection pretesting is enabled.

   Configure a preTestSQLString custom property for the data source.
   a. Open the administrative console.
   b. Navigate to Resources > JDBC_provider > Data Sources > data_source > Custom Properties.
   c. Specify preTestSQLString as the property name.
   d. Specify an SQL statement as the property value. This SQL statement is used for connection pretesting, which helps to ensure that applications obtain valid data sources from connection pools. When connection pretesting is enabled, the SQL statement is executed to determine whether the connection is good. See "Database connection settings that can be added to the administrative console" on page 111 for help configuring this property.

      Note: It is important to specify a valid SQL statement for the preTestSQLString custom property for best performance. The SQL statement should be one that is executed quickly and does not result in any exception thrown. If an exception is thrown, WebSphere Application Server run time needs to check whether the exception

indicates the connection is bad or not. This additional checking can affect performance. One example of this SQL statement is ″SELECT 1 from [TESTTABLE]″, in which [TESTTABLE] is a valid table with only a few rows. A similarly appropriate selection for an Oracle data source is ″SELECT USER FROM DUAL″.

j2c.properties file

Use the j2c.properties file to configure additional database connection settings beyond those available in the administrative console. For example, you can enable connection pretesting, which helps to ensure that applications obtain valid database connections from connection pools.

Location

The file is located at *install_root*/properties/j2c.properties.

Usage notes

- Is this file read-only?

  No
- How do you edit this file?

  Edit this file directly, in conjunction with administrative console settings and custom properties.
- Is this file updated by a product component?

  No.
- How and when are the contents of this file used?

  The j2c.properties file is used to specify fine-tuning parameters of the JCA connection management runtime. It is interrogated by various parts of the pool manager and connection manager. If the file is not found, or if a particular value is not found in the file, suitable default values are used.
- What must you do to have changes take effect?

  Use any editor to save the file, then stop and restart the server.

Database connection settings that can be added to the administrative console

Use the Properties page to set and monitor settings associated with database connections, but that are not displayed on the main settings page by default.

To view the administrative console page, click Resources > JDBC_provider > Data Sources > data_source > Custom Properties.

To add properties to the page, click New and enter at least a name (case-sensitive) and value for the property. Then click Apply. When you are finished entering properties, click OK

preTestSQLString:

Specify the SQL statement to use for connection pretesting, which helps to ensure that applications obtain valid data sources from connection pools. When connection pretesting is enabled, the SQL statement is executed to determine whether the connection is good.

It is important to specify a valid SQL statement for the preTestSQLString custom property for best performance. The SQL statement should be one that is executed quickly and does not result in any exception thrown. For example, you might

specify SELECT COUNT(*) from TESTTABLE.

Data type	String - Valid SQL statement
Default	• For Oracle database: SELECT USER FROM DUAL • For other supported databases: SELECT COUNT(*) FROM rra.x1x1x0x4x

Creating and configuring a JDBC provider and data source

1. Decide which data source to use: Data source (Version 4.0), see Data Sources (Version 4) or a Version 5.0 data source, see Data Source collection .
2. Create a JDBC provider.

   From the administrative console, see Creating a JDBC provider using the administrative console.

   OR

   Using the Java Management Extensions (JMX) API, see Creating a JDBC provider and data source using the Java Management Extensions API.
3. Create a data source.

   From the administrative console, see Creating a data source using the administrative console.

   OR

   Using the JMX API, see Creating a JDBC provider and data source using the Java Management Extensions API.
4. Bind the resource reference. See Binding to a data source
5. Test the connection (for non-container-managed persistence usage).

   Note: When the data source configuration is saved, it will be saved in the *resource.xml* file. You should not need to modify the **jdbc-resource-provider-templates.xml**. However, special consideration should be taken if you need to update the **jdbc-resource-provider-templates.xml file**. See Creating and configuring a JDBC provider and data source using the Java Management Extensions API.

Verifying a connection

Many test connection problems can be easily fixed by verifying some configuration parameters This article provides a checklist of steps that you must complete to enable a successful connection. Click on the link for more information on a specific step.

If your connection is still not successful after completing these steps and reviewing the applicable information, check the SYSOUT of your application servant region for warning or exception messages.

1. Create the authentication data alias.
2. Create the JDBC provider.
3. Create a Version 5.0 or 4.0 data source.
4. Save the data source.
5. Restart the server.

   If you are running Websphere Application Server Version 5.0.2, you do not have to complete this step unless you also created a new Java Authentication and Authorization Service (JAAS) entry.

6. Test the connection

   You can test your connection from the data source collection view or the data source details view. Access either view in the administrative console, and then select a connection from the list. Click the **test connection** button on the connection.

   You must complete all the steps and restart the server before you test the connection. Neither the authentication data alias nor the WebSphere Application Server environment variables (in Version 5.0.1) are available before a server restart.

Creating and configuring a JDBC provider using the administrative console

To access a database you must first create a JDBC provider. You can do this from the administrative console. You need to know where your database software is installed and where the drivers are located.

1. Open the administrative console.
2. Click **Resources** > **JDBC Providers**.
3. Select the *scope* of your definition.
4. Click **New**.
5. Use the drop-down list to select the type of JDBC provider to create. If the list of supported JDBC provider types does not include the JDBC provider that you want to use, select the **User-Defined JDBC Provider**. Consult the JDBC provider vendor's documentation for information on specific required properties.

   Do not change the name of the JDBC provider if you create it by selecting an existing JDBC provider from the drop-down list. This restriction does not apply if you are using the **User-Defined JDBC Provider** feature and supplying the provider information.
6. Click **Apply** to view the settings page for your JDBC provider.
7. Enter the properties for your JDBC provider For more information, see JDBC Provider settings.
8. Click **Apply** to view the page with your new JDBC provider settings. Note that there is now an *Additional Properties* section on this page. To set up a data source, click one of the data sources links. For more information about creating a data source, see Creating a data source with the administrative console.
9. Click **OK** to return to the JDBC providers page, where your new JDBC provider appears in the list.

JDBC Provider collection:

Use this page to view a JDBC provider.

To view this administrative console page, click **Resources** > **JDBC Providers** in the console navigation tree.

Notice the *Scope* of your JDBC provider. If you pick anything other than the default of *Node* the provider might not be available in other scope contexts.

Name:

Specifies a text identifier for this provider.

For example this field can be *DB2 JDBC Provider (XA)*.

Data type String

Description:

Specifies a text string describing this provider.

Data type String

JDBC provider settings:

Use this page to create or modify JDBC provider settings

To view this administrative console page, click **Resources** > **JDBC Providers** > *JDBC_provider*.

Scope:

Specifies the level to which this resource definition is visible -- the cell, node, or server level.

Resources such as JDBC Providers, Namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

Note that no matter what the scope of a defined resource, the resource's properties only apply at an individual server level. For example, if you define the scope of a data source at the Cell level, all users in that Cell can look up and use that data source, which is unique within that Cell. However, resource property settings are local to each server in the Cell. For example, if you define *max connections* to 10, then each server in that Cell can have 10 connections.

Cell The most general scope. Resources defined at the Cell scope are visible from all Nodes and servers, unless they are overridden. To view resources defined in the cell scope, do not specify a server or a node name in the scope selection form.

Node The default scope for most resource types. Resources defined at the Node scope override any duplicates defined at the Cell scope and are visible to all servers on the same node, unless they are overridden at a server scope on that node. To view resources defined in a node scope, do not specify a server, but select a node name in the scope selection form.

Server The most specific scope for defining resources. Resources defined at the Server scope override any duplicate resource definitions defined at the Cell scope or parent Node scope and are visible only to a specific server. To view resources defined in a server scope, specify a server name as well as a node name in the scope selection form.

When resources are created, they are always created into the current scope selected in the panel. To view resources in other scopes, specify a different node or server in the scope selection form.

Data type String

Name:

Specifies the name of the resource provider.

Data type String

Description:

Specifies a text description for the resource provider.

Data type String

Classpath:

Specifies a list of paths or JAR file names which together form the location for the resource provider classes.

For example, *For example, /usr/lpp/db2/db2710/classes/db2j2classes.zip*

Classpath entries are separated by using the ENTER key and must not contain path separator characters (such as ';' or ':'). Classpaths contain variable (symbolic) names which you can substitute using a variable map. Check the driver installation notes for the specific required JAR file names.

Data type String

Native Library Path:

Specifies a list of paths that forms the location for the resource provider native libraries.

Native path entries are separated by using the ENTER key and must not contain path separator characters (such as ';' or ':'). Native paths can contain variable (symbolic) names which you can substitute using a variable map.

Data type String

Implementation Classname:

Specifies the Java class name of the JDBC driver implementation.

This class is available in the driver file mentioned in the Classpath description above. For example *COM.ibm.db2.jdbc.DB2XADataSource*.

Data type String

New JDBC Provider:

Use this page to choose a type of JDBC Provider to create.

To view this administrative console page, click **Resources** >**JDBC Providers** > **New**.

JDBC Providers:

Specifies the names of the supported JDBC Provider types.

If the list of supported JDBC Provider types does not include the JDBC Provider that you want to use, select the *User-Defined JDBC Provider*. You might need to consult the documentation for the JDBC Provider for more information on specific properties required by that provider.

Data type Drop-down list

Configuring the Application Server for use with the DB2 Universal JDBC Driver

This topic describes how to configure the WebSphere Application Server for z/OS for use with the DB2 Universal JDBC Driver. This information applies only to Universal Driver type 2 connectivity on DB2 UDB for z/OS.

Ensure that DB2 for z/OS Version 7 or higher is installed on your z/OS system.

To enable the DB2 Universal JDBC Driver to run under WebSphere Application Server for z/OS, open the WebSphere Application Server for z/OS administrative console and perform the following steps:

1. Identify the DB2 Universal JDBC Driver to WebSphere Application Server for z/OS.

   To identify the DB2 Universal JDBC Driver to WebSphere Application Server for z/OS, do the following to provide the location of the driver files and the location of the global properties file:

   a. In the administrative console, select **Environment > Manage WebSphere variables**.
   b. In the WebSphere variables page, select **New**.
   c. Enter `DB2UNIVERSAL_JDBC_DRIVER_PATH` for **Name** and the path for the DB2 Universal JDBC Driver for **Value**. (The usual path is `/usr/lpp/db2810/jcc`.)
   d. Click **OK** and then **Save**.

2. Create a JDBC Provider for the DB2 Universal JDBC Driver.

   To create a new JDBC provider that is associated with the DB2 Universal JDBC Driver:

   a. In the administrative console, select **Resources > JDBC Providers**.
   b. Under **Scope**, select a server, and then click **Apply**. **DO NOT** select the same server for the JDBC provider for the DB2 Universal Driver as is already being as the JDBC provider for the JDBC/SQLJ Driver for z/OS.
   c. In the same panel, select **New**.
   d. In the JDBC providers drop-down list, click User-defined **JDBC Provider**, and then click **Apply**.
   e. Enter the following values for the JDBC provider:

      Name

      Description
      Custom JDBC 2.0-compliant provider configuration

      Classpath
      - ${DB2UNIVERSAL_JDBC_DRIVER_PATH}/classes/db2jcc.jar
      - ${DB2UNIVERSAL_JDBC_DRIVER_PATH}/classes/db2jcc_license_cisuz.jar
      - ${DB2UNIVERSAL_JDBC_DRIVER_PATH}/classes/sqlj.zip

      Native Library Path
      ${DB2UNIVERSAL_JDBC_DRIVER_PATH}/lib

Implementation Classname
com.ibm.db2.jcc.DB2ConnectionPoolDataSource

f. Click **Apply**, and then click **Save**.

3. Create a data source for the DB2 Universal JDBC Driver JDBC provider.
   a. In the administrative console, select **Resources > JDBC Providers**.
   b. In the JDBC providers drop-down list, select **Universal Driver type 2 connectivity on DB2 UDB for z/OS JDBC provider**.
   c. Select **Data Sources**, and then click **New**.
   d. Complete the Data Source page:
      1) Fill in the following fields:

         Name
         DB2JccT2zOSDataSource

         JNDI Name
         jdbc/DB2JccT2zOSDataSource

         Description
         New JDBC DataSource

      2) Select the **Container managed persistence** checkbox.
      3) Optionally, enter a value in the **Container-managed Authentication Alias** field.

         If your system administrator set up an authentication alias that contains the user ID and password that is to be associated with all connections that are obtained from the data source, specify that authentication alias. Otherwise, leave this field blank.

         If all of the following conditions are true, all connections that are obtained from the data source are associated with the server:

         - The **Container-managed Authentication Alias** field is blank.
         - The resource reference to which the data source is bound is defined with `resAuth=Container`.

           Use the Application Assembly Tool (AAT) or WebSphere Studio Application Developer Integration Edition (WSADIE) to indicate the `resauth=Container` setting.

   e. Click **Apply**, and then select **Custom Properties**.
   f. In the Custom Properties Page, select **New**
   g. Create each of the following variables. Enter the Name fields exactly as shown. After creating a variable, click **Apply**.

Name	Value	Description
databaseName	The name of the data source, which is the same as the DB2 location name.	New JDBC DataSource, or some other descriptive name.
loginTimeout	0	Any value.
planName	If you bound the JDBC packages into a plan, the name of that plan. NULLID otherwise. The default is DSNJDBC.	Any value.

   h. Optionally, repeat the previous two steps to create variables for any of the other DB2 Universal JDBC Driver properties that are listed in *DB2 Universal Database for OS/390 and z/OS Application Programming Guide and Reference for Java*[tm], SC26-9932-04.
   a. Click **Save**.

4. Bind JDBC DataSource references to a DB2 Universal JDBC Driver data source. Before you can run a JDBC application in WebSphere Application Server, under the DB2 Universal JDBC Driver, you must bind the JDBC DataSource references in your application to the data source that you defined under the JDBC provider for Universal Driver type 2 connectivity on DB2 UDB for z/OS. See the Information Center topic *Deploying > Applications > Deployment > Installing a new application > Preparing for application installation settings* for a description of how to create this bind.

The DB2 Universal JDBC Driver can now run under the Application Server.

Creating and configuring a data source using the administrative console

After you create a JDBC provider, you must create a data source to access the backend data store. Follow these steps to create either a new Version 5.0 data source or a Version 4.0 data source.

1. Open the administrative console.
2. Click **Resources** > **JDBC Providers**.
3. You might need to change the **Scope** selection to find the JDBC provider for which you want to create a data source. Scope settings are used to limit the availability of resources to a particular cell, node, or server. When new items are created in this view, they are created within the current scope.
4. Choose the JDBC resource provider in which you want to create the data source. The detail page for this provider appears.
5. Click **Data Sources** in Additional Properties if you want to create a Version 5.0 data source. If you want to create a Version 4.0 data source, click **Data Sources (Version 4)** in Additional Properties. The *Data Sources* or *Data Sources (Version 4)* page appears.
6. Click **New** to display the settings page for your V5.0 or V4.0 data source.
7. Enter the properties for your data source.

   For your Version 5.0 data source:

   - Optionally choose an existing **Component-managed Authentication Alias** or **Container-managed Authentication Alias** from the lists. These aliases are used for database authentication in run time. If you do not set one of these fields and your database requires the user ID and password to get a connection, then you receive an exception during run time.

     If your resource authentication (res-auth) is set to *Application*, set the alias in the Component-managed Authentication Alias. If your res-auth is set to *Container*, set the Container-managed Authentication Alias. If your database does not support a *user ID* and *password* on a connection (for example, Cloudscape), then do not specify the alias in either one of these entries.

     Because defining a *user ID* and *password* in the Custom Properties page is not always desirable (your password can be seen by anyone who accesses the *resources.xml* file), you can define the user ID and password as an alias. To define a new alias from the J2C Authentication Data Entries choice, follow these steps:

     a. Click **Apply**. The J2C Authentication Data Entries choice does not appear on the "New" data source page, but does appear on an existing data source's page. Because you were just creating a new data source, you must click on the apply button for the J2C Authentication Data Entries choice to show. This also saves the properties that you have already applied to this data source.
     b. Click **J2C Authentication Data Entries** in the Related Items section.

c. Click **New** on the J2C Authentication Data Entries page.
d. Fill in the fields on the resulting page. Click **Apply**.
e. Return to the data source page.
f. If the new alias does not appear in the picklists for Component- or Container-managed Authentication Alias, close the page and re-open it.

For more information, see Data source settings, Data source (Version 4) settings., and Minimum required properties for vendor-specific data sources

8. Click **Apply** to view a page with your new data source settings. Note that there are now *Additional Properties* and *Related Items* sections on this page. Additional Properties contains the *Connection Pool* and *Custom Properties* choices. Some database vendors might require additional custom properties for data sources that access the database. Click on either or both of these to modify their properties.

   The Related Items section contains the *J2C Authentication Data Entries* choice. Here, you can specify a list of user IDs and passwords for use by Java 2 Connector (J2C) security.
9. Click **Save**.
10. Return to the data source page to confirm that your new data source appears in the list.

Data Source collection:

Use this page to create or modify a Version 5.0 data source.

To view this administrative console page, click **Resources** > **JDBC Providers** > *JDBC_provider* > **Data Sources**.

Name:

Specifies the display name of this data source.

Data type String

JNDI name:

Specifies the Java Naming and Directory Interface (JNDI) name for this data source.

Data type String

Description:

Specifies a text description of the data source.

Data type String

Category:

Specifies a string that you can use to classify or group a data source.

Data type String

Data Sources (Version 4):

Use this page to view the settings of a Version 4.0 style data source.

These Version 4.0 data sources use the WebSphere Application Server Version 4.0 Connection Manager architecture. All EJB 1.1 modules must use a Version 4.0 data source.

To view this administrative console page, click **Resources** > **JDBC Providers** > *JDBC_provider* > **Data Sources (Version 4)**.

Name:

Specifies a text identifier of the data source.

Data type String

JNDI Name:

Specifies the Java Naming and Directory Interface (JNDI) name of the data source.

Data type String

Description:

Specifies a text description of the data source.

Data type String

Category:

Specifies a text string that you can use to classify or group the data source.

Data type String

Data source (Version 4) settings:

Use this page to create a Version 4.0 style data source. This data source uses the WebSphere Application Server Version 4.0 Connection Manager architecture. All the EJB1.x modules must use this data source.

To view this administrative console page, click **Resources** > **JDBC Providers** > *JDBC_provider* > **Data Sources (Version 4)** > *data_source*.

Scope:

Specifies the level to which this resource definition is visible -- the cell, node, or server level.

Resources such as JDBC Providers, Namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

Note that no matter what the scope of a defined resource, the resource's properties only apply at an individual server level. For example, if you define the scope of a data source at the Cell level, all users in that Cell can look up and use that data

source, which is unique within that Cell. However, resource property settings are local to each server in the Cell. For example, if you define *max connections* to 10, then each server in that Cell can have 10 connections.

Cell The most general scope. Resources defined at the Cell scope are visible from all Nodes and servers, unless they are overridden. To view resources defined in the cell scope, do not specify a server or a node name in the scope selection form.

Node The default scope for most resource types. Resources defined at the Node scope override any duplicates defined at the Cell scope and are visible to all servers on the same node, unless they are overridden at a server scope on that node. To view resources defined in a node scope, do not specify a server, but select a node name in the scope selection form.

Server The most specific scope for defining resources. Resources defined at the Server scope override any duplicate resource definitions defined at the Cell scope or parent Node scope and are visible only to a specific server. To view resources defined in a server scope, specify a server name as well as a node name in the scope selection form.

When resources are created, they are always created into the current scope selected in the panel. To view resources in other scopes, specify a different node or server in the scope selection form.

Data type String

Name:

Specifies the display name for the resource.

For example you can set this field to *Test Data Source*.

Data type String

JNDI Name:

Specifies the Java Naming and Directory Interface (JNDI) name.

Distributed computing environments often employ naming and directory services to obtain shared components and resources. Naming and directory services associate names with locations, services, information, and resources.

Naming services provide name-to-object mappings. Directory services provide information on objects and the search tools required to locate those objects.

There are many naming and directory service implementations, and the interfaces to them vary. JNDI provides a common interface that is used to access the various naming and directory services.

For example, you can use the name *jdbc/markSection*.

If you leave this field blank a JNDI name is generated from the name of the data source. For example, a data source name of markSection generates a JNDI name of *jdbc/markSection*.

After you set this value, save it, and restart the server, you can see this string when you run *dumpnamespace*.

Data type String

Description:

Specifies a text description for the resource.

Data type String

Category:

Specifies a category string that you can use to classify or group the resource.

Data type String

Database Name:

Specifies the name of the database that this data source accesses.

For example, you can call the database *SAMPLE*.

Data type String

Default User ID:

Specifies the user name to use for connecting to the database.

For example, you can use the ID *db2admin*.

Data type String

Default Password:

Specifies the password used for connecting to the database.

For example, you can use the password *db2admin*.

Data type String

Custom Properties collection:

Use this page to view the custom properties.

Custom properties are unique to each resource, zero or more can be required.

Note: After you enter this page, the first thing to do is click on the *Required* field to sort in descending order. All of the required (true) values are then sorted at the beginning of the page. You must repeat this process until all the required properties are set.

To view this administrative console page, click **Resources** >**JDBC Providers** > *JDBC_provider* > **Data Sources** > *data_source*> **Custom Properties**.

Name:

Specifies the property name.

You must ensure that the resource adapter has the setting for this name.

Data type String

Value:

Specifies the property value.

Data type Integer

Description:

Specifies text to describe any bounds or well-defined values for this property.

Data type String

Required:

Specifies properties that are required for this resource.

Data type String

Custom property settings:

Use this page to set custom properties that might be required for Resource Providers and Resource Factories

To view this administrative console page, click **Resources** >**JDBC Providers** > *JDBC_provider* > **Data Sources** > *data_source*> **Custom Properties** > *custom_property*.

Scope:

Specifies the level to which this resource definition is visible -- the cell, node, or server level.

Resources such as JDBC Providers, Namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

Note that no matter what the scope of a defined resource, the resource's properties only apply at an individual server level. For example, if you define the scope of a data source at the Cell level, all users in that Cell can look up and use that data source, which is unique within that Cell. However, resource property settings are local to each server in the Cell. For example, if you define *max connections* to 10, then each server in that Cell can have 10 connections.

Cell The most general scope. Resources defined at the Cell scope are visible from all Nodes and servers, unless they are overridden. To view resources defined in the cell scope, do not specify a server or a node name in the scope selection form.

Node The default scope for most resource types. Resources defined at the Node scope override any duplicates defined at the Cell scope and are visible to all servers on the same node, unless they are overridden at a server scope

on that node. To view resources defined in a node scope, do not specify a server, but select a node name in the scope selection form.

Server The most specific scope for defining resources. Resources defined at the Server scope override any duplicate resource definitions defined at the Cell scope or parent Node scope and are visible only to a specific server. To view resources defined in a server scope, specify a server name as well as a node name in the scope selection form.

When resources are created, they are always created into the current scope selected in the panel. To view resources in other scopes, specify a different node or server in the scope selection form.

Data type String

Required:

Specifies properties that are required for this resource.

Setting can be *true* or *false*.

Data type String

Name:

Specifies the name associated with this property (PortNumber, ConnectionURL, etc).

Data type String

Value:

Specifies the value associated with this property in this property set.

Data type Integer

Description:

Specifies text to describe any bounds or well-defined values for this property.

Data type String

Type:

Specifies the fully qualified Java data type of this property .

There are specific types that are valid:
- java.lang.Boolean
- java.lang.String
- java.lang.Integer
- java.lang.Double
- java.lang.Byte
- java.lang.Short
- java.lang.Long

- java.lang.Float
- java.lang.Character

Data type Pick file

Custom Properties (Version 4) collection:

Use this page to view properties for a Version 4.0 datasource.

To view this administrative console page, click **Resources** > **JDBC Providers** > *JDBC_provider* > **Data Sources (Version 4)** > *data_source* > **Custom Properties**

Name:

Specifies the name of the custom property

Insure that the data source has the setter of this property.

Data type String

Value:

Specifies the value of the custom property.

Data type Integer

Description:

Specifies text to describe any bounds or well-defined values for this property.

Data type String

Required:

Specifies properties that are required for this resource.

Data type String

Custom property (Version 4) settings:

Use this page to add properties for a Version 4.0 datasource.

To view this administrative console page, click **Resources** >**JDBC Providers**> *JDBC_provider* > **Data Sources (Version 4)** > *data_source* > **Custom Properties** > *custom_property*.

Scope:

Specifies the level to which this resource definition is visible -- the cell, node, or server level.

Resources such as JDBC Providers, Namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

Note that no matter what the scope of a defined resource, the resource's properties only apply at an individual server level. For example, if you define the scope of a data source at the Cell level, all users in that Cell can look up and use that data source, which is unique within that Cell. However, resource property settings are local to each server in the Cell. For example, if you define *max connections* to 10, then each server in that Cell can have 10 connections.

Cell The most general scope. Resources defined at the Cell scope are visible from all Nodes and servers, unless they are overridden. To view resources defined in the cell scope, do not specify a server or a node name in the scope selection form.

Node The default scope for most resource types. Resources defined at the Node scope override any duplicates defined at the Cell scope and are visible to all servers on the same node, unless they are overridden at a server scope on that node. To view resources defined in a node scope, do not specify a server, but select a node name in the scope selection form.

Server The most specific scope for defining resources. Resources defined at the Server scope override any duplicate resource definitions defined at the Cell scope or parent Node scope and are visible only to a specific server. To view resources defined in a server scope, specify a server name as well as a node name in the scope selection form.

When resources are created, they are always created into the current scope selected in the panel. To view resources in other scopes, specify a different node or server in the scope selection form.

Data type String

Required:

Specifies properties that are required for this resource.

Data type String

Name:

Specifies the name associated with this property (PortNumber, ConnectionURL, etc).

Insure that the data source has the setter of this property.

Data type String

Value:

Specifies the value associated with this property in this property set.

Data type Integer

Description:

Specifies text to describe any bounds or well-defined values for this property.

Data type String

Type:

Specifies the fully qualified Java type of this property (java.lang.Integer, java.lang.Byte).

Data type String

Creating a JDBC provider on multiple nodes

These are the instructions to create a JDBC provider in a Network Development (ND) environment. This sample configuration involves two base nodes: node A and node B added to ND on node C.

1. Open the administrative console.
2. Click **Resources** > **JDBC Providers**. The default listing is at the *node* level.
3. Leave the node text field blank and click **Apply** to change the scope to the *cell* level.
4. Click **New** to create a new JDBC provider at the cell level. The classpath for your new JDBC provider will already be filled in, and part of that classpath will be specified using a symbolic variable, for example *${DB2390_JDBC_DRIVER_PATH}/classes/db2j2classes.zip*. Leave it at the default.
5. Finish creating the JDBC provider.
6. Click **Environment**.
7. Click **Manage WebSphere Variables**.
8. For each node, select the symbolic variable used in the classpath of your JDBC provider, and provide a value that is appropriate for the selected node. For example, if the classpath of your JDBC provider uses the symbolic variable *${DB2390_JDBC_DRIVER_PATH}*, you might supply the value /usr/lpp/db2 on one node and /usr/lpp/db2710 on another node, depending on where your DB2 390 installation is located.
9. Click **DB2_JDBC_DRIVER_PATH** (this already exists by default). Here provide the path (in the *value* field) where *db2java.zip* exists on the selected node.
10. Click **Apply** and save the changes.

    Note: This variable must be defined on each node under the cell.

Creating and configuring a JDBC provider and data source using the Java Management Extensions API

You need two JAR files in your classpath -- *wsexception.jar* and *wasjmx.jar*. The following command is an example for setting your classpath:

```
export CLASSPATH=$CLASSPATH:/WebSphere/V5R1M0/AppServer/lib/wsexception.jar:
 /WebSphere/V5R1M0/AppServer/lib/wasjmx.jar
```

Note: You must enter this command on one line; here it is split for formatting purposes only.

The usual program follows these main points:

1. Look up the host and get an administration client handle.
2. Get a configuration service handle.
3. Update the *resource.xml* file using the configuration service as desired.
   a. Add a JDBC provider

b. Add the data source

c. Add the connection factory (for container-managed persistence)

4. Reload the *resource.xml* file to bind the newly created data source into the JNDI namespace. Perform this step if you want to use the newly created data source right away without restarting the application server.

   a. Locate the DataSourceConfigHelper MBean using the name.

   b. Put together the signature and parameters for the call.

   c. Invoke the reload() call.

   Note: The **jdbc-resource-provider-templates.xml** file should be handled as read only. When updating this file, special consideration should be taken. Before installing the PTF, you should save your updated **jdbc-resource-provider-templates.xml file**. After applying the PTF, you will need to verify that the new **jdbc-resource-provider-templates.xml** file has your correct entries. If the entries are not valid, you will have to merge your changes into this new **jdbc-resource-provider-templates.xml** file manually.

Example: Using the Java Management Extensions API to create a JDBC driver and data source for container-managed persistence:

```
//
// "This program may be used, executed, copied, modified and distributed without royalty
// for the purpose of developing, using, marketing, or distributing."
//
// Product 5630-A36,  (C) COPYRIGHT International Business Machines Corp., 2001, 2002
// All Rights Reserved * Licensed Materials - Property of IBM
//

import java.util.*;
import javax.sql.*;
import javax.transaction.*;
import javax.management.*;
import java.io.*;

import com.ibm.websphere.management.*;
import com.ibm.websphere.management.configservice.*;
import com.ibm.ws.management.*;
import com.ibm.ws.exception.*;

/**
 * Creates a node-scoped resource.xml entry for a
 * DB2 for zOS Local JDBC Provider (RRS) DataSource
 * when WebSphere security is not enabled
 *
 * The datasource created is for CMP use.
 *
 * To run this example, the following must be done:
 *
 * 1)  Set the WAS_HOME environment variable to the location of
 *     your WebSphere Application Server for z/OS Configuration
 *     directory
 *
 *     Example:  export WAS_HOME=/WebSphereV5R1M0/AppServer
 *
 * 2)  Set the following environment variables:
 *
 *     export WAS_LIB=$WAS_HOME/lib
 *     export WAS_CLASSPATH=[DIRECTORY_CONTAINING_THIS_FILE]
 *     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_LIB/jmxc.jar
 *     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_LIB/wsexception.jar
 *     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_LIB/admin.jar
 *     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_LIB/wasjmx.jar
 *     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_HOME/java/jre/lib/ext/mail.jar
 *     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_LIB/ibmjlog.jar
```

```
 *
 * 3)  Execute the following commands:
 *
 *     javac -classpath $WAS_CLASSPATH CreateDataSourceCMP.java
 *     java -classpath $WAS_CLASSPATH CreateDataSourceCMP
 */
public class CreateDataSourceCMP {

    String dsName           = "MyDataSourceCMP";// ds display name , also jndi name and CF name
    String dbName           = "LOC1";           // database name
    String authDataAlias    = "IBMUSER";        // an authentication data alias
    String uid              = "IBMUSER";        // userid
    String pw               = "IBMUSER";        // password
    String dbclasspath      = "/db2beta/db2710/classes/db2j2classes.zip"; // path to the db driver
    String dblibpath        = "/db2beta/db2710/lib"; // path to the db lib directory

    /**
     * Main method.
     */
    public static void main(String[] args) {
        CreateDataSourceCMP cds = new CreateDataSourceCMP();
        try {
            cds.run(args);
        } catch (com.ibm.ws.exception.WsException ex) {
            System.out.println("Caught this " + ex);
            ex.printStackTrace();
            ex.getCause().printStackTrace();
        } catch (Exception ex) {
            System.out.println("Caught this " + ex);
            ex.printStackTrace();
        }
    }

    /**
     * This method creates the datasource using JMX.
     * The datasource created here is only written into resources.xml.
     * It is not bound into namespace until the server is restarted, or an application started
     */
    public void run(String[] args) throws Exception {
        try {
            // Initialize the AdminClient.
            Properties adminProps = new Properties();
            adminProps.setProperty(AdminClient.CONNECTOR_TYPE, AdminClient.CONNECTOR_TYPE_SOAP);
            adminProps.setProperty(AdminClient.CONNECTOR_HOST, "localhost");
            adminProps.setProperty(AdminClient.CONNECTOR_PORT, "8880");
            AdminClient adminClient = AdminClientFactory.createAdminClient(adminProps);

            // Get the ConfigService implementation.
            com.ibm.websphere.management.configservice.ConfigServiceProxy configService =
                new com.ibm.websphere.management.configservice.ConfigServiceProxy(adminClient);

            Session session = new Session();

            // Use this group to add to the node scoped resource.xml.
            ObjectName node1 = ConfigServiceHelper.createObjectName(null, "Node", null);
            ObjectName[] matches = configService.queryConfigObjects(session, null, node1, null);
            node1 = matches[0];     // use the first node found

            // Use this group to add to the server1 scoped resource.xml.
            ObjectName server1 = ConfigServiceHelper.createObjectName(null, "Server", "server1");
            matches = configService.queryConfigObjects(session, null, server1, null);
            server1 = matches[0];   // use the first server found

            // Create the JDBCProvider
            String providerName = "My DB2 for zOS Local JDBC Provider (RRS) for CMP";
            System.out.println("Creating JDBCProvider " + providerName );
```

```
      // Prepare the attribute list
      AttributeList provAttrs = new AttributeList();
      provAttrs.add(new Attribute("name", providerName));
      provAttrs.add(new Attribute("implementationClassName",
"com.ibm.db2.jcc.DB2ConnectionPoolDataSource"));
```

Note: The previous line of code (beginning `provAttrs.add(new Attribute("implementationClassName...)` must be entered on one line; it is split here for formatting purposes only.

```
provAttrs.add(new Attribute("description","Legacy DB2 for z/OS driver using RRS"));

      //create it
      ObjectName jdbcProv = configService.createConfigData(session,node1,"JDBCProvider",
"resources.jdbc:JDBCProvider", provAttrs);
```

Note: The previous line of code must be entered on one line; it is split here for formatting purposes only.

```
// now plug in the classpath
      configService.addElement(session,jdbcProv,"classpath",dbclasspath,-1);
      configService.addElement(session,jdbcProv,"nativepath",dblibpath,-1);

      // Search for RRA so we can link it to the datasource
      ObjectName rra = ConfigServiceHelper.createObjectName
(null, "J2CResourceAdapter", null);
```

Note: The previous line of code must be entered on one line; it is split here for formatting purposes only.

```
matches = configService.queryConfigObjects(session, node1, rra, null);
      rra = matches[0]; // use the first J2CResourceAdapter segment for builtin_rra

      // Prepare the attribute list
      AttributeList dsAttrs = new AttributeList();
      dsAttrs.add(new Attribute("name", dsName));
      dsAttrs.add(new Attribute("jndiName", "jdbc/" + dsName));
      dsAttrs.add(new Attribute("datasourceHelperClassname",
"com.ibm.websphere.rsadapter.DB2DataStoreHelper"));
```

Note: The previous line of code must be entered on one line; it is split here for formatting purposes only.

```
dsAttrs.add(new Attribute("statementCacheSize", new Integer(10)));
      dsAttrs.add(new Attribute("relationalResourceAdapter", rra));
// this is where we make the link to "builtin_rra"
      dsAttrs.add(new Attribute("description", "JDBC Datasource for CMP usage"));
      dsAttrs.add(new Attribute("authDataAlias", authDataAlias));

      // Create the datasource
      System.out.println("  **  Creating datasource");
      ObjectName dataSource = configService.createConfigData
(session,jdbcProv,"DataSource", "resources.jdbc:DataSource",dsAttrs);
```

Note: The previous line of code must be entered on one line; it is split here for formatting purposes only.

```
      // Add a propertySet.
      AttributeList propSetAttrs = new AttributeList();
      ObjectName resourcePropertySet = configService.createConfigData
(session,dataSource,"propertySet","",propSetAttrs);
```

Note: The previous line of code must be entered on one line; it is split here for formatting purposes only.

```
        // Add resourceProperty databaseName
        AttributeList propAttrs1 = new AttributeList();
        propAttrs1.add(new Attribute("name", "databaseName"));
        propAttrs1.add(new Attribute("type", "java.lang.String"));
        propAttrs1.add(new Attribute("value", dbName));

        configService.addElement(session,resourcePropertySet,
    "resourceProperties",propAttrs1,-1);
```

Note: The previous line of code must be entered on one line; it is split here for formatting purposes only.

```
        // Now Create the corresponding J2CResourceAdapter Connection Factory object.
        ObjectName jra = ConfigServiceHelper.createObjectName(null,"J2CResourceAdapter",null);

        // Get all the J2CResourceAdapter, and I want to add my datasource
        System.out.println("  **  Get all J2CResourceAdapter's");
        ObjectName[] jras = configService.queryConfigObjects(session, node1, jra, null);

        int i=0;

        for (;i< jras.length;i++) {
            System.out.println(ConfigServiceHelper.getConfigDataType(jras[i])
    + " " + i + " = "
```

Note: The previous line of code must be entered on one line; it is split here for formatting purposes only.

```
+ jras[i].getKeyProperty(SystemAttributes._WEBSPHERE_CONFIG_DATA_DISPLAY_NAME)
                    + "\nFrom scope ="
                    + jras[i].getKeyProperty(SystemAttributes._WEBSPHERE_CONFIG_DATA_ID));
                // quit on the first builtin_rra
                if (jras[i].getKeyProperty(SystemAttributes._WEBSPHERE_CONFIG_DATA_DISPLAY_NAME)
      .equals("WebSphere Relational Resource Adapter")) {
```

Note: The previous line of code must be entered on one line; it is split here for formatting purposes only.

```
      break;
                }
            }


            if (i >= jras.length) {
                System.out.println("Did not find builtin_rra J2CResourceAdapter object
    creating CF anyways" );
```

Note: The previous line of code must be entered on one line; it is split here for formatting purposes only.

```
    } else {
                System.out.println("Found builtin_rra J2CResourceAdapter object at index
     " + i + " creating CF" );
```

Note: The previous line of code must be entered on one line; it is split here for formatting purposes only.

```
    }

        // Prepare the attribute list
        AttributeList cfAttrs = new AttributeList();
        cfAttrs.add(new Attribute("name", dsName + "_CF"));
        cfAttrs.add(new Attribute("authMechanismPreference","BASIC_PASSWORD"));
        cfAttrs.add(new Attribute("authDataAlias",authDataAlias));
        cfAttrs.add(new Attribute("cmpDatasource", dataSource ));
```

```
      // this is where we make the link to DataSource's xmi:id
            ObjectName cf = configService.createConfigData(session,jras[i],"CMPConnectorFactory",
      "resources.jdbc:CMPConnectorFactory",cfAttrs);
```

Note: The previous line of code must be entered on one line; it is split here for formatting purposes only.

```
            // ===== start Security section
            System.out.println("Creating an authorization data alias " + authDataAlias);

            // Find the parent security object
            ObjectName security = ConfigServiceHelper.createObjectName(null, "Security", null);

            ObjectName[] securityName = configService.queryConfigObjects
     (session, null, security, null);
```

Note: The previous line of code must be entered on one line; it is split here for formatting purposes only.

```
      security=securityName[0];

            // Prepare the attribute list
            AttributeList authDataAttrs = new AttributeList();
            authDataAttrs.add(new Attribute("alias", authDataAlias));
            authDataAttrs.add(new Attribute("userId", uid));
            authDataAttrs.add(new Attribute("password", pw));
            authDataAttrs.add(new Attribute("description","Auto created alias for datasource"));

            //create it
            ObjectName authDataEntry = configService.createConfigData
     (session,security,"authDataEntries", "JAASAuthData",authDataAttrs);
```

Note: The previous line of code must be entered on one line; it is split here for formatting purposes only.

```
// ===== end Security section

            // Save the session
            System.out.println("Saving session" );
            configService.save(session, false);

            // reload resources.xml to bind the new datasource into the name space
            reload(adminClient,true);
        } catch (Exception ex) {
            ex.printStackTrace(System.out);
            throw ex;
        }
    }

    /**
     * Get the DataSourceConfigHelperMbean and call reload() on it
     *
     * @param adminClient
     * @param verbose true - print messages to stdout
     */
    public void reload(AdminClient adminClient,boolean verbose) {
        if (verbose) {
            System.out.println("Finding the Mbean to call reload()");
        }

        // First get the  Mbean
        ObjectName handle = null;
        try {
            ObjectName queryName = new ObjectName("WebSphere:type=DataSourceCfgHelper,*");
            Set s = adminClient.queryNames(queryName, null);
            Iterator iter = s.iterator();
            if (iter.hasNext()) handle = (ObjectName)iter.next();
```

```
        } catch (MalformedObjectNameException mone) {
            System.out.println("Check the program variable queryName" + mone);
        } catch (com.ibm.websphere.management.exception.ConnectorException ce) {
            System.out.println("Cannot connect to the application server" + ce);
        }

        if (verbose) {
            System.out.println("Calling reload()");
        }
        Object result = null;
        try {
            result = adminClient.invoke(handle, "reload", new Object[] {}, new String[] {});
        } catch (MBeanException mbe) {
            if (verbose) {
                System.out.println("\tMbean Exception calling reload" + mbe);
            }
        } catch (InstanceNotFoundException infe) {
            System.out.println("Cannot find reload ");
        } catch (Exception ex) {
            System.out.println("Exception occurred calling reload()"  + ex);
        }

        if (result==null && verbose) {
            System.out.println("OK reload()");
        }
    }
}
```

Example: Using the Java Management Extensions API to create a JDBC driver and data source for bean-managed persistence, session beans, or servlets:

```
// "This program may be used, executed, copied, modified and distributed without royalty for the
// purpose of developing, using, marketing, or distributing."
//
// Product 5630-A36,  (C) COPYRIGHT International Business Machines Corp., 2001, 2002
// All Rights Reserved * Licensed Materials - Property of IBM
//
import java.util.*;
import javax.sql.*;
import javax.transaction.*;
import javax.management.*;

import com.ibm.websphere.management.*;
import com.ibm.websphere.management.configservice.*;
import com.ibm.ws.exception.WsException;

/**
 * Creates a node-scoped resource.xml entry for a
 * DB2 for zOS Local JDBC Provider (RRS) DataSource
 * when WebSphere security is not enabled
 *
 * To run this example, the following must be done:
 *
 * 1)  Set the WAS_HOME environment variable to the location of
 *     your WebSphere Application Server for z/OS Configuration
 *     directory
 *
 *     Example:  export WAS_HOME=/WebSphereV5R1M0/AppServer
 *
 * 2)  Set the following environment variables:
 *
 *     export WAS_LIB=$WAS_HOME/lib
 *     export WAS_CLASSPATH=[DIRECTORY_CONTAINING_THIS_FILE]
 *     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_LIB/jmxc.jar
 *     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_LIB/wsexception.jar
 *     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_LIB/admin.jar
 *     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_LIB/wasjmx.jar
```

```
*     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_HOME/java/jre/lib/ext/mail.jar
*     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_LIB/ibmjlog.jar
*
* 3)  Execute the following commands:
*
*     javac -classpath $WAS_CLASSPATH CreateDataSourceBMP.java
*     java -classpath $WAS_CLASSPATH CreateDataSourceBMP
*/
public class CreateDataSourceBMP {

   String dsName = "MyDataSourceBMP";  // ds display name , also jndi name and CF name
   String dbName = "LOC1";             // database name
   String authDataAlias = "IBMUSER";   // an authentication data alias
   String uid = "IBMUSER";             // userid
   String pw  = "IBMUSER";             // password
   String dbclasspath = "/db2beta/db2710/classes/db2j2classes.zip";  // path to the db driver
   String dblibpath = "/db2beta/db2710/lib"; // path to the db native library directory

   /**
    * Main method.
    */
   public static void main(String[] args) {
       CreateDataSourceBMP cds = new CreateDataSourceBMP();
       try {
           cds.run(args);
       } catch (com.ibm.ws.exception.WsException ex) {
           System.out.println("Caught this " + ex   );
           ex.printStackTrace();
       } catch (Exception ex) {
           System.out.println("Caught this " + ex );
           ex.printStackTrace();
       }
   }

   /**
    * This method creates the datasource using JMX.
    *
    * The datasource created here is only written into resources.xml.
    * It is not bound into the namespace until the server is restarted,
 * or an application is started
    */
   public void run(String[] args) throws Exception {

       try {
           // Initialize the AdminClient.
           Properties adminProps = new Properties();
           adminProps.setProperty(AdminClient.CONNECTOR_TYPE, AdminClient.CONNECTOR_TYPE_SOAP);
           adminProps.setProperty(AdminClient.CONNECTOR_HOST, "localhost");
           adminProps.setProperty(AdminClient.CONNECTOR_PORT, "8880");
           AdminClient adminClient = AdminClientFactory.createAdminClient(adminProps);

           // Get the ConfigService implementation.
           com.ibm.websphere.management.configservice.ConfigServiceProxy configService =
           new com.ibm.websphere.management.configservice.ConfigServiceProxy(adminClient);

           Session session = new Session();

           // Use this group to add to the node scoped resource.xml.
           ObjectName node1 = ConfigServiceHelper.createObjectName(null, "Node", null);
           ObjectName[] matches = configService.queryConfigObjects(session, null, node1, null);
           node1 = matches[0];     // use the first node found

           // Use this group to add to the server1 scoped resource.xml.
           ObjectName server1 = ConfigServiceHelper.createObjectName(null, "Server", "server1");
           matches = configService.queryConfigObjects(session, null, server1, null);
           server1 = matches[0];   // use the first server found
```

```
        // Create the JDBCProvider
        String providerName = "My DB2 for zOS Local JDBC Provider (RRS) for BMP";
        System.out.println("Creating JDBCProvider " + providerName );

        // Prepare the attribute list
        AttributeList provAttrs = new AttributeList();
        provAttrs.add(new Attribute("name", providerName));
        provAttrs.add(new Attribute("implementationClassName",
"com.ibm.db2.jcc.DB2ConnectionPoolDataSource"));
```

Note: The previous line of code (beginning provAttrs.add(new Attribute("implementationClassName" must be entered on one line; it is split here for formatting purposes.

```
provAttrs.add(new Attribute("description","Legacy DB2 for z/OS driver using RRS"));

        //create it
        ObjectName jdbcProv = configService.createConfigData(session,node1,"JDBCProvider",
"resources.jdbc:JDBCProvider",provAttrs);
```

Note: The previous line of code must be entered on one line; it is split here for formatting purposes.

```
// now plug in the classpath
        configService.addElement(session,jdbcProv,"classpath",dbclasspath,-1);
        configService.addElement(session,jdbcProv,"nativepath",dblibpath,-1);

        // Search for RRA so we can link it to the datasource
        ObjectName rra = ConfigServiceHelper.createObjectName
(null, "J2CResourceAdapter", null);
```

Note: The previous line of code (beginning ObjectName rra =) must be entered on one line; it is split here for formatting purposes.

```
matches = configService.queryConfigObjects(session, node1, rra, null);
        rra = matches[0]; // use the first J2CResourceAdapter segment for builtin_rra

        // Prepare the attribute list
        AttributeList dsAttrs = new AttributeList();
        dsAttrs.add(new Attribute("name", dsName));
        dsAttrs.add(new Attribute("jndiName", "jdbc/" + dsName));
        dsAttrs.add(new Attribute("datasourceHelperClassname",
"com.ibm.websphere.rsadapter.DB2DataStoreHelper"));
```

Note: The previous line of code (beginning dsAttrs.add(new Attribute("datasourceHelperClassname" must be entered on one line; it is split here for formatting purposes.

```
dsAttrs.add(new Attribute("statementCacheSize", new Integer(10)));
        dsAttrs.add(new Attribute("relationalResourceAdapter", rra));
// the previous line is where we make the link to "builtin_rra"
        dsAttrs.add(new Attribute("description", "JDBC Datasource for BMP usage"));
        dsAttrs.add(new Attribute("authDataAlias",authDataAlias));

        // Create the datasource
        System.out.println("  **  Creating datasource");
        ObjectName dataSource = configService.createConfigData(session,jdbcProv,"DataSource",
"resources.jdbc:DataSource",dsAttrs);
```

Note: The previous line of code must be entered on one line; it is split here for formatting purposes.

```
        // Add a propertySet.
        AttributeList propSetAttrs = new AttributeList();
        ObjectName resourcePropertySet =configService.createConfigData
(session,dataSource,"propertySet","",propSetAttrs);
```

Note: The previous line of code must be entered on one line; it is split here for formatting purposes.

```
        // Add resourceProperty databaseName
        AttributeList propAttrs1 = new AttributeList();
        propAttrs1.add(new Attribute("name", "databaseName"));
        propAttrs1.add(new Attribute("type", "java.lang.String"));
        propAttrs1.add(new Attribute("value", dbName));

        configService.addElement(session,resourcePropertySet,
    "resourceProperties",propAttrs1,-1);
```

Note: The previous line of code must be entered on one line; it is split here for formatting purposes.

```
        // ===== start Security section
        System.out.println("Creating an authorization data alias " + authDataAlias);

        // Find the parent security object
        ObjectName security = ConfigServiceHelper.createObjectName(null, "Security", null);
        ObjectName[] securityName =
    configService.queryConfigObjects(session, null, security, null);
```

Note: The previous line of code must be entered on one line; it is split here for formatting purposes.

```
    security=securityName[0];

        // Prepare the attribute list
        AttributeList authDataAttrs = new AttributeList();
        authDataAttrs.add(new Attribute("alias", authDataAlias));
        authDataAttrs.add(new Attribute("userId", uid));
        authDataAttrs.add(new Attribute("password", pw));
        authDataAttrs.add(new Attribute("description","Auto created alias for datasource"));

        //create it
        ObjectName authDataEntry =
    configService.createConfigData(session,security,"authDataEntries",
    "JAASAuthData",authDataAttrs);
```

Note: The previous line of code (beginning `ObjectName authDataEntry =` must be entered on one line; it is split here for formatting purposes.

```
// ===== end Security section

        // Save the session
        System.out.println("Saving session" );
        configService.save(session, false);

        // reload resources.xml
        reload(adminClient,true);

      } catch (Exception ex) {
          ex.printStackTrace(System.out);
          throw ex;
      }
  }

  /**
   * Get the DataSourceConfigHelperMbean and call reload() on it
   *
   * @param adminClient
   * @param verbose true - print messages to stdout
   */
  public void reload(AdminClient adminClient,boolean verbose) {
      if (verbose) {
          System.out.println("Finding the Mbean to call reload()");
```

```
        }

        // First get the  Mbean
        ObjectName handle = null;
        try {
            ObjectName queryName = new ObjectName("WebSphere:type=DataSourceCfgHelper,*");
            Set s = adminClient.queryNames(queryName, null);
            Iterator iter = s.iterator();
            if (iter.hasNext()) handle = (ObjectName)iter.next();
        } catch (MalformedObjectNameException mone) {
            System.out.println("Check the program variable queryName" + mone);
        } catch (com.ibm.websphere.management.exception.ConnectorException ce) {
            System.out.println("Cannot connect to the application server" + ce);
        }

        if (verbose) {
            System.out.println("Calling reload()");
        }
        Object result = null;
        try {
            result = adminClient.invoke(handle, "reload", new Object[] {}, new String[] {});
        } catch (MBeanException mbe) {
            if (verbose) {
                System.out.println("\tMbean Exception calling reload" + mbe);
            }
        } catch (InstanceNotFoundException infe) {
            System.out.println("Cannot find reload ");
        } catch (Exception ex) {
            System.out.println("Exception occurred calling reload()"  + ex);
        }

        if (result==null && verbose) {
            System.out.println("OK reload()");
        }
    }
}
```

Example: Test a connection to a data source: This resource adapter test program ensures that the MBean interfaces work. The following interfaces are tested:

- getPropertiesForDataSource()
- reload()
- testConnectionToDataSource()

```
// "This program may be used, executed, copied, modified and distributed without royalty for the
// purpose of developing, using, marketing, or distributing."
// Product 5630-A36,  (C) COPYRIGHT International Business Machines Corp., 2001, 2002
// All Rights Reserved * Licensed Materials - Property of IBM

import java.util.*;
import javax.sql.DataSource;
import javax.transaction.*;
import javax.management.*;

import com.ibm.websphere.management.*;
import com.ibm.websphere.management.configservice.*;
import com.ibm.ws.exception.WsException;
import com.ibm.websphere.rsadapter.DSPropertyEntry;

/**
 * Tests a connection to a DataSource when WebSphere
 * Security is disabled.
 *
 * To run this example, the following must be done:
 *
 * 1)  Set the WAS_HOME environment variable to the location of
 *     your WebSphere Application Server for z/OS Configuration
 *     directory
```

```
 *
 *     Example:  export WAS_HOME=/WebSphereV5R1M0/AppServer
 *
 * 2)  Set the following environment variables:
 *
 *     export WAS_LIB=$WAS_HOME/lib
 *     export WAS_CLASSPATH=[DIRECTORY_CONTAINING_THIS_FILE]
 *     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_LIB/jmxc.jar
 *     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_LIB/wsexception.jar
 *     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_LIB/admin.jar
 *     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_LIB/wasjmx.jar
 *     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_HOME/java/jre/lib/ext/mail.jar
 *     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_LIB/ibmjlog.jar
 *     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_LIB/utils.jar
 *
 * 3)  Execute the following commands:
 *
 *     javac -classpath $WAS_CLASSPATH TestDS.java
 *     java -classpath $WAS_CLASSPATH TestDS
 */
public class TestDS {

    String port = "8880";
    String host = "localhost";
    final static boolean verbose = true;

    /**
     * Main method.
     */
    public static void main(String[] args) {
        TestDS cds = new TestDS();
        try {
            cds.run(args);
        } catch (com.ibm.ws.exception.WsException ex) {
            System.out.println("Caught this " + ex   );
            ex.printStackTrace();
        } catch (Exception ex) {
            System.out.println("Caught this " + ex );
            ex.printStackTrace();
        }
    }


    /**
     * This method tests the DataSourceCfgHelper Mbean.
     */
    public void run(String[] args) throws Exception {
        try {
            System.out.println("Connecting to the application server.......");
            // Initialize the AdminClient.
            Properties adminProps = new Properties();
            adminProps.setProperty(AdminClient.CONNECTOR_TYPE, AdminClient.CONNECTOR_TYPE_SOAP);
            adminProps.setProperty(AdminClient.CONNECTOR_HOST, host);
            adminProps.setProperty(AdminClient.CONNECTOR_PORT, port);
            AdminClient adminClient = null;
            try {
                adminClient = AdminClientFactory.createAdminClient(adminProps);
            } catch (com.ibm.websphere.management.exception.ConnectorException ce) {
                System.out.println("Cannot make a connection to the application server\n"+ce);
                System.exit(1);
            }

            // First get the  Mbean
            ObjectName handle = null;
            try {
                ObjectName queryName = new ObjectName("WebSphere:type=DataSourceCfgHelper,*");
                Set s = adminClient.queryNames(queryName, null);
```

```
            Iterator iter = s.iterator();
            if (iter.hasNext()) handle = (ObjectName)iter.next();
        } catch (MalformedObjectNameException mone) {
            System.out.println("Check the program variable queryName" + mone);
        } catch (com.ibm.websphere.management.exception.ConnectorException ce) {
            System.out.println("Cannot connect to the application server" + ce);
        }

        String dsClassName = "com.ibm.db2.jcc.DB2ConnectionPoolDataSource";
        String providerClassPath = "/db2beta/db2710/classes/db2j2classes.zip";
        String[] signature = { "java.lang.String", "java.lang.String"};
        Object[] params = { dsClassName, providerClassPath};
        Object result = null;

        if (verbose) {
            System.out.println("Calling getPropertiesForDataSource() for "
 + dsClassName + "\n");
```

Note: The previous line of code (beginning `System.out.println`) must be entered on one line; it is split here for formatting purposes.

```
    }
        try {
           // get the properties
           result = adminClient.invoke(handle, "getPropertiesForDataSource", params,
 signature);
```

Note: The previous line of code (beginning `result =`) must be entered on one line; it is split here for formatting purposes.

```
    } catch (MBeanException mbe) {
            if (verbose) {
                System.out.println("\tMbean Exception " + dsClassName);
            }
        } catch (InstanceNotFoundException infe) {
            System.out.println("Cannot find " + dsClassName);
        } catch (Exception ex) {
            System.out.println("Exception occurred calling getPropertiesForDataSource()
 for " + dsClassName + ex);
```

Note: The previous line of code (beginning `System.out.println`) must be entered on one line; it is split here for formatting purposes.

```
}

        // Pretty print what we found
        Iterator propIterator = ((List)result).iterator();
        System.out.println(format("Name",21)+ "|" + format("Default Value",34) +
    "|" + format("Type",17) +"|Reqd");
```

Note: The previous line of code (beginning `System.out.println`) must be entered on one line; it is split here for formatting purposes.

```
String line = "_____________________________________________________________________";
        System.out.println(line);
        while (propIterator.hasNext()) {
            DSPropertyEntry dspe = (DSPropertyEntry)propIterator.next();
            System.out.print(format(dspe.getPropertyName(),21)+"|"+
 format(dspe.getDefaultValue(),34) + "|");
            System.out.println(format(dspe.getPropertyType(),17) +"|"+
 ((dspe.isRequired())? " Y" : " N"));
```

Note: The previous two lines of code (both beginning `System.out.println`) must be entered on one line each; they are split here for formatting purposes.

```
}
            System.out.println(line);

            //---------------reload

            if (verbose) {
                System.out.println("Calling reload()");
            }
            try {
                result = adminClient.invoke(handle, "reload", new Object[] {}, new String[] {});
            } catch (MBeanException mbe) {
                if (verbose) {
                    System.out.println("\tMbean Exception calling reload" + mbe);
                }
            } catch (InstanceNotFoundException infe) {
                System.out.println("Cannot find reload ");
            } catch (Exception ex) {
                System.out.println("Exception occurred calling reload()"  + ex);
            }
            if (result==null && verbose) {
                System.out.println("OK reload()");
            }

            //---------------reload

            if (verbose) {
                System.out.println("\nTesting connection to the database using " + dsClassName);
            }

            String user = "IBMUSER";
            String password = "IBMUSER";
            Properties props = new Properties();
            props.setProperty("databaseName", "LOC1");
            props.setProperty("dataStoreHelperClass",
      "com.ibm.websphere.rsadapter.DB2DataStoreHelper");
```

Note: The previous line of code must be entered on one line; it is split here for formatting purposes.

```
     String[] signature2 = { "java.lang.String", "java.lang.String", "java.lang.String",
      "java.util.Properties", "java.lang.String","java.util.Locale"};
```

Note: The previous line of code must be entered on one line; it is split here for formatting purposes.

```
      Object[] params2 = { dsClassName, user, password,props,
     providerClassPath, Locale.US};
```

Note: The previous line of code must be entered on one line; it is split here for formatting purposes.

```
     Object result2 = null;

            try {
                // OK lets test.
                result2 = adminClient.invoke(handle, "testConnectionToDataSource",
      params2, signature2);
```

Note: The previous line of code must be entered on one line; it is split here for formatting purposes.

```
     } catch (MBeanException mbe) {
                if (verbose) {
                    System.out.println("\tMbean Exception " + dsClassName);
                }
            } catch (InstanceNotFoundException infe) {
                System.out.println("Cannot find " + dsClassName);
```

```
            } catch (RuntimeMBeanException rme) {
                Exception ex = rme.getTargetException();
                ex.printStackTrace(System.out);
                throw ex;
            } catch (Exception ex) {
                System.out.println("Exception occurred calling testConnectionToDataSource()
       for " + dsClassName + ex);
```

Note: The previous line of code (beginning `System.out.println` must be entered on one line; it is split here for formatting purposes.

```
       ex.printStackTrace();
            }

            // the testConnectionToDataSource method doesn't always throw an exception
            // if the test fails; it only returns a String showing the results,
            // so we can't really know if the test was successful
            //
            // the best option here is to just print the String for the user

            System.out.println("Test Result:  " + result2);

        } catch (RuntimeOperationsException roe) {
            Exception ex = roe.getTargetException();
            ex.printStackTrace(System.out);
            throw ex;
        } catch (Exception ex) {
            ex.printStackTrace(System.out);
            throw ex;
        }
    }

    /**
     * Format the string right justified in the space provided,
     * or truncate the string.
     *
     * @param in
     * @param length
     * @return
     */
    public String format(Object in, int length) {
        if (in ==null) {
            in = "-null-";
        }

        String ins = in.toString();
        int insLength = ins.length();
        if ( insLength > length) {
            return ins.substring(0,length);
        } else {
            StringBuffer sb = new StringBuffer(length);
            while (length - insLength > 0) {
                sb.append(" ");
                length--;
            }
            sb.append(ins);
            return sb.toString();
        }
    }
}
```

Example: Creating a JDBC provider and data source using Java Management Extensions API and the scripting tool: Following is a JACL (WSadmin - scripting tool) script used to create a data source and test the connection. This script:

- Creates a data source fvtDS_1
- Creates a 4.0 data source fvtDS_3

- Creates a container-managed persistence (CMP) data source linked to fvtDS_1
- Tests the connection

```
# Following is a JACL (WSadmin - scripting tool) script used to create a data source
# and test the connection.
# This script:
#   Creates a data source fvtDS_1
#   Creates a container-managed persistence (CMP) data source linked to fvtDS_1

# The classpath that will be used by your database driver
set driverClassPath "/db2beta/db2710/classes/db2j2classes.zip"
set driverNativePath "/db2beta/db2710/lib"

set server "server1"

set fvtbase "c:/wssb/fvtbase"

# Users and passwords..
set defaultUser "IBMUSER"
set defaultPassword "IBMUSER"
set aliasName "IBMUSER"

set databaseName "LOC1"

puts "Add an alias alias1"
set cell [$AdminControl getCell]
set sec [$AdminConfig getid /Cell:$cell/Security:/]

#-------------------------------------------------------------
# Create a JAASAuthData object for component-managed authentication
#-------------------------------------------------------------

puts "create JAASAuthData object for alias1"

set alias_attr    [list alias $aliasName]
set desc_attr     [list description "Alias 1"]
set userid_attr   [list userId $defaultUser]
set password_attr [list password $defaultPassword]
set attrs [list $alias_attr $desc_attr $userid_attr $password_attr]

set authdata [$AdminConfig  create JAASAuthData $sec $attrs]
$AdminConfig save

puts "Installing DB2 for zOS Local JDBC Provider (RRS) DataSource"

#Get the server name...
puts "Finding the server $server"

set servlist [$AdminConfig list Server]
set servsize [llength $servlist]
foreach srvr $servlist {
  set sname [lindex [lindex [$AdminConfig show $srvr {name}] 0] 1]
  if {($sname == $server)} {
    puts "Found server $srvr"
    set serv $srvr
  }
}

puts "Finding the Resource Adapter"
set rsadapter [$AdminConfig list J2CResourceAdapter $serv]

#Now create a JDBC Provider for the 5.0 data sources
puts "Creating the provider for com.ibm.db2.jcc.DB2ConnectionPoolDataSource"
set attrs1 [subst {{classpath $driverClassPath} {nativepath $driverNativePath}
 {implementationClassName com.ibm.db2.jcc.DB2ConnectionPoolDataSource}
 {name "FVTProvider2"} {description "DB2 JDBC Provider"}}]
```

Note: The previous line of code (beginning set attrs1) must be entered on one line; it is split here for formatting purposes.

```
set provider1 [$AdminConfig create JDBCProvider $serv $attrs1]

#Create the first data source
puts "Creating the datasource fvtDS_1"
set attrs2 [subst {{name fvtDS_1} {description "FVT DataSource 1"}}]
set ds1 [$AdminConfig create DataSource $provider1  $attrs2]

#Set the properties for the data source.
set propSet1 [$AdminConfig create J2EEResourcePropertySet $ds1 {}]

set attrs3 [subst {{name databaseName} {type java.lang.String} {value $databaseName}}]
$AdminConfig create J2EEResourceProperty $propSet1 $attrs3

set attrs10 [subst {{jndiName jdbc/fvtDS_1} {statementCacheSize 10}
 {datasourceHelperClassname com.ibm.websphere.rsadapter.DB2DataStoreHelper}
 {relationalResourceAdapter $rsadapter} {authM echanismPreference "BASIC_PASSWORD"}
 {authDataAlias $aliasName}}]
```

Note: The previous line of code (beginning set attrs10) must be entered on one line; it is split here for formatting purposes.

```
$AdminConfig modify $ds1 $attrs10

#Create the connection pool object...
$AdminConfig create ConnectionPool $ds1 {{connectionTimeout 1000} {maxConnections 30}
 {minConnections 1} {agedTimeout 1000} {reapTime 2000} {unusedTimeout 3000} }
```

Note: The previous line of code must be entered on one line; it is split here for formatting purposes.

```
#Now we will add a connection factory for the CMPs..
puts "Creating the CMP Connector Factory for fvtDS_1"
set attrs12 [subst {{name "FVT DS 1_CF"} {authMechanismPreference BASIC_PASSWORD}
 {cmpDatasource $ds1} {authDataAlias $aliasName}}]
```

Note: The previous line of code (beginning set attrs12) must be entered on one line; it is split here for formatting purposes.

```
set cf1 [$AdminConfig create CMPConnectorFactory $rsadapter $attrs12]

#Set the properties for the data source.
$AdminConfig create MappingModule $cf1 {{mappingConfigAlias "DefaultPrincipalMapping"}
 {authDataAlias "alias1"}}
```

Note: The previous line of code must be entered on one line; it is split here for formatting purposes.

```
$AdminConfig save
```

Test connection

The *test connection* service enables developers to test a connection to a data source.

There are three ways to test a connection to a database that uses the parameters defined in a data source in the WebSphere Application Server. You can use the administrative console, the *wsadmin* tool, or a Java stand alone program. All three processes invoke the same methods on the same MBean.

Administrative console: WebSphere Application Server enables you to test a connection from the administrative console by simply pushing a button. The *Data Source Collection* and *Data Source Details* pages have new **Test Connection** buttons. After you have defined and saved a data source, you can click this button to ensure that the parameters in the data source definition are correct. On the

collection page, you can select several data sources and test them all at once. Note that there are certain conditions that must be met first. For more information, see Testing a connection with the administrative console.

Wsadmin tool: The *wsadmin tool* provides a scripting interface to a full range of WebSphere Application Server administration activities. Because the Test Connection functionality is implemented as a method on an MBean, and wsadmin can invoke MBean methods, wsadmin can be utilized to test connections to DataSources. Following are the three options:

There is a *testConnection* facility in wsadmin that you can read about in Example: Migrating - Testing the DataSource object connection.

The *AdminControl* object of wsadmin has a testConnection operation that tests the configuration properties of a data source object. For information, see Testing a connection using wsadmin.

Finally, you can test a connection by invoking the MBean operation. You can find information about this by referring to the Information Center topic *Administering > System administration > Scripting > Managing running objects with scripting > Operation management examples with wsadmin > Example: Testing data source connection using wsadmin*.

Java stand alone program: The test connection commands can also be invoked from a Java program, by using JMX to connect directly to the MBean. For information about invoking the test connection operations before 5.0.1, see Example: Test a connection to a data source.

You can invoke the same test connection operation on the *DataSourceCfgHelper* MBean from a Java program that wsadmin uses, passing in the properties you want to test. You can find an example of this code here: Example: Test a connection using country and language (properties).

While both of these methods are still viable with this release, the preferred method is shown in the following paragraphs.

The new 5.0.1 test connection method can also be invoked through a Java program. The advantage is that you can pass the *configuration ID* of a configured data source, rather than the properties of the data source. This program uses JMX to connect to a running server and invoke the *testConnection* method on the *DataSourceCfgHelper* MBean. In a base installation, you connect to the running server, usually on port 8880.

The return value from this invocation will be either 0, a positive number, or an exception. 0 indicates that the operation completed successfully, with no warnings. A positive number indicates that the operation completed successfully, with the number of warnings. An exception indicates that the test of the connection failed.

You can find an example of this code here: Example: Test a connection using testConnection(ConfigID).

Testing a connection with the administrative console:

After you have defined and saved a data source, you can click the **Test Connection** button to ensure that the parameters in the data source definition are correct. On

the collection panel, you can select multiple data sources and test them all at once. Be sure that the following conditions are met before using the Test Connection button.

1. Depending on your specific needs, a valid *Authentication Data alias* may need to exist and be supplied on the data source panels.
2. If you are testing a connection using a WebSphere Application Server Version 4.0 type of data source, ensure that the *user* and *password* information is filled in.
3. If you used a WebSphere Environment entry for the classpath or other fields, such as *${DB2390_JDBC_DRIVER_PATH}/classes/db2j2classes.zip*, make sure that you assign it a value in the *Manage WebSphere Variables* page.
4. Click **Test Connection**.

   A Test Connection operation can have three different outcomes, each resulting in a different message being displayed in the messages panel of the page on which you press the Test Connection button.
   a. The test can complete successfully, meaning that a connection is successfully obtained to the database using the configured data source parameters. The resulting message states: Test Connection for DataSource *DataSourceName* on process *ProcessName* at node *NodeName* was successful.
   b. The test can complete successfully with warnings. This means that while a connection is successfully obtained to the database, warnings were issued. The resulting message states: Test Connection for DataSource *DataSourceName* on process *ProcessName* at node *NodeName* was successful with warning(s).View the JVM Logs for more details.

      The **View the JVM Logs** text is a hyperlink that takes you to the JVM Logs console screen for the process.
   c. The test can fail. A connection to the database with the configured parameters is not obtained. The resulting message states: Test Connection failed for DataSource *DataSourceName* on process *ProcessName* at node *NodeName* with the following exception: *ExceptionText*. View the JVM Logs for more details.

      Again, the text for **View the JVM Logs** is a hyperlink to the appropriate logs screen.

Testing a connection using wsadmin:

The *AdminControl* object of wsadmin has a testConnection operation that tests the configuration properties of a data source object. It takes a *configuration ID* as an argument.

Note: This invocation style is currently supported only for databases that *do not* require a user ID and password to make a connection, such as DB2 on a Windows NT machine.

1. Invoke the *getid* method for your data source.
2. Set the value of the *configuration id* to a variable.

   ```
   set myds [$AdminConfig getid /JDBCProvider:mydriver/DataSource:mydatasrc/]
   ```

   where */JDBCProvider:mydriver/DataSource:mydatasrc/* is the data source you want to test. After you have the configuration ID of the data source, you can test the connection to the database.
3. Test the connection to the database.

   ```
   $AdminControl testConnection $myds
   ```

Example: Test a connection using country and language (properties): It is possible to invoke the same test connection operation on the DataSourceCfgHelper MBean from a Java program that wsadmin uses, passing in the properties you wish to test.

```
import java.util.*;
import javax.sql.DataSource;
import javax.transaction.*;
import javax.management.*;

import com.ibm.websphere.management.*;
import com.ibm.websphere.management.configservice.*;
import com.ibm.ws.exception.WsException;
import com.ibm.websphere.rsadapter.DSPropertyEntry;

/**
 * Tests a connection to a DataSource when WebSphere
 * Security is disabled
 *
 * To run this example, the following must be done:
 *
 * 1)  Set the WAS_HOME environment variable to the location of
 *     your WebSphere Application Server for z/OS Configuration
 *     directory
 *
 *     Example:  export WAS_HOME=/WebSphereV5R1M0/AppServer
 *
 * 2)  Set the following environment variables:
 *
 *     export WAS_LIB=$WAS_HOME/lib
 *     export WAS_CLASSPATH=[DIRECTORY_CONTAINING_THIS_FILE]
 *     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_LIB/jmxc.jar
 *     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_LIB/wsexception.jar
 *     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_LIB/admin.jar
 *     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_LIB/wasjmx.jar
 *     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_HOME/java/jre/lib/ext/mail.jar
 *     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_LIB/ibmjlog.jar
 *     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_LIB/utils.jar
 *
 * 3)  Execute the following commands:
 *
 *     javac -classpath $WAS_CLASSPATH TestDS2.java
 *     java -classpath $WAS_CLASSPATH TestDS2
 */
public class TestDS2 {

    String port = "8880";
    String host = "localhost";
    final static boolean verbose = true;

    /**
     * Main method.
     */
    public static void main(String[] args) {
        TestDS2 cds = new TestDS2();
        try {
            cds.run(args);
        } catch (com.ibm.ws.exception.WsException ex) {
            System.out.println("Caught this " + ex   );
            ex.printStackTrace();
        } catch (Exception ex) {
            System.out.println("Caught this " + ex );
            ex.printStackTrace();
        }
    }

    /**
```

```
 * This method tests the DataSourceCfgHelper Mbean.
 */
public void run(String[] args) throws Exception {
    try {
        System.out.println("Connecting to the application server.......");

        /*******************************************************************/
        /**    Initialize the AdminClient                                 */
        /*******************************************************************/
        Properties adminProps = new Properties();
        adminProps.setProperty(AdminClient.CONNECTOR_TYPE, AdminClient.CONNECTOR_TYPE_SOAP);
        adminProps.setProperty(AdminClient.CONNECTOR_HOST, host);
        adminProps.setProperty(AdminClient.CONNECTOR_PORT, port);
        AdminClient adminClient = null;
        try {
            adminClient = AdminClientFactory.createAdminClient(adminProps);
        } catch (com.ibm.websphere.management.exception.ConnectorException ce) {
            System.out.println("Cannot make a connection to the application server\n"+ce);
            System.exit(1);
        }

        /*******************************************************************/
        /**    Locate the Mbean                                           */
        /*******************************************************************/
        ObjectName handle = null;
        try {
            ObjectName queryName = new ObjectName("WebSphere:type=DataSourceCfgHelper,*");
            Set s = adminClient.queryNames(queryName, null);
            Iterator iter = s.iterator();
            if (iter.hasNext()) handle = (ObjectName)iter.next();
        } catch (MalformedObjectNameException mone) {
            System.out.println("Check the program variable queryName" + mone);
        } catch (com.ibm.websphere.management.exception.ConnectorException ce) {
            System.out.println("Cannot connect to the application server" + ce);
        }

        /*******************************************************************/
        /**    Call the Mbean to get the data source properties           */
        /*******************************************************************/
        String dsClassName = "com.ibm.db2.jcc.DB2ConnectionPoolDataSource";
        String providerClassPath = "/db2beta/db2710/classes/db2j2classes.zip";
        String[] signature = { "java.lang.String", "java.lang.String"};
        Object[] params = { dsClassName, providerClassPath};
        Object result = null;

        if (verbose) {
            System.out.println("Calling getPropertiesForDataSource()
  for " + dsClassName + "\n");
```

Note: The previous line of code (beginning `System.out.println`) must be entered on one line; it is split here for formatting purposes.

```
        }
        try {
            // get the properties
            result = adminClient.invoke(handle, "getPropertiesForDataSource",
  params, signature);
```

Note: The previous line of code must be entered on one line; it is split here for formatting purposes.

```
        } catch (MBeanException mbe) {
            if (verbose) {
                System.out.println("\tMbean Exception " + dsClassName);
            }
        } catch (InstanceNotFoundException infe) {
```

```
            System.out.println("Cannot find " + dsClassName);
        } catch (Exception ex) {
            System.out.println("Exception occurred calling getPropertiesForDataSource()
 for " + dsClassName + ex);
```

Note: The previous line of code (beginning System.out.println) must be entered on one line; it is split here for formatting purposes.

```
}

        // Pretty print what we found
        Iterator propIterator = ((List)result).iterator();
        System.out.println(format("Name",21)+ "|" + format("Default Value",34) + "|"
 + format("Type",17) +"|Reqd");
```

Note: The previous line of code (beginning System.out.println) must be entered on one line; it is split here for formatting purposes.

```
String line = "_____________________________________________________________________";
        System.out.println(line);
        while (propIterator.hasNext()) {
            DSPropertyEntry dspe = (DSPropertyEntry)propIterator.next();
            System.out.print(format(dspe.getPropertyName(),21)+"|"+
 format(dspe.getDefaultValue(),34) + "|");
```

Note: The previous line of code (beginning System.out.print(format...) must be entered on one line; it is split here for formatting purposes.

```
System.out.println(format(dspe.getPropertyType(),17) +"|"+
 ((dspe.isRequired())? " Y" : " N"));
```

Note: The previous line of code must be entered on one line; it is split here for formatting purposes.

```
}
        System.out.println(line);


        /*********************************************************************/
        /** Invoke the reload function from the AdminClient to pickup the    */
        /*  data source from the naming space.                               */
        /*********************************************************************/

        if (verbose) {
            System.out.println("Calling reload()");
        }
        try {
            result = adminClient.invoke(handle, "reload", new Object[] {}, new String[] {});
        } catch (MBeanException mbe) {
            if (verbose) {
                System.out.println("\tMbean Exception calling reload" + mbe);
            }
        } catch (InstanceNotFoundException infe) {
            System.out.println("Cannot find reload ");
        } catch (Exception ex) {
            System.out.println("Exception occurred calling reload()"  + ex);
        }
        if (result==null && verbose) {
            System.out.println("OK reload()");
        }

        /*********************************************************************/
        /**  Start to test the connection to the database                   */
        /*********************************************************************/

        if (verbose) {
            System.out.println("\nTesting connection to the database using " + dsClassName);
```

```
        }

        String user = "IBMUSER";
        String password = "IBMUSER";
        Properties props = new Properties();
        props.setProperty("databaseName", "LOC1");
        props.setProperty("dataStoreHelperClass",
    "com.ibm.websphere.rsadapter.DB2DataStoreHelper");
```

Note: The previous line of code must be entered on one line; it is split here for formatting purposes.

```
        Object result2 = null;

        String[] signature2 = { "java.lang.String", "java.lang.String", "java.lang.String",
    "java.util.Properties", "java.lang.String", "java.lang.String","java.lang.String" };
```

Note: The previous line of code must be entered on one line; it is split here for formatting purposes.

```
    Object[] params2 = { dsClassName, user, password,props ,providerClassPath, "EN", "US"};

        try {
            result2 = adminClient.invoke(handle, "testConnectionToDataSource",
     params2, signature2);
```

Note: The previous line of code must be entered on one line; it is split here for formatting purposes.

```
    } catch (MBeanException mbe) {
              if (verbose) {
                  System.out.println("\tMbean Exception " + dsClassName);
              }
        } catch (InstanceNotFoundException infe) {
            System.out.println("Cannot find " + dsClassName);
        } catch (RuntimeMBeanException rme) {
            Exception ex = rme.getTargetException();
            ex.printStackTrace(System.out);
            throw ex;
        } catch (Exception ex) {
            System.out.println("Exception occurred calling testConnectionToDataSource()
      for " + dsClassName + ex);
```

Note: The previous line of code (beginning `System.out.println`) must be entered on one line; it is split here for formatting purposes.

```
       ex.printStackTrace();
        }

        // the testConnectionToDataSource method doesn't always throw an exception
        // if the test fails; it only returns a String
    //showing the results, so we can't really know if the
        // test was successful
        //
        // the best option here is to just print the String for the user

        System.out.println("Test Result:  " + result2);

      } catch (RuntimeOperationsException roe) {
        Exception ex = roe.getTargetException();
        ex.printStackTrace(System.out);
        throw ex;
      } catch (Exception ex) {
        ex.printStackTrace(System.out);
        throw ex;
      }
    }
```

```
    /**
     * Format the string right justified in the space provided,
     * or truncate the string.
     *
     * @param in
     * @param length
     * @return
     */
    public String format(Object in, int length) {
        if (in ==null) {
            in = "-null-";
        }

        String ins = in.toString();
        int insLength = ins.length();
        if ( insLength > length) {
            return ins.substring(0,length);
        } else {
            StringBuffer sb = new StringBuffer(length);
            while (length - insLength > 0) {
                sb.append(" ");
                length--;
            }
            sb.append(ins);
            return sb.toString();
        }
    }
}
```

Example: Test a connection using testConnection(ConfigID): You can pass the *configuration ID* of a configured data source, rather than the properties of the data source. This program uses JMX to connect to a running server and invoke the *testConnection* method on the *DataSourceCfgHelper* MBean.

```
import java.util.Iterator;
import java.util.Locale;
import java.util.Properties;
import java.util.Set;

import javax.management.InstanceNotFoundException;
import javax.management.MBeanException;
import javax.management.MalformedObjectNameException;
import javax.management.ObjectName;
import javax.management.RuntimeMBeanException;
import javax.management.RuntimeOperationsException;

import com.ibm.websphere.management.AdminClient;
import com.ibm.websphere.management.AdminClientFactory;

/**
 * Tests a connection to a DataSource when WebSphere
 * Security is disabled
 *
 * To run this example, the following must be done:
 *
 * 1)  Set the WAS_HOME environment variable to the location of
 *     your WebSphere Application Server for z/OS Configuration
 *     directory
 *
 *     Example:  export WAS_HOME=/WebSphereV5R1M0/AppServer
 *
 * 2)  Set the following environment variables:
 *
 *     export WAS_LIB=$WAS_HOME/lib
 *     export WAS_CLASSPATH=[DIRECTORY_CONTAINING_THIS_FILE]
 *     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_LIB/jmxc.jar
```

```
 *     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_LIB/wsexception.jar
 *     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_LIB/admin.jar
 *     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_LIB/wasjmx.jar
 *     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_HOME/java/jre/lib/ext/mail.jar
 *     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_LIB/ibmjlog.jar
 *     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_LIB/utils.jar
 *
 * 3)  Execute the following commands:
 *
 *     javac -classpath $WAS_CLASSPATH TestDSGUI.java
 *     java -classpath $WAS_CLASSPATH TestDSGUI
 */
public class TestDSGUI {

    //Use port 8880 for base installation or port 8879 for ND installation
    String port = "8880";
    String host = "localhost";
    final static boolean verbose = true;

    // eg a configuration ID for 5.1 DataSource declared at the node level
    private static final String resURI =
  "cells/SY1/nodes/SY1:resources.xml#DataSource_1080685343915";
```

Note: The previous line of code must be entered on one line; it is split here for formatting purposes.

```
  // eg a configuration ID for 5.1 DataSource declared at the server level
   // private static final String resURI =
 //"cells/SY1/nodes/SY1/servers/server1/resources.xml#DataSource_6";


    public static void main(String[] args) {
        TestDSGUI cds = new TestDSGUI();
        cds.run(args);
    }

    /**
     * This method tests the ResourceMbean.
     */
    public void run(String[] args) {
        try {

            System.out.println("Connecting to the application server.......");

            /*****************************************************************/
            /**    Initialize the AdminClient                               */
            /*****************************************************************/

            Properties adminProps = new Properties();
            adminProps.setProperty(AdminClient.CONNECTOR_TYPE, AdminClient.CONNECTOR_TYPE_SOAP);
            adminProps.setProperty(AdminClient.CONNECTOR_HOST, host);
            adminProps.setProperty(AdminClient.CONNECTOR_PORT, port);
            AdminClient adminClient = null;
            try {
                adminClient = AdminClientFactory.createAdminClient(adminProps);
            } catch (com.ibm.websphere.management.exception.ConnectorException ce) {
                System.out.println("NLS: Cannot make a connection to the application server\n");
                ce.printStackTrace();
                System.exit(1);
            }

            /*****************************************************************/
            /**    Locate the Mbean                                         */
            /*****************************************************************/

            ObjectName handle = null;
            try {
```

```
            // Send in a locator string
            ObjectName queryName = new ObjectName("WebSphere:type=DataSourceCfgHelper,*");
            Set s = adminClient.queryNames(queryName, null);
            Iterator iter = s.iterator();
            while (iter.hasNext()) {
                // use the first MBean that is found
                handle = (ObjectName) iter.next();
                System.out.println("Found this ->" + handle);
            }
            if (handle == null) {
                System.out.println("NLS: Did not find this MBean>>" + queryName);
                System.exit(1);
            }
        } catch (MalformedObjectNameException mone) {
            System.out.println("Check the program variable queryName" + mone);
        } catch (com.ibm.websphere.management.exception.ConnectorException ce) {
            System.out.println("Cannot connect to the application server" + ce);
        }

        /*******************************************************************/
        /**           Build parameters to pass to Mbean                   */
        /*******************************************************************/

        String[] signature = { "java.lang.String" };
        Object[] params = { resURI };
        Object result = null;

        if (verbose) {
            System.out.println("\nTesting connection to the database using " + handle);
        }

        try {
            /*******************************************************************/
            /**  Start to test the connection to the database                  */
            /*******************************************************************/
            result = adminClient.invoke(handle, "testConnection", params, signature);
        } catch (MBeanException mbe) {
            // ****** all user exceptions come in here
            if (verbose) {
                Exception ex = mbe.getTargetException();
// The previous line is the real exception from the Mbean
                System.out.println("\nNLS:Mbean Exception was received contains " + ex);
                ex.printStackTrace();
                System.exit(1);
            }
        } catch (InstanceNotFoundException infe) {
            System.out.println("Cannot find " + infe);
        } catch (RuntimeMBeanException rme) {
            Exception ex = rme.getTargetException();
            ex.printStackTrace(System.out);
            throw ex;
        } catch (Exception ex) {
            System.out.println("\nUnexpected Exception occurred: " + ex);
            ex.printStackTrace();
        }

        /*******************************************************************/
        /**  Process the result.  The result will be the number of warnings */
        /**  issued.  A result of 0 indicates a successful connection with  */
        /**  no warnings.                                                   */
        /*******************************************************************/

        //A result of 0 indicates a successful connection with no warnings.
        System.out.println("Result (number of warnings) = " + result);

    } catch (RuntimeOperationsException roe) {
        Exception ex = roe.getTargetException();
```

```
            ex.printStackTrace(System.out);
        } catch (Exception ex) {
            System.out.println("General exception occurred");
            ex.printStackTrace(System.out);
        }
    }
}
```

Example: Migrating - Testing the DataSource object connection: The following examples demonstrate how to test the connection to a DataSource object in the WebSphere Application Server V4.0 and V5.x:

- wscp V4.0

  ```
  set myds /JDBCDriver:mydriver/DataSource:myds/
  DataSource testConnection $myds
  ```

- wsadmin V5.x

  The **testConnection** command is part of the AdminControl object because it is an operational command. This particular type of operational command takes a configuration ID as an argument, so you invoke the **getid** command on the AdminConfig object:

  Using Jacl:

  ```
  set myds [$AdminConfig getid /JDBCProvider:mydriver/DataSource:mydatasrc/]
  $AdminControl testConnection $myds
  ```

  5.1+ Using Jython:

  ```
  myds = AdminConfig.getid('/JDBCProvider:mydriver/DataSource:mydatasrc/')
  AdminControl.testConnection(myds)
  ```

  In many cases, a user ID and password, or other properties are required to complete the test connection. If this is the case, you receive the following message, which describes the missing properties:

  ```
  WASX7216E: 2 attributes required for testConnection are missing: "[user, password]"
  To complete this operation, please supply the missing attributes as an option,
  following this example: {{user user_val} {password password_val}}
  ```

  For this example, issue the following commands:

  Using Jacl:

  ```
  set myds [$AdminConfig getid /JDBCProvider:mydriver/DataSource:mydatasrc/]
  $AdminControl testConnection $myds {{user myuser} {password secret}}
  ```

  5.1+ Using Jython:

  ```
  myds = AdminConfig.getid('/JDBCProvider:mydriver/DataSource:mydatasrc/')
  AdminControl.testConnection(myds, [['user', 'myuser'], ['password', 'secret']])
  ```

Configuring Java 2 Connector connection factories in the administrative console

1. Click **Resources**.
2. Click **Resource Adapters**.
3. Select a resource adapter under Resource Adapters.
4. Click **J2C Connection Factories** under Additional Properties .
5. Click **New**.
6. Specify *General Properties* .
7. Select the authentication preference.
8. Select aliases for **component-managed authentication**, **container-managed authentication**, or both. If none are available, or you want to define a different one, click **Apply** > **J2C Authentication Data Entries** under Related Items.
   a. Click **J2C Auth Data Entries** under Related Items.

b. Click **New**.
c. Specify General Properties.
d. Click **OK**.

9. Click **OK**.
10. Click the J2C connection factory you just created.
11. Under *Additional Properties* click **Connection Pool**.
12. Change any values desired by clicking the property name.
13. Click **OK**.
14. Click **Custom Properties** under *Additional Properties*.
15. Click any property name to change its value. Note that **UserName** and **Password** if present, are overridden by the **component-managed authentication** alias you specified in a previous step.
16. Click **Save**.

Connection pool settings

Use this page to configure connection pool settings.

This administrative console page is common to a range of resource types; for example, JDBC data sources and JMS queue connection factories. To view this page, the path depends on the type of resource, but generally you select an instance of the resource provider, then an instance of the resource type, then click **Connection Pool**. For example: click **Resources** > **JDBC Providers** > *JDBC_provider* > **Data Sources** > *data_source* > **Connection Pool**.

Connection Timeout:

Specifies the interval, in seconds, after which a connection request times out and a ConnectionWaitTimeoutException is thrown.

The wait is necessary when the maximum value of connections (*Max Connections*) to a particular connection pool is reached . For example, if *Connection Timeout* is set to 300 and the maximum number of connections is reached, the Pool Manager waits for 300 seconds for an available physical connection. If a physical connection is *not* available within this time, the Pool Manager throws a ConnectionWaitTimeoutException. It usually does not make sense to retry the *getConnection()* method, because if a longer wait time is required, you should set the *Connection Timeout* setting to a higher value. Therefore, if this exception is caught by the application, the administrator should review the expected usage of the application and tune the connection pool and the database accordingly.

If Connection Timeout is set to 0, the Pool Manager waits as long as necessary until a connection is allocated (which happens when the number of connections falls below the value of *Max Connections*).

If Max Connections is set to 0, which enables an infinite number of physical connections, then the Connection Timeout value is ignored.

Data type	Integer
Units	Seconds
Default	180
Range	0 to max int

Max Connections:

Specifies the maximum number of physical connections that you can create in this pool.

These are the physical connections to the backend resource. Once this number is reached, no new physical connections are created and the requester waits until a physical connection that is currently in use returns to the pool, or a ConnectionWaitTimeoutException is thrown.

For example, if the Max Connections value is set to 5, and there are five physical connections in use, the pool manager waits for the amount of time specified in Connection Timeout for a physical connection to become free.

If Max Connections is set to 0, the Connection Timeout value is ignored.

For better performance, set the value for the connection pool lower than the value for the Max Connections option in the Web container. Lower settings, such as 10-30 connections, perform better than higher settings, such as 100.

If clones are used, one data pool exists for each clone. Knowing the number of data pools is important when configuring the database maximum connections.

Min Connections:

Specifies the minimum number of physical connections to maintain.

Until this number is reached, the pool maintenance thread does not discard physical connections. However, no attempt is made to bring the number of connections up to this number. If you set a value for Aged Timeout, the minimum is not maintained. All connections with an expired age are discarded.

For example if the Min Connections value is set to 3, and one physical connection is created, the Unused Timeout thread does not discard that connection. By the same token, the thread does not automatically create two additional physical connections to reach the Min Connections setting.

Data type	Integer
Default	1
Range	0 to max int

Reap Time:

Specifies the interval, in seconds, between runs of the pool maintenance thread.

For example, if Reap Time is set to 60, the pool maintenance thread runs every 60 seconds. The *Reap Time* interval affects the accuracy of the *Unused Timeout* and *Aged Timeout* settings. The smaller the interval, the greater the accuracy. If the pool maintenance thread is enabled, set the Reap Time value less than the values of Unused Timeout and Aged Timeout. When the pool maintenance thread runs, it discards any connections remaining unused for longer than the time value specified in Unused Timeout, until it reaches the number of connections specified in *Min Connections*. The pool maintenance thread also discards any connections that remain active longer than the time value specified in Aged Timeout.

The Reap Time interval also affects performance. Smaller intervals mean that the pool maintenance thread runs more often and degrades performance.

To disable the pool maintenance thread set Reap Time to 0, or set both Unused Timeout and Aged Timeout to 0. The recommended way to disable the pool maintenance thread is to set Reap Time to 0, in which case Unused Timeout and Aged Timeout are ignored. However, if Unused Timeout and Aged Timeout are set to 0, the pool maintenance thread runs, but only physical connections which timeout due to non-zero timeout values are discarded.

Data type	Integer
Units	Seconds
Default	180
Range	0 to max int

Unused Timeout:

Specifies the interval in seconds after which an unused or idle connection is discarded.

Set the Unused Timeout value higher than the Reap Timeout value for optimal performance. Unused physical connections are only discarded if the current number of connections not in use exceeds the Min Connections setting. For example, if the unused timeout value is set to 120, and the pool maintenance thread is enabled (Reap Time is not 0), any physical connection that remains unused for two minutes is discarded. Note that accuracy of this timeout, as well as performance, is affected by the Reap Time value. See Reap Time for more information.

Data type	Integer
Units	Seconds
Default	1800
Range	0 to max int

Aged Timeout:

Specifies the interval in seconds before a physical connection is discarded.

Setting *Aged Timeout* to 0 supports active physical connections remaining in the pool indefinitely. Set the Aged Timeout value higher than the Reap Timeout value for optimal performance. For example, if the Aged Timeout value is set to 1200, and the Reap Time value is not 0, any physical connection that remains in existence for 1200 seconds (20 minutes) is discarded from the pool. Note that accuracy of this timeout, as well as performance, are affected by the Reap Time value. See Reap Time for more information.

Data type	Integer
Units	Seconds
Default	0
Range	0 to max int

Purge Policy:

Specifies how to purge connections when a *stale connection* or *fatal connection error* is detected.

Valid values are **EntirePool** and **FailingConnectionOnly**. JCA data sources can have either option. WebSphere Version 4.0 data sources always have a purge policy of **EntirePool**.

Data type
String

Default
EntirePool

Range

EntirePool
All connections in the pool are marked stale. Any connection not in use is immediately closed. A connection in use is closed and throws a *StaleConnectionException* during the next operation on that connection. Subsequent *getConnection* requests from the application result in new connections to the database opening. When using this purge policy, there is a slight possibility that some connections in the pool are closed unnecessarily when they are not stale. However, this is a rare occurrence. In most cases, a purge policy of EntirePool is the best choice.

FailingConnectionOnly
Only the connection that caused the *StaleConnectionException* is closed. Although this setting eliminates the possibility that valid connections are closed unnecessarily, it makes recovery from an application perspective more complicated. Because only the currently failing connection is closed, there is a good possibility that the next *getConnection* request from the application can return a connection from the pool that is also stale, resulting in more stale connection exceptions.

The Connection pretest function attempts to insulate an application from pooled connections that are not valid. When a backend resource, such as a database, goes down,, pooled connections that are not valid might exist in the free pool. This is especially true when the purge policy is failingConnectionOnly; in this case, the failing connection is removed from the pool. Depending on the failure, the remaining connections in the pool might not be valid.

Connection pool (Version 4) settings

Use this page to create a connection pool for a Version 4.0 data source.

To view this administrative console page, click **Resources** > **JDBC Providers** > *JDBC_provider* > **Data Sources (Version 4)** > *data_source* > **Connection Pool**.

Scope:

Specifies the level to which this resource definition is visible -- the cell, node, or server level.

Resources such as JDBC Providers, Namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

Note that no matter what the scope of a defined resource, the resource's properties only apply at an individual server level. For example, if you define the scope of a data source at the Cell level, all users in that Cell can look up and use that data source, which is unique within that Cell. However, resource property settings are local to each server in the Cell. For example, if you define *max connections* to 10, then each server in that Cell can have 10 connections.

Cell The most general scope. Resources defined at the Cell scope are visible from all Nodes and servers, unless they are overridden. To view resources defined in the cell scope, do not specify a server or a node name in the scope selection form.

Node The default scope for most resource types. Resources defined at the Node scope override any duplicates defined at the Cell scope and are visible to all servers on the same node, unless they are overridden at a server scope on that node. To view resources defined in a node scope, do not specify a server, but select a node name in the scope selection form.

Server The most specific scope for defining resources. Resources defined at the Server scope override any duplicate resource definitions defined at the Cell scope or parent Node scope and are visible only to a specific server. To view resources defined in a server scope, specify a server name as well as a node name in the scope selection form.

When resources are created, they are always created into the current scope selected in the panel. To view resources in other scopes, specify a different node or server in the scope selection form.

Data type String

Minimum Pool Size:

Specifies the minimum number of connections to maintain in the pool.

The minimum pool size can affect the performance of an application. Smaller pools require less overhead when the demand is low because fewer connections are held open to the database. When the demand is high, the first applications experience a slow response because new connections are created if all others in the pool are in use.

Data type Integer
Default 1
Range Any non-negative integer.

Maximum Pool Size:

Specifies the maximum number of connections to maintain in the pool.

If the maximum number of connections is reached and all connections are in use, additional requests for a connection wait up to the number of seconds specified as the connection timeout. The maximum pool size can affect the performance of an application. Larger pools require more overhead when demand is high because there are more connections open to the database at peak demand. These connections persist until idled out of the pool. If the maximum value is smaller, longer wait times or possible connection timeout errors during peak times can occur. Ensure that the database can support the maximum number of connections in the application server, in addition to any load that it has outside of the application server.

Data type	Integer
Default	10
Range	Any positive integer

Connection Timeout:

Specifies the maximum number of seconds an application waits for a connection from the pool before timing out and throwing a ConnectionWaitTimeoutException to the application.

Setting this value to 0 disables the connection timeout.

Data type	Integer
Units	Seconds
Default	180
Range	Any non-negative integer

Idle Timeout:

Specifies the maximum number of seconds that an idle (unallocated) connection can remain in the pool before being removed to free resources.

Connections need to idle out of the pool because keeping connections open to the database can cause database memory problems. However, not all connections are idled out of the pool, even if they are older than the Idle Timeout setting. A connection is not idled if removing the connection would cause the pool to shrink below its minimum size. Setting this value to 0 disables the idle timeout.

Data type	Integer
Units	Seconds
Default	1800
Range	Any non-negative integer

Orphan Timeout:

Specifies the maximum number of seconds that an application can hold a connection without using it before the connection returns to the pool

If there is no activity on an allocated connection for longer than the Orphan Timeout setting, the connection is marked for orphaning. After another Orphan Timeout number of seconds, if the connection still has no activity, the connection

returns to the pool. If the application tries to use the connection again, it is thrown a StaleConnectionException. Connections that are enlisted in a transaction are not orphaned. Setting this value to 0 disables the orphan timeout.

Data type	Integer
Units	Seconds
Default	1800
Range	Any non-negative integer

Statement Cache Size:

Specifies the number of cached prepared statements to keep per connection.

The largest value you would need to set your cache size to if you do not want any cache discards is determined as follows: for each application that uses this data source on a particular server, add up the number of unique prepared statements (as determined by the *sql* string, concurrency, and the scroll type). This is the maximum number of possible prepared statements that can be cached on a given connection over the life of the server. Setting the cache size to this value means you never have cache discards. This provides better performance. However, because of potential resource limitations, this might not always be possible.

Data type	Integer
Default	10
Range	Any non-negative integer

Auto Connection Cleanup:

Specifies whether or not the connection pooling software automatically closes connections from this data source at the end of a transaction.

The default is *false*, which indicates that when a transaction completes, WebSphere Application Server closes the connection and returns it to the pool. Any use of the connection after the transaction has ended results in a StaleConnectionException because the connection is closed and has returned to the pool. This mechanism ensures that connections are not held indefinitely by the application. If the value is set to *true*, the connection is not returned to the pool at the end of a transaction. In this case, the application must return the connection to the pool by calling *close()*. If the application does not close the connection, the pool can run out of connections for other applications to use.

Data type	Check box
Default	False (clear)

Configuring connection factories for resource adapters within applications

1. Click **Applications**.
2. Click **Install New Application**.
3. Browse to find the appropriate EAR file, which contains an RAR file.
4. Click **Next**.
5. Select **resource ref mapping to a J2C Connection Factory**, then click **Next**.
6. After the application installs, click **Applications**.
7. Select the application just installed.

8. Click **Connector Modules** under Related Items.
9. Select an RAR file name on the Connector Modules page.
10. Click **Resource Adapter** under Additional Properties.
11. Click **J2C Connection Factories** under Additional Properties.
12. Click **New**.
13. Specify General Properties.
14. Select the authentication preference.
15. Select aliases for **component-managed authentication**, **container-managed authentication**, or both. If none are available, or you want to define a different one, click **Apply** > **J2C Authentication Data Entries** under Related Items.
    a. Click **J2C Auth Data Entries** under Related Items.
    b. Click **New**.
    c. Specify General Properties.
    d. Click **OK**.
16. Click **OK**.
17. Click the J2C connection factory you just created.
18. Click **Connection Pool** under Additional Properties .
19. Change any values desired by clicking on the property name.
20. Click **OK**.
21. Click **Custom Properties** under Additional Properties.
22. Click any property name to change its value. Note that **UserName** and **Password** if present, are overridden by the **component-managed authentication** alias you specified in a previous step.
23. Click **Save**.

J2C Connection Factories collection

Use this page to select a connection factory, which represents one set of connection configuration values.

Application components such as enterprise beans have resource reference descriptors that refer to the connection factory, not the resource adapter. The connection factory is really a configuration properties list holder. In addition to the arbitrary set of configuration properties defined by the vendor of the resource adapter, there are several standard configuration properties that apply to the connection factory. These standard properties are used by the Java 2 Connectors connection pool manager in the application server run time and are not known by the vendor supplied resource adapter code.

To view this administrative console page, click **Resources** > **Resource Adapters** > *resource_adapter* > **J2C Connection Factories**.

Name:

Specifies a list of the connection factory display names.

Data type String

JNDI name:

Specifies the Java Naming and Directory Interface (JNDI) name of this connection factory.

Data type String

Description:

Specifies a text description of this connection factory.

Data type String

Category:

Specifies a string that you can use to classify or group this connection factory.

Data type String

J2C Connection Factories settings:

Use this page to specify settings for a connection factory.

To view this administrative console page, click **Resources** > **Resource Adapters** > *resource_adapter* > **J2C Connection Factories** > *connection_factory*.

Scope:

Specifies the level to which this resource definition is visible -- the cell, node, or server level.

Resources such as JDBC Providers, Namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

Note that no matter what the scope of a defined resource, the resource's properties only apply at an individual server level. For example, if you define the scope of a data source at the Cell level, all users in that Cell can look up and use that data source, which is unique within that Cell. However, resource property settings are local to each server in the Cell. For example, if you define *max connections* to 10, then each server in that Cell can have 10 connections.

Cell The most general scope. Resources defined at the Cell scope are visible from all Nodes and servers, unless they are overridden. To view resources defined in the cell scope, do not specify a server or a node name in the scope selection form.

Node The default scope for most resource types. Resources defined at the Node scope override any duplicates defined at the Cell scope and are visible to all servers on the same node, unless they are overridden at a server scope on that node. To view resources defined in a node scope, do not specify a server, but select a node name in the scope selection form.

Server The most specific scope for defining resources. Resources defined at the Server scope override any duplicate resource definitions defined at the Cell scope or parent Node scope and are visible only to a specific server. To view resources defined in a server scope, specify a server name as well as a node name in the scope selection form.

When resources are created, they are always created into the current scope selected in the panel. To view resources in other scopes, specify a different node or server in the scope selection form.

Data type String

Name:

Specifies a list of connection factory display names.

Data type String

JNDI Name:

Specifies the JNDI name of this connection factory.

For example, the name could be *eis/myECIConnection*.

After you set this value, save it and restart the server. You can see this string when you run *dumpNameSpace*.

Data type String
Default eis/*display name*

Description:

Specifies a text description of this connection factory.

Data type String

Category:

Specifies a string that you can use to classify or group this connection factory.

Data type String

Authentication Preference:

Specifies the authentication mechanisms defined for this connection factory.

This setting specifies which of the authentication mechanisms defined for the corresponding resource adapter applies to this connection factory. Common values, depending on the capabilities of the resource adapter, are: *KERBEROS*, *BASIC_PASSWORD*, and *None*.

If None is chosen, the application component is expected to manage authentication (<res-auth>Application</res-auth>). In this case, the user ID and password are taken from one of the following:
- The component-managed authentication alias
- UserName, Password Custom Properties
- Strings passed on the getConnection method

For example, if two authentication mechanism entries are defined for a resource adapter in the *ra.xml* document:
- <authentication-mechanism-type>BasicPassword</authentication-mechanism-type>
- <authentication-mechanism-type>Kerbv5</authentication-mechanism-type>

the authentication preference specifies the mechanism to use for container-managed authentication. An exception is thrown during server startup if a mechanism that is not supported by the resource adapter is selected.

Data type Pick-list
Default BASIC_PASSWORD

Component-managed Authentication Alias:

Specifies authentication data for component-managed signon to the resource.

SecurityJAAS ConfigurationJ2C Authentication Data

To define a new alias not already appearing in the pick list:
- Click **Apply** to expose Related Items.
- Click **J2C Authentication Data Entries**.
- Define an alias.
- Click the connection factory name at the top of the *J2C Authentication Data Entries* page to return to the connection factory page.
- Select the alias.

Data type Pick-list

Container-managed Authentication Alias:

Specifies authentication data (a string path converted to userid and password) for container-managed signon to the resource.

SecurityJAAS ConfigurationJ2C Authentication Data

To define a new alias not already appearing in the pick list:
- Click **Apply** to expose Related Items.
- Click **J2C Authentication Data Entries**.
- Define an alias.
- Click the connection factory name at the top of the *J2C Authentication Data Entries* page to return to the connection factory page.
- Select the alias.

Data type Pick-list

Mapping-Configuration Alias:

Allows users to select from the **Security** > **JAAS Configuration** > **Application Logins Configuration** list.

The **DefaultPrincipalMapping** JAAS configuration maps the authentication alias to the userid and password. You may define and use other mapping configurations.

Data type Pick-list

Connection factory JNDI name tips

Distributed computing environments often employ naming and directory services to obtain shared components and resources. Naming and directory services associate names with locations, services, information, and resources.

Naming services provide name-to-object mappings. Directory services provide information on objects and the search tools required to locate those objects. There are many naming and directory service implementations, and the interfaces to them vary.

Java Naming and Directory Interface (JNDI) provides a common interface that is used to access the various naming and directory services. After you have set this value, saved it, and restarted the server, you should be able to see this string when you run *dumpnamespace*.

For WebSphere Application Server specifically, when you create a data source the default JNDI name is set to *jdbc/data_source_name*. When you create a connection factory, its default name is *eis/j2c_connection_factory_name*. You can, of course, override these values by specifying your own.

In addition, if you click the checkbox *Use this data source for container managed persistence (CMP)* when you create the data source, another reference is created with the name of *eis/jndi_name_of_datasource_CMP*. For example, if a data source has a JNDI name of *jdbc/myDatasource*, the CMP JNDI name is *eis/jdbc/myDatasource_CMP*. This name is used internally by CMP and is provided simply for informational purposes.

When creating a connection factory or data source, a JNDI name is given by which the connection factory or data source can be looked up by a component. Generally an "indirect" name with the *java:comp/env* prefix should be used. This makes any resource-reference data associated with the application available to the connection management runtime, to better manage resources based on the res-auth, res-isolation-level, res-sharing-scope, and res-resolution-control settings.

While the use of a direct JNDI name is supported, such use results in default values of these resource-ref data. You will see an informational message logged such as this:

```
J2CA0122I: Resource reference abc/myCF could not be located, so default values
of the following are used: [Resource-ref settings]
             res-auth:                 1 (APPLICATION)
             res-isolation-level:      0 (TRANSACTION_NONE)
             res-sharing-scope:        true (SHAREABLE)
             res-resolution-control:   999 (undefined)
```

Recreating database tables from the exported table data definition language

When an EAR file deploys the target database is selected. Then an appropriate *Table.ddl* file is created that contains the SQL statements to create the table for the bean. To create your table, using the *Table.ddl* file for DB2:

1. The container-managed bean (CMP) enterprise bean JAR file has a *Table.ddl* file that the Assembly Toolkit (ATK) generates. Extract the *Table.ddl* file to a working directory such as *C:\temp*.
2. To create your tables using SPUFI, extract the `Table.ddl` file located in your EJB JAR, to a temporary directory on your work station.
3. Transfer this `Table.ddl` file to a data set on your z/OS system.
4. Specify the data set as the input data set to SPUFI.

Security of lookups with component managed authentication

Only the J2EE application client can access Java 2 Connector (J2C) connection factories, datasources, or JMS queues. WebSphere Application Server does not support accessing these resources from other external clients.

For instructions on how to access these resources through the J2EE Application client, see Data sources for application clients.

Any client running in the WebSphere Application Server process (such as a Servlet or an enterprise bean) within the same cell that can look up a resource in the JNDI namespace can obtain connections without providing authentication data. It is important to note that J2C authentication alias is per **cell**. An enterprise bean or Servlet in one application server cannot look up a resource in another server process which is in a different cell, because the alias would not be resolved.

Disabling lookup security

By default, all lookups are secure as described in the topic Security of lookups with component managed authentication.

Although it is **not recommended**, it is possible to turn off the secure mode for a particular datasource or connection factory.
Edit *%WAS_HOME%\properties\j2c.properties* (or *$WAS_HOME/properties/j2c.properties* on UNIX or z/OS platforms).

Change this:

```
<!-- The security-properties are in a comment block.  Uncomment to use -->
<!--
   <security-properties connectionFactoryJNDIName="myDataSource">
       <secureMode>false</secureMode>
   </security-properties>
-->
```

to this, for example:

```
<!-- The security-properties are in a comment block.  Uncomment to use -->


  <security-properties connectionFactoryJNDIName="myDataSource">
      <secureMode>false</secureMode>
   </security-properties>
```

Where ″myDataSource″ is the JNDI name of the datasource or connection factory you want to run unsecure.

Configuring data access for application clients

Configuring data access for application clients involves specifying the resource reference and associated database information required for data access. This specification is done as part of the assembly and deployment steps for the application client.

There are two essential tools needed to configure data sources used by J2EE Application Clients: the Assembly Toolkit for defining the resource reference in the deployment descriptor, and the Application Client Resource Configuration Tool (ACRCT) for defining the connection to the database in the client deployment environment.

Data access from an application client uses the JDBC drive connection functions directly from the client side. It does not take advantage of the additional pooling support available in the WebSphere Application Server run time. Configuring data access for an application client does not require configuration of a JDBC provider and data source on the WebSphere Application Server server machine.

If you want to take advantage of the pooling and additional database functions provided by WebSphere Application Server, it is recommended that your client application utilize an enterprise bean running on the server side to perform data access.

Defining an application client resource reference in the Assembly Toolkit

1. Assemble your application client module as described in ″Assembling application clients.″
2. Create a new resource reference:
   a. In a J2EE Hierarchy view, right-click your application client module and click **Open With > Deployment Descriptor Editor**.
   b. On the **References** tab, click **Add > Resource Reference > Next**.
   c. On the Add Resource Reference page, enter the **Name** of this resource reference. The WebSphere Application client run time uses this name for two purposes: to bind the object into the *java:comp/env* portion of the JNDI namespace, and to find client specific configuration information. If the code for the application client performs a lookup for *java:comp/env/jdbc/myDB*, the name of the resource reference should be *jdbc/myDB*.
   d. For **Type**, select *javax.sql.DataSource* for JDBC connections.
   e. For **Authentication**, select *Application* if your client application intends to provide authentication information. If the WebSphere Application Client run time provides the authentication information (as configured by the Application Client Resource Configuration tool), select *Container*.
   f. Ignore the **Sharing scope** setting; it is unused in an application client resource reference. All WebSphere Application Client resources are currently unsharable.
   g. Close the deployment descriptor and save your changes.

   The JNDI name field appears under WebSphere Bindings after your add the reference.

Client configuration with the ACRCT

There are two client resources for you to configure in the Application Client Resource Configuration Tool (ACRCT) to enable data access from an application client: a data source provider and a data source.

Notes

Note: The following WebSphere objects, which can be bound into the server namespace, are not supported on the client:
- Java 2 Connector (J2C) objects
- Connection manager objects

The WebSphere Application Server Client does not provide client database drivers. If your client application uses a database directly, rather than using an enterprise bean, you must provide the database drivers on the client

machine. This action can involve contacting your database vendor to acquire client database driver code and licenses.

Instead of accessing the database directly, it is recommended that your client application use an enterprise bean. Accessing a database through an enterprise bean eliminates the need to have database drivers on the client machine because the database access is handled by the enterprise bean running on the WebSphere Application Server. Enterprise beans can also take advantage of the additional database functions provided by the WebSphere Application Server run time.

1. Configure a new data source provider as described in Configuring new data source providers. This provider describes the JDBC database implementation for your client application.
2. Enter the following information on the General Tab:
   a. A **name** for this data source provider.
   b. A **description** (this is optional).
   c. The **classpath** to the data source provider implementation classes or JAR files. This is optional if the implementation classes or JAR files are already in the classpath configuration of the client.
   d. The name of the **implementation class**. For example, for DB2 this value is *COM.ibm.db2.jdbc.DB2DataSource*. Remember this class must implement the *javax.sql.DataSource* class. The ACRCT does not verify this class and you receive an error when you run your client application if the class does not implement *javax.sql.DataSource*.

   Use the *Custom Tab* to configure non-standard properties of the data source provider. This panel enables you to enter property-value pairs. During run time the *implementation classname* is created and any custom properties added on this panel are set on the newly created data source object using reflection. Any properties configured on this panel must have an appropriate set method on the data source class. For example, assume there is a property called *use2Phase* and its value should be 1. On the custom panel you enter the value *use2Phase* into the **name** column and the value *1* into the **value** column. The WebSphere Application Client run time then uses reflection to find a property on the data source class called, typically *setUse2Phase* and call that method passing the value of 1. See your database product documentation for valid properties on your data source implementation.
3. Click **OK**.
4. Configure a new data source as described in Configuring new data sources for application clients. This describes the client properties of the database your client application uses.
5. Enter the following information on the General Tab:
   a. A **Name**. This field is required and identifies a name for the Application Client Resource Configuration Tool to use. This name is **not** used by your client application program.
   b. A **description** (this field is optional).
   c. The **JNDI name**. This field is required and must match the value entered in the **Name** field on the Add Resource Reference page of the Assembly Toolkit. In the example above, set this value to *jdbc/myDB*.
   d. The **Database Name** (this field is optional).
   e. Your *userid* in the **User** field. This field is optional.
   f. Your *password* in the **Password** field. This password does not display. This field is optional.

g. Your password again to confirm in the **Re-Enter password** field. Note: The **User** and **Password** fields are only used when the **Authentication** field on the Add Resource Reference page of the Assembly Toolkit is set to *Container*.

Configuring Cloudscape Version 5.1

Note: Datasources defined under the DB2 for z/OS Local JDBC Provider (RRS), must not be accessed from within the same JVM as datasources defined under any of the JDBC providers using the DB2 (version 8) Universal JDBC Driver. This means that the datasources must not be used from within the same server, and that similar considerations must be taken for client datasource usage.

In particular, datasources defined under the Cloudscape (5.1) Network Server Using Universal JDBC Driver Provider must not be used from the same server as datasources defined under the DB2 for z/OS Local JDBC Provider (RRS).

The reason for this restriction is that the DB2 Universal JDBC Driver uses the same package and class names (com.ibm.db2.jcc.DB2ConnectionPoolDataSource) as the DB2 for z/OS Local JDBC Provider (RRS). To understand this design, consider that the DB2 Universal JDBC Driver is the DB2 V8-level replacement for the DB2 V7-level JDBC driver used by the DB2 for z/OS Local JDBC Provider (RRS).

Cloudscape Version 5.1 provides the following two separate frameworks for running Cloudscape with WebSphere Application Server:

- Embedded: This framework is the same as the one for Cloudscape Version 5.0. To use this framework, define a Cloudscape JDBC provider. See the Cloudscape section in the minimum required settings article for more information.

  You must use the embedded framework if you are running XA. Cloudscape does not support XA on Network Server.
- Network Server: This framework is a new feature in Cloudscape Version 5.1, and removes these limitations that existed in earlier versions of Cloudscape:
  - inability to access a remote Cloudscape instance
  - only one JVM can boot up the same database instance

  The following steps describe how to configure and run the Network Server framework.

1. Start the Network Server on the machine that hosts the database instance.

   To start the Network Server, run the **startNetworkServer.bat** file, which is located in the `WAS_HOME/cloudscape51/bin/networkserver` directory. On UNIX platforms, the file is **startNetwokServer.sh**.
2. Update the **db2j.properties** file, which is located in the `WAS_HOME/cloudscape` directory, if necessary.

   Cloudscape should work without any modifications to this file.

   Use the entries in the **db2j.properties** file to turn on trace, change the port number on which Network Server listens, and enable other functions of the Network Server framework. The default port number on which the Network Server listens is port 1527.

   For more information on this file, see the Cloudscape documentation at www.ibm.com/software/data/cloudscape/pubs/collateral.html.

3. Define a Cloudscape Network Server using Universal JDBC driver to connect Cloudscape Version 5.1 with WebSphere Application Server using the Network Server framework.
4. Stop the Network Server by invoking the **stopNetworkServer.bat** file.

   You can find this file in the `WAS_HOME/cloudscape/bin/networkserver` directory. On UNIX platforms, the file is **stopNetworkServer.sh**.
5. Review additional tools available in the Network Server framework.

   Find these tools in the `WAS_HOME/cloudscape/bin/networkserver` directory. These tools are:
   - `sysinfo`
   - `cview`
   - `ij`
   - `dropSYSIBM` Use this tool to drop the SYSIBM schema and its contents.
6. Create a SYSIBM schema.

   If you do not create the SYSIBM schema, you cannot see the datatypes when you create tables using the `cview` graphical user interface. The **db2j.drda.loadSYSIBM** property in the **db2j.properties** file controls whether the schema is created on the first connection to the database. The **db2j.drda.loadSYSIBM** property default value is true.

   Note: You cannot run a new version and an older version of Cloudscape in the same WebSphere Application Server environment. For datasources created under the old version of Cloudscape, you must change the classpath in these datasources to point to the new version of Cloudscape; otherwise, the jar files from both versions of Cloudscape are loaded in the same environment. This might result in undesirable consequences or exception situations.

   Note: When you run ij, surround the dbname by double quotation marks (″ ″) if it includes the full path name; for example: ij> connect ′″c:temp;create=true″′

   This is ′ ″ ″ ′ without spaces.

Choosing which Cloudscape version to use

Occasionally, WebSphere Application Server offers updates to its software in the form of *fixpacks*. Some of these fixpacks might include an updated version of Cloudscape. You have the option to continue using your current level of Cloudscape if you want to. (It is not, however, recommended that you return to a lower level of Cloudscape if a fixpack has overwritten your current one.)

About the Cloudscape versions included with WebSphere Application Server 5.0.2 and 5.1

Cloudscape 5.0 database is shipped with Websphere Application Server 5.0.2. It is a required part of Websphere Application Server and is used for samples, the default application and is required for installation of cumulative fixes. With fixpack 2 (5.0.2), Cloudscape 5.0 and 5.1 were shipped. Cloudscape 5.1 is a fully functionally version of the Cloudscape database and is supported in a production environment with the fixpack 2. Cloudscape is installed in its own subdirectory. You should not remove these directories, even if you are not planning on directly using Cloudscape.

About your migration options

- If you choose to upgrade to a version of Cloudscape later than version 5.0, be aware that after migration (using `upgrade=true connectionAttributes`) you can no longer run your databases with Cloudscape 5.0.

- 5.1 + If you want to revert from Cloudscape version 5.1.x to a version 5.0.x, you can issue the java `-jar cloudscapeUninstall.jar` command to uninstall Cloudscape version 5.1.x. The `cloudscapeUninstall.jar file` is in the *install_root*`/cloudscape51/Uninstaller` directory.
- If you want to revert from either Cloudscape version 5.0 or 5.1 to an earlier version, however, use the following *Steps for this task* as your guide.

Note: If you cannot locate the *oldVersions* directory mentioned in the following steps, there is probably no old version installed for that release.

1. For fixes applied to WebSphere Application Server Version 5.0.2
   a. Open the *WAS_HOME/cloudscapeCommon/oldVersions* directory and pick the version you want.
   b. Unjar the JAR file that you select in step 1. Put it into a temporary directory, for example */tmp*. Now you have two options:
      1) If Cloudscape 5.1 is already installed:
         a) replace the Cloudscape jar files (db2j.jar and db2jtools.jar) in *WAS_HOME/lib* with those in *tmp/lib*
         b) replace the Cloudscape jar files in *WAS_HOME/cloudscape51/lib* with */tmp/lib*.
      2) If Cloudscape 5.1 is not already installed:
         a) replace the following in *WAS_HOME/cloudscape51* with those in */tmp/z* (where *z* applies only to zOS) For non-zOS, use:
            - *WAS_cloudscape51.jar*
2. Complete this additional step for fixes applied to WebSphere Application Server Version 5.1 and beyond:
   a. Open the *WAS_HOME/cloudscape/oldVersions* directory and choose the version that you want
   b. Unjar the jar file that you selected in step 1. This gives you a *lib* directory.
   c. Replace the jar file in the *WAS_HOME/cloudscape51/lib* with the unjared one.

Cloudscape Version 5.1 post installation instructions

After installing Cloudscape Version 5.1, you must complete the following steps before you can access the database.

1. Upgrade or migrate any existing database instances.
   a. Backup an existing database.

      You must complete a backup in case you have to access the previous version of Cloudscape. After you migrate a database, you cannot access your old database unless you perform a backup.
   b. Migrate an existing database by doing the following:
      - Set the ***connectionAttributes*** custom property to **`upgrade=true`**.

        The data source is located in the WebSphere Application Server administrative console under the JDBC providers.
      - If you are using cview, located in the `WAS_HOME/cloudscape51/bin/embedded` directory, click **`yes`** when you see the *upgrade database* prompt.

        Note: Ensure you migrate **`defaultDB`**, which is located in the `WAS_HOME/bin/DefaultDB` directory.

2. Set or change the classpath definitions in any existing JDBC providers, which are defined to use Cloudscape. Cloudscape jar files will not load when WebSphere Application Server is active.

   Use the WebSphere Application Server environment variable **${CLOUDSCAPE51_JDBC_DRIVER_PATH}\db2j.jar** to point to the new version of Cloudscape.

   The **CLOUDSCAPE51_JDBC_DRIVER_PATH** environment variable is defined in WebSphere Application Server with a value of `WAS_HOME/cloudscape51/lib`.
3. If the application server is running Cloudscape as a persistent store for UDDI in previous versions, additional steps are necessary.

   The server SystemOut.log might issue this message:

   ```
   The data source class name com.ibm.db2j.jdbc.db2jConnectionPoolDataSource could not be found.
   ```

   This is because the Cloudscape .jar file has moved to from its location in version 5.x to a new location in version 5.1.

   To correct this situation, do the following:

   a. Upgrade the database to Cloudscape 5.1.

   b. Rerun the install script, ***or*** edit the classpath field in the datasource.

Vendor-specific data sources minimum required settings

The following list contains descriptions for every JDBC provider supported in WebSphere Application Server Versions 5.0, 5.0.1, and 5.0.2. It also shows the supported data source classes and their required properties.

Specific fields are designated for the *user* and *password* properties. Inclusion of a property in the list does not imply that you should add it to the data source properties list. Rather, inclusion in the list means that a value is typically required for that field.

Use this list to find the provider information:
- DB2
- DB2 UDB for iSeries
- DB2 for z/OS
- Cloudscape
- Informix
- Sybase
- Oracle
- MS SQL Server

DB2

1. **DB2 Universal JDBC Driver Provider**

   The DB2 Universal JDBC Driver is an architecture-neutral JDBC driver for distributed and local DB2 access. Because the Universal Driver architecture is independent of any particular JDBC driver connectivity or target platform, it allows both Java connectivity (Type 4) or Java Native Interface (JNI) based connectivity (Type 2) in a single driver instance to DB2. Starting with WebSphere Application Server Version 5.0.2, the product now supports both Type 2 and Type 4 JDBC drivers. To use the Type 4 driver, you must install DB2 Version 8.1 or a later version. To use the Type 2 driver, you must install DB2 Version 8.1 Fix Pack 2 or a later version.

   This driver only supports one phase transactions. This JDBC driver allows applications to use both JDBC and Structured Query Language in Java (SQLJ) access.

   The DB2 Universal JDBC Driver Provider supports one phase data source:

com.ibm.db2.jcc.DB2ConnectionPoolDataSource

Requires JDBC driver files:

- **db2jcc.jar** After you install DB2, you can find this jar file in the DB2 *java* directory. For Type 4 JDBC driver support from a client machine where DB2 is not installed, copy this file to the local machine. If you install any fixes or upgrades to DB2, you must update this file as well. You must also set the ***DB2UNIVERSAL_JDBC_DRIVER_PATH*** path variable to point to the **db2jcc.jar** file. See the Cloudscape section for more information on the ***DB2UNIVERSAL_JDBC_DRIVER_PATH*** path variable.

  Note: To find out the version of the universal driver you are using, issue this DB2 command:

  ```
  java com.ibm.db2.jcc.DB2Jcc -version
  ```

  The output for the above example is:

  ```
  IBM DB2 JDBC Universal Driver Architecture 1.5.xx
  ```

- **db2jcc_license_cu.jar** This is the DB2 Universal JDBC driver license file that allows access to the DB2 Universal database. Use this jar file or the next one to gain access to the database. This jar file ships with WebSphere Application Server in a directory defined by **${UNIVERSAL_JDBC_DRIVER_PATH}**environment variable.
- **db2jcc_license_cisuz.jar** This is the DB2 Universal JDBC driver license file that allows access to the following databases:
  - DB2 Universal
  - DB2 for iSeries
  - DB2 for z/OS
  - SQLDS

  The **db2jcc_license_cisuz.jar** does not ship with Websphere Application Server and should be located in the same directory as the db2jcc.jar file, so that the ***DB2UNIVERSAL_JDBC_DRIVER_PATH*** points to both.

```
The classpath for this provider is set as follows:

   <classpath>${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc.jar </classpath>
   <classpath>${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cu.jar</classpath>
   <classpath>${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cisuz.jar</classpath>
```

Note: The license jar files are independent of each other; therefore, order does not matter.

Requires **DataStoreHelper** class:

```
com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper
```

If you use this driver provider to connect to DB2 for z/OS, you must change the data store helper class to:

```
com.ibm.websphere.rsadapter.DB2390DataStoreHelper
```

Requires a valid authentication alias.

Requires properties:

- **databaseName** This is an actual database name if the **driverType** is set to **4**, or a locally cataloged database name if the **driverType** is set to **2**.
- **driverType** The JDBC connectivity type of a data source. *There are two permitted values: 2 and 4*. If you want to use Universal JDBC Type 2 driver, set this value to 2. If you want to use Universal JDBC Type 4 driver, set this value to 4.

- **serverName** The TCP/IP address or host name for the Distributed Relational Database Architecture (DRDA) server. Provide a value for this property only if your **driverType** is set to **4**. This property is not required if your **driverType** is set to **2**.
- **portNumber** The TCP/IP port number where the DRDA server resides. Provide a value for this property only if your **driverType** is set to **4**. This property is not required if your **driverType** is set to **2**.

2. **DB2 Universal JDBC Driver Provider (XA)**

   The DB2 Universal JDBC Driver (XA) is an architecture-neutral JDBC driver for distributed and local DB2 access. In WebSphere Application Server Version 5.0.2, this driver only supports Java Native Interface (JNI) based connectivity (Type 2) in a single driver instance to DB2. To use this driver, you must install DB2 Version 8.1 Fix Pack 2 or a later version. This driver supports two phase transactions and the WebSphere Application Server Version 5.0 data source. This driver allows applications to use both JDBC and SQLJ access.

   The DB2 Universal JDBC Driver Provider supports the two phase data source:

   com.ibm.db2.jcc.DB2XADataSource

   Requires JDBC driver files:
   - **db2jcc.jar** After you install DB2, you can find this `.jar` file in the DB2 java directory. For Type 4 JDBC driver support from a client machine where DB2 is not installed, copy this file to the local machine. If you install any fixes or upgrades to DB2, you must update this file as well. You must also set the ***DB2UNIVERSAL_JDBC_DRIVER_PATH*** environment variable to point to the `db2jcc.jar` file. See the Cloudscape section for more information on the ***DB2UNIVERSAL_JDBC_DRIVER_PATH*** environment variable.

     Note: To find the level of universal driver you are using, issue the following DB2 command:

     ```
     java com.ibm.db2.jcc.DB2Jcc -version
     ```

     example output of the above:

     ```
     IBM DB2 JDBC Universal Driver Architecture 1.5.xx
     ```
   - **db2jcc_license_cu.jar** This is the DB2 Universal JDBC driver license file that allows access to the DB2 Universal database. Use this jar file or the next one to gain access to the database. This jar file ships with WebSphere Application Server in the *WAS_HOME/universalDriver/lib* directory.
   - **db2jcc_license_cisuz.jar** This is the DB2 Universal JDBC driver license file that allows access to the following databases:
     - DB2 Universal
     - DB2 for iSeries
     - DB2 for z/OS
     - SQLDS

     You must use the right license jar file to access a specific database backend.

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper`

   If you use this driver provider to connect to DB2 for z/OS, you must change the data store helper class to:

   `com.ibm.websphere.rsadapter.DB2390DataStoreHelper`

   Requires a valid authentication alias.

   Requires properties:
   - **databaseName** This is a locally cataloged database name.
   - **driverType** This is the JDBC connectivity type of a data source. *The only permitted value is 2.*

3. **DB2 legacy CLI-based Type 2 JDBC Driver**

   The DB2 legacy CLI-based Type 2 JDBC Driver Provider is built on top of DB2 CLI (Call Level Interface). It uses the DB2 CLI interface to communicate with DB2 UDB servers.

   DB2 legacy CLI-based Type 2 JDBC Driver supports one phase data source:

   COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource

   Requires JDBC driver files: **db2java.zip** (Note: If you run SQLJ in DB2 Version 8, **db2jcc.jar** is also required.)

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.DB2DataStoreHelper`

   Does not require a valid authentication alias.

   Requires properties:
   - **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

4. **DB2 legacy CLI-based Type 2 JDBC Driver (XA)**

   The DB2 legacy CLI-based Type 2 JDBC Driver (XA) is built on top of DB2 CLI (Call Level Interface). It uses the DB2 CLI interface to communicate with DB2 UDB servers.

   DB2 legacy CLI-based Type 2 JDBC Driver (XA) supports two phase data source:

   COM.ibm.db2.jdbc.DB2XADataSource

   Requires JDBC driver files: **db2java.zip** (Note: If you run SQLJ in DB2 Version 8, **db2jcc.jar** is also required.)

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.DB2DataStoreHelper`

   Does not require a valid authentication alias.

   Requires properties:
   - **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

5. **DB2 JDBC Provider** -- ***Deprecated***

   This JDBC provider is the same as the DB2 Legacy CLI-based Type 2 JDBC Driver. This JDBC driver is ***deprecated***. Use the DB2 Legacy CLI-based Type 2 JDBC Driver instead of this one.

   DB2 JDBC Provider supports one phase data source:

   COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource

   Requires JDBC driver files: **db2java.zip** (Note: If you run SQLJ in DB2 Version 8, **db2jcc.jar** is also required.)

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.DB2DataStoreHelper`

   Does not require a valid authentication alias.

   Requires properties:
   - **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

6. **DB2 JDBC Provider (XA)** -- ***Deprecated***

   This JDBC provider is the same as the DB2 Legacy CLI-based Type 2 JDBC Driver (XA). This JDBC driver is ***deprecated***. Use the DB2 Legacy CLI-based Type 2 JDBC Driver (XA) instead of this one.

   DB2 JDBC Provider (XA) supports two phase data source:

   COM.ibm.db2.jdbc.DB2XADataSource

Requires JDBC driver files: **db2java.zip** (Note: If you run SQLJ in DB2 Version 8, **db2jcc.jar** is also required.)

Requires **DataStoreHelper** class:

```
com.ibm.websphere.rsadapter.DB2DataStoreHelper
```

Does not require a valid authentication alias.

Requires properties:

- **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

For more information on DB2, visit the DB2 Web site at: http://www.ibm.com/software/data/db2/.

DB2 UDB for iSeries

1. **DB2 UDB for iSeries (Native - Version 5 Release 2 and later)**

   The iSeries Developer Kit for Java contains this Type 2 JDBC driver that is built on top of the iSeries DB2 Call Level Interface (CLI) native libraries. Only use this driver for local DB2 connections on iSeries. It is not recommended for remote access. Use this driver for iSeries V5R2, or later releases.

   DB2 UDB for iSeries (Native V5R2 and later) supports one phase data source:

   com.ibm.db2.jdbc.app.UDBConnectionPoolDataSource

   Requires JDBC driver files: **db2_classes.jar**

   Requires **DataStoreHelper** class:

   ```
   com.ibm.websphere.rsadapter.DB2AS400DataStoreHelper
   ```

   Does not require an authentication alias.

   Requires properties:

   - **databaseName** The name of the relational database to which the data source connections are established. This name must appear in the iSeries Relational Database Directory. The default is *LOCAL.

2. **DB2 UDB for iSeries (Native XA - Version 5 Release 2 and later)**

   The iSeries Developer Kit for Java contains this XA-compliant Type 2 JDBC driver built on top of the iSeries DB2 Call Level Interface (CLI) native libraries. Only use this driver for local DB2 connections on iSeries. It is not recommended for remote access. Use this driver for iSeries V5R2 or later releases.

   DB2 UDB for iSeries (Native XA - V5R2 and later) supports two phase data source:

   com.ibm.db2.jdbc.app.UDBXADataSource

   Requires JDBC driver files: **db2_classes.jar**

   Requires **DataStoreHelper** class:

   ```
   com.ibm.websphere.rsadapter.DB2AS400DataStoreHelper
   ```

   Does not require an authentication alias.

   Requires properties:

   - **databaseName** The name of the relational database to which the data source connections are established. This name must appear in the iSeries Relational Database Directory. The default is *LOCAL.

3. **DB2 UDB for iSeries (Native - Version 5 Release 1 and earlier)**

   The iSeries Developer Kit for Java contains this Type 2 JDBC driver that is built on top of the iSeries DB2 Call Level Interface (CLI) native libraries. Only use this driver for local DB2 connections on iSeries. It is not recommended for remote access. Use this driver for iSeries V5R1, or earlier releases.

   DB2 UDB for iSeries (Native V5R1 and earlier) supports one phase data source:

com.ibm.db2.jdbc.app.DB2StdConnectionPoolDataSource

Requires JDBC driver files: **db2_classes.jar**

Requires **DataStoreHelper** class:

com.ibm.websphere.rsadapter.DB2AS400DataStoreHelper

Does not require an authentication alias.

Requires properties:

- **databaseName** The name of the relational database to which the data source connections are established. This name must appear in the iSeries Relational Database Directory. The default is *LOCAL.

The V5R1 and earlier JDBC drivers, although still supported, will receive no further enhancements. We recommend that you replace them with the V5R2 JDBC driver providers.

4. **DB2 UDB for iSeries (Native XA - Version 5 Release 1 and earlier)**

   The iSeries Developer Kit for Java contains this XA-compliant Type 2 JDBC driver built on top of the iSeries DB2 Call Level Interface (CLI) native libraries. Only use this driver for local DB2 connections on iSeries. It is not recommended for remote access. Use this driver for iSeries V5R1, or earlier releases.

   DB2 UDB for iSeries (Native XA - V5R1 and earlier) supports two phase data source:

   com.ibm.db2.jdbc.app.DB2StdXADataSource

   Requires JDBC driver files: **db2_classes.jar**

   Requires **DataStoreHelper** class:

   com.ibm.websphere.rsadapter.DB2AS400DataStoreHelper

   Does not require an authentication alias.

   Requires properties:

   - **databaseName** The name of the relational database to which the data source connections are established. This name must appear in the iSeries Relational Database Directory. The default is *LOCAL.

   The V5R1 and earlier JDBC drivers, although still supported, will receive no further enhancements. We recommend that you replace them with the V5R2 JDBC driver providers.

5. **DB2 UDB for iSeries (Toolbox)**

   This JDBC driver, also known as iSeries Toolbox driver for Java, is provided in the DB2 for iSeries database server. Use this driver for remote DB2 connections on iSeries. We recommend you use this driver instead of the IBM Developer Kit for Java JDBC Driver to access remote DB2 UDB for iSeries systems.

   DB2 UDB for iSeries (Toolbox) supports one phase data source:

   com.ibm.as400.access.AS400JDBCConnectionPoolDataSource

   Requires JDBC driver files: **jt400.jar**

   Requires **DataStoreHelper** class:

   com.ibm.websphere.rsadapter.DB2AS400DataStoreHelper

   Does not require an authentication alias if WebSphere Application Server and DB2 UDB for iSeries are installed in the same server. If they are installed in different servers, the user ID and password are required.

   Requires properties:

   - **serverName** The name of the server from which the data source obtains connections. Example: *myserver.mydomain.com*.

6. **DB2 UDB for iSeries (Toolbox XA)**

This XA compliant JDBC driver, also known as iSeries Toolbox XA compliant driver for Java, is provided in the DB2 for iSeries database server. Use this driver for remote DB2 connections on iSeries. We recommend you use this driver instead of the IBM Developer Kit for Java JDBC Driver to access remote DB2 UDB for iSeries systems.

DB2 UDB for iSeries (Toolbox XA) supports two phase data source:

com.ibm.as400.access.AS400JDBCXADataSource

Requires JDBC driver files: **jt400.jar**

Requires **DataStoreHelper** class:

`com.ibm.websphere.rsadapter.DB2AS400DataStoreHelper`

Does not require an authentication alias if WebSphere Application Server and DB2 UDB for iSeries are installed in the same server. If they are installed in different servers, the user ID and password are required.

Requires properties:

- **serverName** The name of the server from which the data source obtains connections. Example: *myserver.mydomain.com*.

For more information on DB2 UDB for iSeries, visit the DB2 Web site at: http://www.ibm.com/software/data/db2/

DB2 for z/OS

1. **DB2 for z/OS JDBC Provider**

   The DB2 for z/OS JDBC Provider provides remote JDBC access to DB2 for z/OS. The DB2 for z/OS JDBC Provider provides remote JDBC access to DB2 on z/OS. This is the Legacy CLI-based JDBC driver. You cannot use this driver locally on DB2 z/OS. Use the DB2 for z/OS Local JDBC Provider (RRS) on local z/OS systems. You cannot see this JDBC Provider on the JDBC Provider drop down list in the administrative console. To create DB2 for z/OS data sources, use the DB2 Legacy CLI-based Type 2 JDBC Driver instead of this one, and change the DataStoreHelper class to **com.ibm.websphere.rsadapter.DB2390DataStoreHelper**.

   DB2 for z/OS JDBC provider supports one phase data source:

   COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource

   Requires JDBC driver files: **db2java.zip**

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.DB2390DataStoreHelper`

   Requires a valid authentication alias.

   Requires properties:

   - **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

2. **DB2 for z/OS JDBC Provider (XA)**

   The DB2 for z/OS JDBC Provider provides remote XA-compliant JDBC access to DB2 for z/OS. The DB2 for z/OS JDBC Provider provides remote JDBC access to DB2 on z/OS. This is the Legacy CLI-based JDBC driver. You cannot use this driver locally on DB2 z/OS. Use the DB2 for z/OS Local JDBC Provider (RRS) on local z/OS systems. You cannot see this JDBC Provider on the JDBC Provider drop down list in the administrative console. To create DB2 for z/OS XA data sources, use the DB2 Legacy CLI-based Type 2 JDBC Driver (XA) instead of this one, and change the DataStoreHelper class to **com.ibm.websphere.rsadapter.DB2390DataStoreHelper**.

   DB2 for z/OS JDBC (XA) provider supports two phase data source:

   COM.ibm.db2.jdbc.DB2XADataSource

Requires JDBC driver files: **db2java.zip**

Requires **DataStoreHelper** class:

`com.ibm.websphere.rsadapter.DB2390DataStoreHelper`

Requires a valid authentication alias.

Requires properties:

- **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

3. **DB2 for z/OS Local JDBC Provider (RRS)**

   The DB2 for z/OS JDBC Provider provides local JDBC access to DB2 for z/OS. The DB2 for z/OS Local JDBC Provider provides local JDBC access to DB2 on z/OS. This provider uses the native z/OS Resource Recovery Services (RRS) transaction manager to handle two-phase commit transactions. You cannot see this JDBC Provider on the JDBC Provider drop down list in the administrative console. This JDBC Provider is only used when accessing local databases in the z/OS system. Currently this JDBC Provider is notonly supported in WebSphere Application Server for z/OS.

   DB2 for z/OS Local JDBC provider supports two phase data source:

   COM.ibm.db2.jcc.DB2ConnectionPoolDataSource

   Requires JDBC driver files: **db2j2classes.zip**

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.DB2390LocalDataStoreHelper`

   Does not require a valid authentication alias.

   Requires properties:

   - **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

For more information on DB2 for z/OS, visit the DB2 Web site at: http://www.ibm.com/software/data/db2/.

Cloudscape

1. **Cloudscape JDBC Provider**

   The Cloudscape JDBC Provider provides the JDBC access to the Cloudscape database. This Cloudscape JDBC driver used the embedded framework. You cannot use any Version 4.0 data sources with Cloudscape.

   Cloudscape JDBC Provider supports one phase data source:

   com.ibm.db2j.jdbc.DB2jConnectionPoolDataSource

   Requires JDBC driver files: **db2j.jar.** (This file ships with WebSphere Application Server. If you are running Cloudscape Version 5.1, you can find this file in the `WAS_HOME/cloudscape/lib` directory).

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.CloudscapeDataStoreHelper`

   Does not require a valid authentication alias.

   Requires properties:

   - **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

2. **Cloudscape JDBC Provider (XA)**

   The Cloudscape JDBC Provider (XA) provides the XA-compliant JDBC access to the Cloudscape database. This Cloudscape JDBC driver uses the embedded framework. You cannot use any Version 4.0 data sources with Cloudscape.

   Cloudscape JDBC Provider (XA) supports two phase data source:

   com.ibm.db2j.jdbc.DB2jXADataSource

Requires JDBC driver files: **db2j.jar** (This file ships with WebSphere Application Server. If you are running Cloudscape Version 5.1, you can find this file in the `WAS_HOME/cloudscape/lib` directory).

Requires **DataStoreHelper** class:

`com.ibm.websphere.rsadapter.CloudscapeDataStoreHelper`

Does not require a valid authentication alias.

Requires properties:

- **databaseName** The name of the database from which the data source obtains connections. Example: *Sample* or *c:\sample*.

3. **Cloudscape Network Server using Universal JDBC driver**

   This Cloudscape driver takes advantage of the Network Server support that the DB2 universal Type 4 JDBC driver provides. You cannot use any Version 4.0 data sources with Cloudscape.

   Cloudscape uses the DB2 Universal Driver when using the Network Server. It supports one phase data source:

   com.ibm.db2.jcc.DB2ConnectionPoolDataSource

   Requires JDBC driver files:

   - **db2jcc.jar** If you install and run DB2, you must use the **db2jcc.jar** file that comes with DB2. To do that, the classpath in the JDBC template for Cloudscape network server is set to be:

     ```
     <classpath>${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc.jar</classpath>

     <classpath>${UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc.jar</classpath>

     <classpath>${CLOUDSCAPE_JDBC_DRIVER_PATH}/db2j.jar</classpath>

     <classpath>${CLOUDSCAPE51_JDBC_DRIVER_PATH}/db2j.jar</classpath>

           Note: This classpath is for Cloudscape 5.1.

     <classpath>${CLOUDSCAPE50_JDBC_DRIVER_PATH}/db2j.jar</classpath>

           Note: This classpath is for Cloudscape 5.0.

     <classpath>${UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cu.jar</classpath>
     ```

     which means that the db2jcc.jar from DB2 always takes precedence. Note that this also means that you must set the DB2 environment variable **DB2UNIVERSAL_JDBC_DRIVER_PATH** in WebSphere when you set up your DB2 datasource. This is instead of hard coding the path of the db2jcc.jar for DB2 datasources.
   - **db2jcc_license_cu.jar** This file is the DB2 Universal JDBC license file that provides access to the Cloudscape databases using the **Network Server** framework. Use this file to gain access to the database. This file ships with WebSphere and is located in **${UNIVERSAL_JDBC_DRIVER_PATH}**.

     Note: **UNIVERSAL_JDBC_DRIVER_PATH**is a WebSphere environment variable that is already defined to the location in Websphere Application Server where the license jar file above is located, and will only be used if the **DB2UNIVERSAL_JDBC_DRIVER_PATH** is not set. DB2 users should ensure that **DB2UNIVERSAL_JDBC_DRIVER_PATH** is set to avoid loading multiple vesions of the db2jcc.jar file.

     Note: **DB2UNIVERSAL_JDBC_DRIVER_PATH** is a WebSphere environment variable that you must set to point to the location of

db2jcc.jar file (that comes with DB2). This variable is set only if you create a db2 provider. See the DB2 section for more information on the **DB2UNIVERSAL_JDBC_DRIVER_PATH** path variable.

Note: Cloudscape requires only db2jcc_license_c.jar; however, WebSphere Application Server uses db2jcc_license_cu.jar because this works for both DB2 UDB and Cloudscape.

Requires **DataStoreHelper** class:

`com.ibm.websphere.rsadapter.CloudscapeNetworkServerDataStoreHelper`

Note: The administrative console incorrectly lists the DB2UniversalDataStoreHelper as the default value for the **DataStoreHelper** class. You must change the default value to `com.ibm.websphere.rsadapter.CloudscapeNetworkServerDataStoreHelper`. Also change the custom properties, using the instructions in the customer property section.

Requires a valid authentication alias.

Requires properties:

- **databaseName** The name of the database and the list of Cloudscape attributes. Example: *c:\sample;create=true*.
- **driverType** Only the Type 4 driver is allowed.
- **serverName** The TCP/IP address or the host name for the Distributed Relational Database Architecture (DRDA) server.
- **portNumber** The TCP/IP port number where the DRDA server resides. The default value is port ***1527***.
- **retrieveMessagesfromServerOnGetMessage** This property is required by WebSphere Application Server, not the database. The default value is ***false***. You must set the value of this property to ***true***, to enable text retrieval using the **SQLException.getMessage()** method.

See the Cloudscape setup instructions for more information on configuring the Cloudscape Network Server.

For more information on IBM Cloudscape, visit the Cloudscape Web site at: http://www.ibm.com/software/data/cloudscape/

Informix

1. **Informix JDBC Driver**

   The Informix JDBC Driver is a Type 4 JDBC driver that provides JDBC access to the Informix database.

   Informix JDBC Driver supports one phase data source:

   com.informix.jdbcx.IfxConnectionPoolDataSource

   Requires JDBC driver files:

   ifxjdbc.jar
   ifxjdbcx.jar

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.InformixDataStoreHelper`

   Requires a valid authentication alias.

   Requires properties:

   - **serverName** The name of the Informix instance on the server. Example: *ol_myserver*.
   - **portNumber** The port on which the instances listen. Example: *1526*.
   - **ifxIFXHOST** The physical name of the database server. Example: *myserver.mydomain.com*.

- **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.
- **informixLockModeWait** Although not required, this property enables you to set the number of seconds that Informix software waits for a lock. By default, Informix code throws an exception if it cannot immediately acquire a lock. Example: *2*.

2. **Informix JDBC Driver (XA)**

   The Informix JDBC Driver (XA) is a Type 4 JDBC driver that provides XA-compliant JDBC access to the Informix database.

   Informix JDBC Driver (XA) supports two phase data source:

   com.informix.jdbcx.IfxXADataSource

   Requires JDBC driver files:

   ifxjdbc.jar
   ifxjdbcx.jar

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.InformixDataStoreHelper`

   Requires a valid authentication alias.

   Requires properties:
   - **serverName** The name of the Informix instance on the server. Example: *ol_myserver*.
   - **portNumber** The port on which the instances listen. Example: *1526*.
   - **ifxIFXHOST** The physical name of the database server. Example: *myserver.mydomain.com*.
   - **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.
   - **informixLockModeWait** Although not required, this property enables you to set the number of seconds that Informix software waits for a lock. By default, Informix code throws an exception if it cannot immediately acquire a lock. Example: *2*.

For more information on Informix, visit the Informix Web site at: http://www.ibm.com/software/data/informix/

Sybase

1. **Sybase JDBC Driver**

   The Sybase JDBC Driver is a Type 4 JDBC driver that provides JDBC access to the Sybase database.

   Sybase JDBC Driver supports one phase data source:

   com.sybase.jdbc2.jdbc.SybConnectionPoolDataSource

   Requires JDBC driver files: **jconn2.jar**.

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.SybaseDataStoreHelper`

   Requires a valid authentication alias.

   Requires properties:
   - **serverName** The name of the database server. Example: *myserver.mydomain.com*.
   - **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.
   - **portNumber** The TCP/IP port number through which all communications to the server take place. Example: *4100*.

2. **Sybase JDBC Driver (XA)**

The Sybase JDBC Driver (XA) is a Type 4 JDBC driver that provides XA-compliant JDBC access to the Sybase database.

Sybase JDBC Driver (XA) supports two phase data source:

com.sybase.jdbc2.jdbc.SybXADataSource

Requires JDBC driver files: **jconn2.jar**.

Requires **DataStoreHelper** class:

`com.ibm.websphere.rsadapter.SybaseDataStoreHelper`

Requires a valid authentication alias.

Requires properties:

- **serverName** The name of the database server. Example: *myserver.mydomain.com*
- **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.
- **portNumber** The TCP/IP port number through which all communications to the server take place. Example: *4100*.

3. **Sybase 12.0 JDBC Driver -- *Deprecated***

   This JDBC Driver provider is the same as Sybase JDBC Driver. This JDBC driver is ***deprecated***. Use Sybase JDBC Driver instead of this one.

   Sybase 12.0 JDBC Driver supports one phase data source:

   com.sybase.jdbc2.jdbc.SybConnectionPoolDataSource

   Requires JDBC driver files: **jconn2.jar**.

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.SybaseDataStoreHelper`

   Requires a valid authentication alias.

   Requires properties:

   - **serverName** The name of the database server. Example: *myserver.mydomain.com*
   - **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.
   - **portNumber** The TCP/IP port number through which all communications to the server take place. Example: *4100*.

4. **Sybase 12.0 JDBC Driver (XA) -- *Deprecated***

   This JDBC Driver provider is the same as Sybase JDBC Driver (XA). This JDBC driver is ***deprecated***. Use the Sybase JDBC Driver (XA) instead of this one.

   Sybase 12.0 JDBC Driver (XA) supports two phase data source:

   com.sybase.jdbc2.jdbc.SybXADataSource

   Requires JDBC driver files: **jconn2.jar**.

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.SybaseDataStoreHelper`

   Requires a valid authentication alias.

   Requires properties:

   - **serverName** The name of the database server. Example: *myserver.mydomain.com*
   - **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.
   - **portNumber** The TCP/IP port number through which all communications to the server take place. Example: *4100*.

For more information on Sybase, visit the Sybase Web site at: http://www.sybase.com/

Oracle

1. **Oracle JDBC Driver**

   The Oracle JDBC Driver provides JDBC access to the Oracle database. This JDBC driver supports both Type 2 JDBC access and Type 4 JDBC access.

   Oracle JDBC Driver supports one phase data source:

   oracle.jdbc.pool.OracleConnectionPoolDataSource

   Requires JDBC driver files: **ojdbc14.jar**. (Note: If you require Oracle trace, use **ojdbc14_g.jar**.)

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.OracleDataStoreHelper`

   Requires a valid authentication alias.

   Requires properties:

   - **URL** The URL that indicates the database from which the data source obtains connections. Example: *jdbc:oracle:thin:@myServer:1521:myDatabase*, where *myServer* is the server name, *1521* is the port it is using for communication, and *myDatabase* is the database name.

2. **Oracle JDBC Driver (XA)**

   The Oracle JDBC Driver (XA) provides XA-compliant JDBC access to the Oracle database. This JDBC driver supports both Type 2 JDBC access and Type 4 JDBC access.

   Oracle JDBC Driver (XA) supports two phase data source:

   oracle.jdbc.xa.client.OracleXADataSource

   Requires JDBC driver files: **ojdbc14.jar**. (Note: If you require Oracle trace, use **ojdbc14_g.jar**.)

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.OracleDataStoreHelper`

   Requires a valid authentication alias.

   Requires properties:

   - **URL** The URL that indicates the database from which the data source obtains connections. Example: *jdbc:oracle:thin:@myServer:1521:myDatabase*, where *myServer* is the server name, *1521* is the port it is using for communication, and *myDatabase* is the database name.

3. **Oracle JDBC Thin Driver -- *Deprecated***

   Both the **Oracle JDBC Thin Driver** and the **Oracle JDBC oci8 Driver** are deprecated. Use the **Oracle JDBC Driver** instead of these two.

   Oracle JDBC Thin Driver is a Type 4 JDBC driver that provides JDBC access to the Oracle database. Oracle JDBC Thin Driver is ***deprecated***. Use the Oracle JDBC Driver instead of this one.

   Oracle JDBC Thin Driver supports one phase data source:

   oracle.jdbc.pool.OracleConnectionPoolDataSource

   Requires JDBC driver files: **ojdbc14.jar**. (Note: If you require Oracle trace, use **ojdbc14_g.jar**.)

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.OracleDataStoreHelper`

   Requires a valid authentication alias.

   Requires properties:

   - **URL**. The URL that indicates the database from which the data source obtains connections. Example: *jdbc:oracle:thin:@localhost:1521:sample* for thin

driver and *jdbc:oracle:oci8:@sample* for thick driver, where *localhost* is the server name, *1521* is the port it is using for communication, and *sample* is the database name.

4. **Oracle JDBC Thin Driver (XA) -- *Deprecated***

   Both the **Oracle JDBC Thin Driver (XA)** and the **Oracle JDBC oci8 Driver (XA)** are deprecated. Use the **Oracle JDBC Driver (XA)** instead of these two.

   Oracle JDBC Thin Driver (XA) is a Type 4 JDBC driver that provides XA-compliant JDBC access to the Oracle database. Oracle JDBC Thin Driver (XA) is ***deprecated***. Use the Oracle JDBC Driver (XA) instead of this one.

   Oracle JDBC Thin Driver (XA) supports two phase data source:

   oracle.jdbc.xa.client.OracleXADataSource

   Requires JDBC driver files: **ojdbc14.jar**. (Note: If you require Oracle trace, use **ojdbc14_g.jar**.)

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.OracleDataStoreHelper`

   Requires a valid authentication alias.

   Requires properties:

   - **URL** The URL that indicates the database from which the data source obtains connections. Example: *jdbc:oracle:thin:@localhost:1521:sample* for thin driver and *jdbc:oracle:oci8:@sample* for thick driver, where *localhost* is the server name, *1521* is the port it is using for communication, and *sample* is the database name.

5. **Oracle JDBC oci8 Driver -- *Deprecated***

   Both the **Oracle JDBC Thin Driver** and the **Oracle JDBC oci8 Driver** are deprecated. Use the **Oracle JDBC Driver** instead of these two.

   Oracle JDBC oci8 Driver is a Type 2 JDBC driver that provides JDBC access to the Oracle database. Oracle JDBC oci8 Driver is ***deprecated***. Use the Oracle JDBC Driver instead of this one.

   Oracle JDBC oci8 Driver supports one phase data source:

   oracle.jdbc.pool.OracleConnectionPoolDataSource

   Requires JDBC driver files: **ojdbc14.jar**. (Note: If you require Oracle trace, use **ojdbc14_g.jar**.)

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.OracleDataStoreHelper`

   Requires a valid authentication alias.

   Requires properties:

   - **URL** The URL that indicates the database from which the data source obtains connections. Example: *jdbc:oracle:thin:@localhost:1521:sample* for thin driver and *jdbc:oracle:oci8:@sample* for thick driver, where *localhost* is the server name, *1521* is the port it is using for communication, and *sample* is the database name.

6. **Oracle JDBC oci8 Driver (XA) -- *Deprecated***

   Both the **Oracle JDBC Thin Driver (XA)** and the **Oracle JDBC oci8 Driver (XA)** are deprecated. Use the **Oracle JDBC Driver (XA)** instead of these two.

   Oracle JDBC oci8 Driver (XA) is a Type 2 JDBC driver that provides XA-compliant JDBC access to the Oracle database. Oracle JDBC oci8 Driver (XA) is ***deprecated***. Use the Oracle JDBC Driver (XA) instead of this one.

   Oracle JDBC oci8 Driver (XA) supports two phase data source:

   oracle.jdbc.xa.client.OracleXADataSource

   Requires JDBC driver files: **ojdbc14.jar**. (Note: If you require Oracle trace, use **ojdbc14_g.jar**.)

Requires **DataStoreHelper** class:

`com.ibm.websphere.rsadapter.OracleDataStoreHelper`

Requires a valid authentication alias.

Requires properties:

- **URL** The URL that indicates the database from which the data source obtains connections. Example: *jdbc:oracle:thin:@localhost:1521:sample* for thin driver and *jdbc:oracle:oci8:@sample* for thick driver, where *localhost* is the server name, *1521* is the port it is using for communication, and *sample* is the database name.

For more information on Oracle, visit the Oracle Web site at:
http://www.oracle.com/

MS SQL Server

1. **DataDirect ConnectJDBC type 4 driver for MS SQL Server**

   DataDirect ConnectJDBC type 4 driver for MS SQL Server is a Type 4 JDBC driver that provides JDBC access to the MS SQL Server database. Only use this provider for the ConnectJDBC driver purchased from DataDirect Technologies. Do not use it with the Connect JDBC driver embedded in WebSphere.

   This JDBC provider supports this data source:

   com.ddtek.jdbcx.sqlserver.SQLServerDataSource

   Requires JDBC driver files:

   sqlserver.jar,
   base.jar and **util.jar**

   (The **spy.jar** file is optional. You need this file to enable spy logging. The **spy.jar** file is not in the same directory as the other three jar files. Instead, it is located in the `../spy/` directory.)

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.ConnectJDBCDataStoreHelper`

   Requires a valid authentication alias.

   Requires properties:

   - **serverName** The name of the server in which MS SQL Server resides. Example: *myserver.mydomain.com*
   - **portNumber** The TCP/IP port that MS SQL Server uses for communication. Port 1433 is the default.
   - **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

2. **DataDirect ConnectJDBC type 4 driver for MS SQL Server (XA)**

   DataDirect ConnectJDBC type 4 driver for MS SQL Server (XA) is a Type 4 JDBC driver which provides XA-compliant JDBC access to the MS SQL Server database. Only use this provider for the ConnectJDBC driver purchased from DataDirect Technologies. Do not use it with the Connect JDBC driver embedded in WebSphere.

   This JDBC provider supports this data source:

   com.ddtek.jdbcx.sqlserver.SQLServerDataSource.

   Requires JDBC driver files:

   sqlserver.jar,
   base.jar and **util.jar.**

   (The **spy.jar** file is optional. You need this file to enable spy logging. The **spy.jar** file is not in the same directory as the other three jar files. Instead, it is located in the `../spy/ directory`.)

   Requires **DataStoreHelper** class:

`com.ibm.websphere.rsadapter.ConnectJDBCDataStoreHelper`

Requires a valid authentication alias.

Requires properties:
- **serverName** The name of the server in which MS SQL Server resides. Example: *myserver.mydomain.com*
- **portNumber** The TCP/IP port that MS SQL Server uses for communication. Port 1433 is the default.
- **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

For more information on the DataDirect ConnectJDBC driver, visit the DataDirect Web site at: http://www.datadirect-technologies.com/

3. **WebSphere embedded ConnectJDBC driver for MS SQL Server**

   WebSphere embedded ConnectJDBC driver for MS SQL Server is a Type 4 JDBC driver that provides JDBC access to the MS SQL Server database. This JDBC driver ships with WebSphere Application Server. Only use this provider with the Connect JDBC driver embedded in WebSphere; it cannot be used with a Connect JDBC driver purchased separately from DataDirect Technologies.

   This JDBC provider supports this data source:

   `com.ibm.websphere.jdbcx.sqlserver.SQLServerDataSource.`

   Requires JDBC driver files:

   `sqlserver.jar`
   `base.jar` and
   `util.jar.`

   (The **`spy.jar`** file is optional. You need this file to enable spy logging. The **`spy.jar`** file for the WebSphere embedded Connect JDBC driver ships with WebSphere Application Server. All the files are located in the `WAS_HOME/lib/` directory.)

   Requires **`DataStoreHelper`** class:

   `com.ibm.websphere.rsadapter.WSConnectJDBCDataStoreHelper`

   Requires a valid authentication alias.

   Requires properties:
   - **serverName** The name of the server in which MS SQL Server resides. Example: *myserver.mydomain.com*
   - **portNumber** The TCP/IP port that MS SQL Server uses for communication. Port 1433 is the default.
   - **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

4. **WebSphere embedded ConnectJDBC driver for MS SQL Server (XA)**

   WebSphere embedded ConnectJDBC driver for MS SQL Server (XA) is a Type 4 JDBC driver which provides XA-compaint JDBC access to the MS SQL Server database. This JDBC driver ships with WebSphere Application Server. Use this provider with the Connect JDBC driver embedded in WebSphere. Do not use it with a Connect JDBC driver purchased separately from DataDirect Technologies.

   This JDBC provider supports this data source:

   `com.ibm.websphere.jdbcx.sqlserver.SQLServerDataSource.`

   Requires JDBC driver files:

   `sqlserver.jar`
   `base.jar` and
   `util.jar.`

(The **spy.jar** file is optional. You need this file to enable spy logging. The **spy.jar** file for the WebSphere embedded Connect JDBC driver ships with WebSphere Application Server. All the files are located in the `WAS_HOME/lib/` directory.)

Requires **DataStoreHelper** class:

`com.ibm.websphere.rsadapter.WSConnectJDBCDataStoreHelper`

Requires a valid authentication alias.

Requires properties:

- **serverName** The name of the server in which MS SQL Server resides. Example: *myserver.mydomain.com*
- **portNumber** The TCP/IP port that MS SQL Server uses for communication. Port 1433 is the default.
- **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

To view the instructions for installing *Stored Procedures for JTA* (required when enabling two-phase commit) on the WebSphere CD, refer to the following FTP site:

`How to install "Stored Procedures for JTA" from the WebSphere CD`

You can download the latest patches and upgrades to the WebSphere embedded ConnectJDBC drivers from the following FTP site:

`ftp://ftp.software.ibm.com/software/websphere/info/tools/DataDirect/datadirect.htm`

5. **DataDirect SequeLink type 3 JDBC driver for MS SQL Server**

   DataDirect SequeLink type 3 JDBC driver for MS SQL Server is a Type 3 JDBC driver that provides JDBC access to MS SQL Server via SequeLink server. This JDBC driver provider supports both the SequeLink type 3 JDBC driver shipped with WebSphere Application Server and the SequeLink type 3 JDBC driver purchased from DataDirect.

   This JDBC provider supports this data source:

   com.ddtek.jdbcx.sequelink.SequeLinkDataSource

   Requires JDBC driver files:

   sljc.jar and
   spy-sl53.jar

   (The JDBC driver shipped with WebSphere Application Server requires the **sljc.jar** and the **spy-sl53.jar** files. The JDBC driver purchased from DataDirect requires the **sljc.jar** and the **spy.jar** files. The **spy.jar** and **spy-sl35.jar** files are optional. You need these files to enable spy logging.)

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.SequeLinkDataStoreHelper`

   Requires a valid authentication alias.

   Requires properties:

   - **serverName** The name of the server in which SequeLink Server resides. Example: *myserver.mydomain.com*
   - **portNumber** The TCP/IP port that SequeLink Server uses for communication. By default, SequeLink Server uses port 19996.
   - **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

6. **DataDirect SequeLink type 3 JDBC driver for MS SQL Server (XA)**

   DataDirect SequeLink type 3 JDBC driver for MS SQL Server (XA) is a Type 3 JDBC driver which provides XA-compliant JDBC access to MS SQL Server via the SequeLink server. This JDBC driver provider supports both the SequeLink type 3 JDBC driver shipped with WebSphere Application Server and the SequeLink type 3 JDBC driver purchased from DataDirect.

This JDBC provider supports this data source:

com.ddtek.jdbcx.sequelink.SequeLinkDataSource

Requires JDBC driver files:

sljc.jar and
spy-sl53.jar

(The JDBC driver shipped with WebSphere Application Server requires the **sljc.jar** and the **spy-sl53.jar** files. The JDBC driver purchased from DataDirect requires the **sljc.jar** and the **spy.jar** files. The **spy.jar** and **spy-sl35.jar** files are optional. You need these files to enable spy logging.)

Requires **DataStoreHelper** class:

com.ibm.websphere.rsadapter.SequeLinkDataStoreHelper

Requires a valid authentication alias.

Requires properties:

- **serverName** The name of the server in which SequeLink Server resides. Example: *myserver.mydomain.com*
- **portNumber** The TCP/IP port that SequeLink Server uses for communication. By default, SequeLink Server uses port 19996.
- **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

To view instructions for installing SequeLink Server from the WebSphere CD, refer to this FTP site:

How to install SequeLink Server from the WebSphere CD

Only install the SequeLink server from the WebSphere CD if you are using the SequeLink JDBC driver embedded in WebSphere. Otherwise, install the SequeLink server purchased from DataDirect Technologies.

You can download the latest patches and upgrades to the WebSphere embedded SequeLink type 3 JDBC drivers from the following FTP site:

ftp://ftp.software.ibm.com/software/websphere/info/tools/DataDirect/datadirect.htm

For more information on the DataDirect SequeLink type 3 JDBC driver, visit the DataDirect Web site at:

http://www.datadirect-technologies.com/

7. **Microsoft JDBC driver for MSSQLServer 2000**

   Microsoft JDBC driver for MSSQLServer 2000 is a type 4 JDBC driver that provides JDBC access to the MS SQL Server database.

   This JDBC provider supports this data source:

   com.microsoft.jdbcx.sqlserver.SQLServerDataSource

   Requires JDBC driver files:

   mssqlserver.jar,
   msbase.jar and **msutil.jar**

   (The **spy.jar** file is optional. You need it to enable spy logging. However, Microsoft does not ship the **spy.jar** file. Contact Microsoft about this issue.)

   Requires **DataStoreHelper** class:

   com.ibm.websphere.rsadapter.ConnectJDBCDataStoreHelper

   Requires a valid authentication alias.

   Requires properties:

   - **serverName** The name of the server in which MS SQL Server resides. Example: *myserver.mydomain.com*
   - **portNumber** The TCP/IP port that MS SQL Server uses for communication. Port 1433 is the default.
   - **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

8. **Microsoft JDBC driver for MSSQLServer 2000 (XA)**

   Microsoft JDBC driver for MSSQLServer 2000 (XA) is a type 4 JDBC driver that provides XA-compaint JDBC access to the MS SQL Server database.

   This JDBC provider supports this data source:

   com.microsoft.jdbcx.sqlserver.SQLServerDataSource

   Requires JDBC driver files:

   mssqlserver.jar,
   msbase.jar and **msutil.jar**

   (The **spy.jar** file is optional. You need it to enable spy logging. However, Microsoft does not ship the **spy.jar** file. Contact Microsoft about this issue.)

   Requires **DataStoreHelper** class:

   com.ibm.websphere.rsadapter.ConnectJDBCDataStoreHelper

   Requires a valid authentication alias.

   Requires properties:
   - **serverName** The name of the server in which MS SQL Server resides. Example: *myserver.mydomain.com*
   - **portNumber** The TCP/IP port that MS SQL Server uses for communication. Port 1433 is the default.
   - **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

For more information on the Microsoft JDBC driver, visit the Microsoft Web site at:
http://www.microsoft.com/sql

Connector Modules collection

Use this page to view connector module settings.

An instance of ConnectorModuleDeployment is created for every connector module (RAR) in the application.

To view this administrative console page, click **Applications** >**Enterprise Applications** > *application* > **Connector Modules**.

URI

Specifies the logical path to the resource that will be serviced by the product.

Data type String

Name

Specifies the display name of the connector module.

Data type String

Connector module settings

Use this page to view connector modules.

To view this administrative console page, click **Applications** > **Enterprise Applications** > *application* > **Connector Modules** > *connector_module*.

Uri:

Specifies the logical path to the resource that is serviced by WebSphere Application Server.

Data type String

Name:

Specifies the display name of the connector module.

Data type String

altDD:

Specifies the alternate DD of the connector module.

The alternate DD URI for a given module.

Data type String

Starting weight:

Specifies the startup priority of the connector module over others.

When your application contains multiple modules, the starting weight you specify determines this module's startup priority over other modules during server startup. Modules with lower startup order are started first.

Data type String

Data access : Resources for learning

Use the following links to find relevant supplemental information about data access. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

These links are provided for convenience. Often, the information is not specific to this product but is useful all or in part for understanding the product. When possible, links are provided to technical papers and Redbooks that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas.

Program Specifications
- What's new in the Enterprise JavaBeans 2.0 Specification?

  You can also download the specification itself from this URL.
- Java™ 2 Platform, Enterprise Edition (J2EE™)
- Java™ Management Extensions (JMX)

CMP persistence functions
- Enterprise JavaBeans™ 2.0 Container-Managed Persistence Example

Container managed relationships
- Enterprise JavaBeans™ 2.0 Container-Managed Persistence Example

Local interfaces
- Enterprise JavaBean™ 2.0 Specification Changes

Resource references
- Accessing Databases from Web Applications

WebSphere Studio Application Developer (WSAD)
- WebSphere Studio Application Developer

WebSphere Studio Application Developer Integration Edition (WSADIE)
- WebSphere Studio Application Developer Integration Edition (WSADIE)

WebSphere Version 4.0 Information Center
- IBM WebSphere™ Version 4.0 Information Center

IBM Cloudscape
- IBM Cloudscape

Oracle
- Oracle

DB2 database software
- DB2

Supported hardware, software, and APIs
- Supported hardware, software, and APIs

Tuning databases

Tuning parameters vary according to the type of database you are using. DB2 tuning tips for z/OS are provided for your convenience. Since DB2 is not a WebSphere Application Server product and can change, consider these descriptions as suggestions.

Chapter 3. Using asynchronous messaging

Use these topics to use asynchronous messaging with WebSphere Application Server, to enable enterprise applications to use JMS resources, message-driven beans, and extended messaging.

WebSphere Application Server supports asynchronous messaging as a method of communication based on the Java Message Service (JMS) programming interface.

The base JMS support enables WebSphere enterprise applications to exchange messages asynchronously with other JMS clients by using JMS destinations (queues or topics). An enterprise application can explicitly poll for messages on a destination.

WebSphere Application Server also provides a message listener service that enterprise applications can use to automatically retrieve messages from JMS destinations for processing by message-driven beans, without the application having to explicitly poll JMS destinations.

5.1 + With WBI Server Foundation, enterprise applications can use another level of functionality for asynchronous messaging, called extended messaging. The application server manages the messaging infrastructure, and extra standard types of messaging beans are provided to add functionality to that provided by message-driven beans. This level of functionality enables application developers to concentrate on the business logic to be implemented by the enterprise beans and to leave the messaging usage to standard messaging objects and configuration of the extended messaging service.

You can use the WebSphere administrative console to administer the WebSphere Application Server support for asynchronous messaging. For example, you can configure JMS providers and their resources, and can control the activity of the JMS server and the messaging services.

For more information about implementing WebSphere enterprise applications that use asynchronous messaging, see the following topics:
- An overview of WebSphere asynchronous messaging
- Implementing WebSphere enterprise applications that use JMS
- Implementing WebSphere enterprise applications that use message-driven beans

For more information about JMS, see the JMS documentation at http://java.sun.com/products/jms/docs.html.

Asynchronous messaging with WebSphere - an overview

WebSphere Application Server supports asynchronous messaging as a method of communication based on the Java Message Service (JMS) programming interface.

The base JMS support enables WebSphere J2EE applications to exchange messages asynchronously with other JMS clients by using JMS destinations (queues or topics). A J2EE application can explicitly poll for messages on a destination.

WebSphere Application Server also provides a message listener service that J2EE applications can use to automatically retrieve messages from JMS destinations for processing by message-driven beans, without the application having to explicitly poll JMS destinations.

5.1 + With WBI Server Foundation, J2EE applications can use another level of functionality for asynchronous messaging, called extended messaging. The application server manages the messaging infrastructure, and extra standard types of messaging beans are provided to add functionality to that provided by message-driven beans. This level of functionality enables application developers to concentrate on the business logic to be implemented by the enterprise beans and to leave the messaging usage to standard messaging objects and configuration of the extended messaging service.

You can use the WebSphere administrative console to administer the WebSphere Application Server support for asynchronous messaging. For example, you can configure JMS providers and their resources, and can control the activity of the JMS server and the messaging services.

Styles of messaging in applications

Applications can use the following styles of asynchronous messaging:

Point-to-Point
: Point-to-point applications use queues to pass messages between each other. The applications are called point-to-point, because a client sends a message to a specific queue and the message is picked up and processed by a server listening to that queue. It is common for a client to have all its messages delivered to one queue. Like any generic mailbox, a queue can contain a mixture of messages of different types.

Publish/subscribe
: Publish/subscribe systems provide named collection points for messages, called topics. To send messages, applications publish messages to topics. To receive messages, applications subscribe to topics; when a message is published to a topic, it is automatically sent to all the applications that are subscribers of that topic. By using a topic as an intermediary, message publishers are kept independent of subscribers.

Both styles of messaging can be used in the same application.

Applications can use asynchronous messaging in the following ways:

One-way
: An application sends a message, and does not want a response. This pattern of use is often referred to as a datagram.

Request / response
: An application sends a request to another application and expects to receive a response in return.

One-way and forward
: An application sends a request to another application, which sends a message to yet another application.

These messaging techniques can be combined to produce a variety of asynchronous messaging scenarios.

For more information about how such messaging scenarios are used by WebSphere enterprise applications, see the following topics:

- An overview of asynchronous messaging with JMS

- An overview of asynchronous messaging with message-driven beans

For more information about these messaging techniques and the Java Messaging Service (JMS), see Sun's Java Message Service (JMS) specification documentation.

WebSphere Application Server cloning and WebSphere MQ clustering

This topic provides a summary of information about using WebSphere Application Server horizontal cloning with WebSphere MQ server clustering support. It describes a scenario that shows how the message listener service can be configured to take advantage of WebSphere MQ server clustering and provides some information about how to resolve potential runtime failures in the clustering scenario. The information in this topic is based on the scenario shown in the following figure labeled "WebSphere Application Server horizontal cloning with WebSphere MQ clustered queues."

Note: WebSphere MQ server clustering is only available with the full WebSphere MQ product installed as the JMS provider.

For a WebSphere application server configured to use the extended messaging service, each JMS listener is used to retrieve messages from destinations defined to the server. In the following information the listener configurations are the same for each WebSphere application server. Each application server host contains a WebSphere application server and an WebSphere MQ server. If a host is only used to distribute messages, it only contains an WebSphere MQ server. There can be many servers defined in the configuration, although for simplicity the information in this topic is based on a scenario containing only three servers (as shown in the figure).

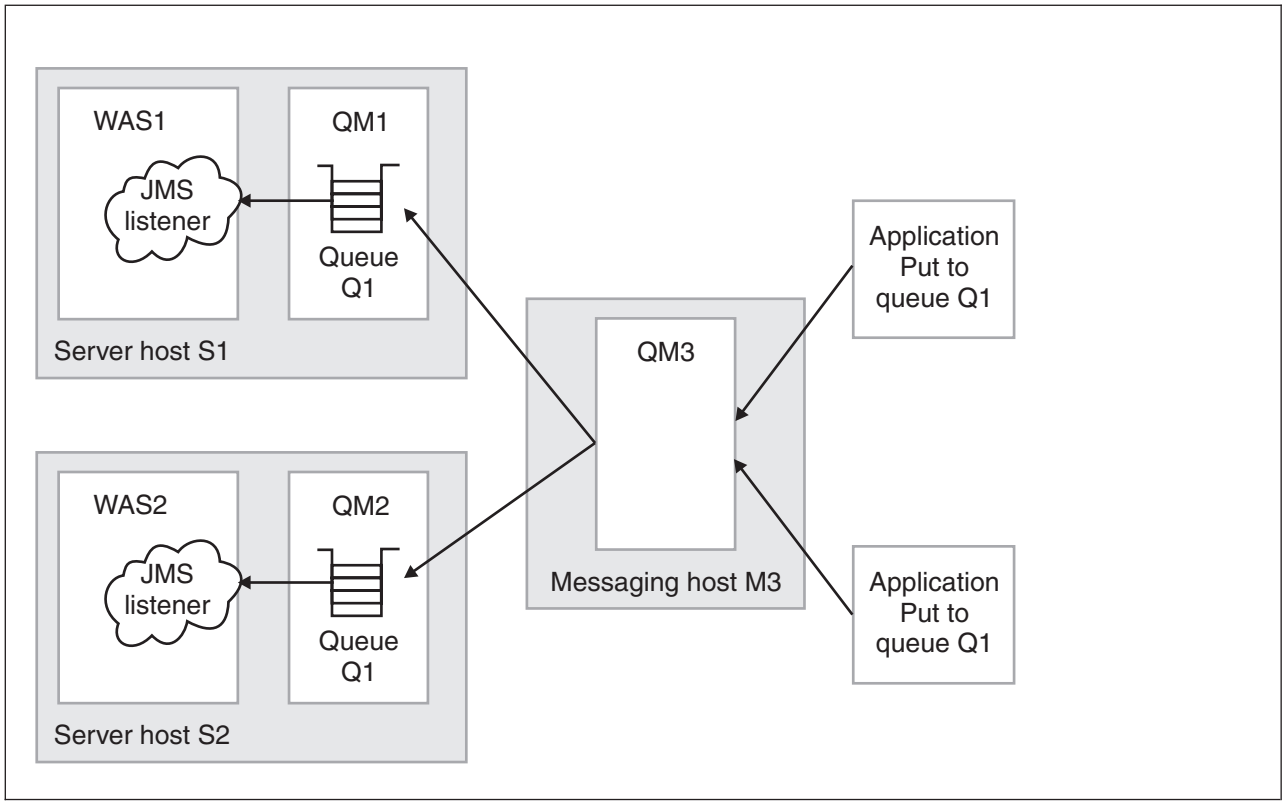

Figure 1. WebSphere Application Server horizontal cloning with WebSphere MQ clustered queues. This figure shows two WebSphere Application Server hosts, with horizontal clustering, and a messaging host used to distribute messages for WebSphere MQ server clustering.

The scenario shown in the figure comprises the following three hosts:

- Server host S1 contains the following servers:

  WebSphere MQ server.
  The server is defined to have a queue manager, QM1, and a local queue, Q1. The queue manager belongs to a cluster. The queue is populated by the WebSphere MQ server located on host M3. Applications can add messages directly to the queue, Q1, but would not be subjected to the control of the WebSphere MQ cluster.

  WebSphere Application Server
  This contains a cloned application server, WAS1, which is configured with a JMS listener. The listener is configured to retrieve messages from JMS destination Q1.
- Server host S2 contains the following servers:

  WebSphere MQ server.
  The server is defined to have a queue manager, QM2, and a local queue, Q1. The queue manager belongs to the same cluster as QM1 on host S1. As with QM1, the queue is populated by the WebSphere MQ server located on host M3. Applications can add messages directly to the queue, Q1, but would not be subjected to the control of the MQ cluster.

  WebSphere Application Server
  This contains a cloned application server, WAS2 (identical to WAS1 on host S1), which is configured with a JMS listener. The listener is configured to retrieve messages from JMS destination Q1.
- Messaging host M3 contains the following servers:

  WebSphere MQ server.
  The server is defined to have a queue manager, QM3, which also belongs to the same cluster as QM1 and QM2. Applications add

messages to the queue manager and queue Q1. The cluster to which this queue manager belongs causes messages to be distributed to all other queue managers in the cluster which have queue Q1 defined.

Note: Queue Q1 is not defined as a local queue on this host. If the queue was defined locally, then messages would remain on the server for local processing; messages would not be distributed by the queue manager cluster control to the other queue managers in the cluster that do have the queue defined.

This host does not have a WebSphere application server defined. All message retrieval processing is performed by the other two application servers on hosts S1 and S2.

Recovery scenarios

There are several failure scenarios that could occur with the clustering configuration; for example:

- WAS server failures.

  In this scenario the failure of any single WebSphere application server results in the messages for the specified destination remaining on the queue, until the server is restarted.
- WebSphere MQ Queue Manager failures.

  There are two different failures to consider:
  1. Failure of a queue manager on the same host as a WebSphere application server (for example, failure of QM2 on host S2). In this case messages are delivered to the other available application servers, until the WebSphere MQ server is back online, when messages are processed as expected.
  2. Failure of the messaging host M3 and its queue manager, QM3. In this case, the result of an outage is more significant because no messages are delivered to the other queue managers in the cluster. In a fully-deployed and scaled production system, host M3 would not be designed to be a single point of failure, and additional messaging servers would be added to the cluster configuration.

Using JMS and messaging in applications

Use these tasks to implement WebSphere J2EE applications that use JMS.

WebSphere Application Server supports asynchronous messaging as a method of communication based on the Java Message Service (JMS) programming interface.

The base JMS support enables WebSphere enterprise applications to exchange messages asynchronously with other JMS clients by using JMS destinations (queues or topics). An enterprise application can explicitly poll for messages on a destination.

Using the base support for JMS, you can build enterprise beans that use the JMS API directly to provide messaging services along with methods that implement business logic.

You can use the WebSphere administrative console to administer the JMS support of WebSphere Application Server. For example, you can configure JMS providers and their resources, and can control the activity of the JMS server.

For more information about implementing WebSphere enterprise applications that use JMS, see the following topics:

- An overview of WebSphere asynchronous messaging using JMS
- Administering WebSphere messaging
- Developing a J2EE application to use JMS
- Developing a JMS client
- Deploying a J2EE application to use JMS
- Tuning Java messaging service
- Troubleshooting WebSphere messaging

For more information about JMS, see the JMS documentation at http://java.sun.com/products/jms/docs.html.

WebSphere MQ and IBM WebSphere Application Server

Applications written in the Java programming language running under IBM WebSphere Application Server can use the Java Messaging Service (JMS) specification to perform messaging. Point-to-point messaging in this environment is provided by a WebSphere MQ for z/OS queue manager. Two classes of configuration are available:

- A reduced function form of WebSphere MQ for z/OS, which is also known as the integral JMS provider, or IJP.

  Here, the queue manager providing the point-to-point messaging runs the WebSphere MQ for z/OS base function code provided with IBM WebSphere Application Server. This environment is suited to simple messaging between JMS applications running in the IBM WebSphere Application Server environment.
- Full function WebSphere MQ for z/OS.

  Here, the queue manager providing the point-to-point messaging runs WebSphere MQ for z/OS full function code installed as a separate product from IBM WebSphere Application Server for z/OS. In this environment, JMS applications running in the IBM WebSphere Application Server can participate fully in the functionality of a WebSphere MQ for z/OS network. For example, they can make use of the IMS Bridge, or exchange messages with WebSphere MQ for z/OS queue managers running on other platforms.

The information for WebSphere MQ for z/OS is supplied as a number of books, in PDF format. These books describe the full function form of WebSphere MQ for z/OS, but also identify the features, commands, API verbs, and so on, that are unavailable or restricted in the reduced function form. The WebSphere MQ for z/OS Library Web site describes the documentation available.

Connection between IBM WebSphere Application Server and a queue manager

If the queue manager providing point-to-point messaging is a reduced function queue manager, connection must be through either TCP client/server channel. In JMS terms, this means choosing client transport for the queue connection factory object. The integral JMS provier (IJP) supports direct IP transport or MQ transport.

If the queue manager providing point-to-point messaging is a full function queue manager, you can choose either client transport or bindings transport for the queue connection factory object. If you choose bindings transport, the WebSphere Application Server and the queue manager must both exist on the same z/OS image. If you choose client transport, you must install the Client Attachment feature of WebSphere MQ for z/OS.

Both types of connection support transactional applications: the client transport by using XA protocols; the bindings transport by using a WebSphere Application Server stub, CSQBWSTB, which uses RRS services.

Integral JMS provider (IJP)

The WebSphere MQ for z/OS information tells you what reduced function means by identifying those functions not available in the reduced function form of WebSphere MQ for z/OS supplied with WebSphere Application Server, both in general terms, and at the command level. The list below summarizes the restrictions in the reduced function form of WebSphere MQ for z/OS:

Queue sharing groups

Queue sharing groups are not available:

- If you set the QSGDATA system parameter to anything other than blank, the queue manager does not start.
- CSQ5PQSG does not run.

Clusters

Clusters are not available. You cannot:

- Define SYSTEM.CLUSTER queues
- Define, alter, or delete CLUSSDR or CLUSRCVR channels
- Set queue CLUSTER or CLUSNL attributes to anything other than blank
- Set queue manager REPOS or REPOSNL attributes to anything other than blank

The following commands have no effect:

- RESET or REFRESH CLUSTER
- SUSPEND or RESUME QMGR CLUSTER or CLUSNL
- DISPLAY CLUSQMGR

The following functions do not run:

- Mover repository manager
- CLUSSDR or CLUSRCVR channels

Distributed queuing

The reduced function form of WebSphere MQ for z/OS supplied with WebSphere Application Server does not support channels other than server-connection channels. This means that:

- MCA channels are not available:
  - You cannot define, alter, or delete SDR, SVR, RCVR, or RQSTR channels
  - SDR, SVR, RCVR, or RQSTR channels do not run
- Only SVRCONN channels are available; you cannot define or alter CLNTCONN channels

TCP types other than OESOCKET

If you set the TCPTYPE system parameter to anything other than OESOCKET, the mover does not start.

LU62 communications

You cannot use LU6.2 channels and listeners. Specifically, you must not set the LU62CHL system parameter to anything other than 0 or the mover does not start.

Events

Events are not available; you cannot:

- Define SYSTEM.ADMIN queues
- Enable queue manager event attributes

CICS connection

There is no support for the WebSphere MQ for z/OS CICS connection. As a result, an MQCONN or MQCONNX from a CICS transaction fails with an MQRC_CONNECTION ERROR.

IMS connection
There is no support for the WebSphere MQ for z/OS IMS connection. As a result, an MQCONN or MQCONNX from an IMS application fails with an MQRC_CONNECTION ERROR.

RRS connection
There is no support for the WebSphere MQ for z/OS RRS connection. As a result, an MQCONN or MQCONNX from an RRS application fails with an MQRC_CONNECTION ERROR.

IMS Bridge
There is no support for the WebSphere MQ for z/OS IMS Bridge; the resource manager does not start.

Measured usage license charge (MULC)
This function is bypassed.

An overview of WebSphere asynchronous messaging using JMS

WebSphere Application Server supports asynchronous messaging as a method of communication based on the Java Message Service (JMS) programming interface. JMS provides a common way for Java programs (clients and J2EE applications) to create, send, receive, and read asynchronous requests, as JMS messages.

This topic provides an overview of asynchronous messaging using JMS support provided by WebSphere Application Server. For more details about JMS, see Sun's Java Message Service (JMS) specification documentation.

The base support for asynchronous messaging using JMS, shown in the following figure, provides the common set of JMS interfaces and associated semantics that define how a JMS client can access the facilities of a JMS provider. This enables WebSphere J2EE applications, as JMS clients, to exchange messages asynchronously with other JMS clients by using JMS destinations (queues or topics). A J2EE application can use JMS queue destinations for point-to-point messaging and JMS topic destinations for publish/subscribe messaging. A J2EE application can explicitly poll for messages on a destination then retrieve messages for processing by business logic beans (enterprise beans).

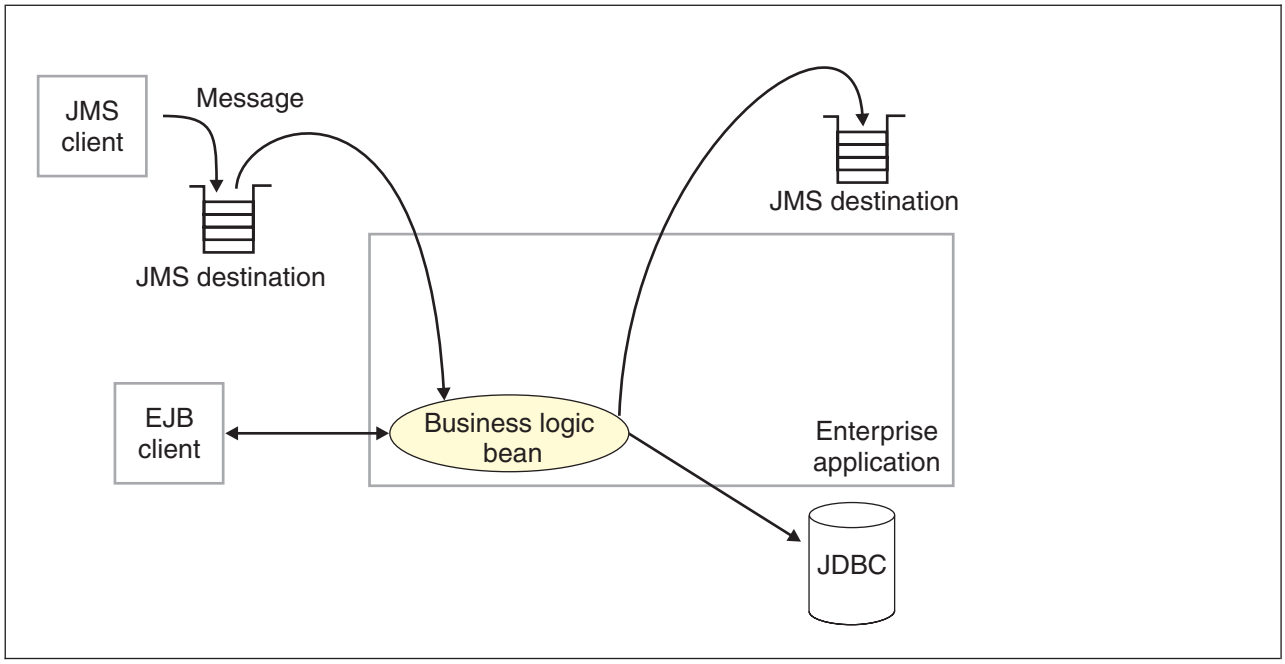

Figure 2. Asynchronous messaging using JMS. This figure shows an enteprise application polling a JMS destination to retrieve an incoming message, which it processes with a business logic bean. The business logic bean uses standard JMS calls to process the message; for example, to extract data or to send the message on to another JMS destination. For more information, see the text that accompanies this figure.

With the base JMS/XA support, the J2EE application uses standard JMS calls to process messages, including any responses or outbound messaging. Responses can be handled by an enterprise bean acting as a sender bean, or handled in the enterprise bean that receives the incoming messages. Optionally, this process can use two-phase commit within the scope of a transaction. This level of functionality for asynchronous messaging is called bean-managed messaging, and gives an enterprise bean complete control over the messaging infrastructure; for example, for connection and session pool management. The application server has no role in bean-managed messaging.

WebSphere Application Server also supports automatic asynchronous messaging using message-driven beans (a type of enterprise bean defined in the EJB 2.0 specification) and JMS listeners (part of the JMS application server facilities). Messages are automatically retrieved from JMS destinations, optionally within a transaction, then sent to the message-driven bean in a J2EE application, without the application having to explicitly poll JMS destinations. For more information about asynchronous messaging with message-driven beans, see An overview of asynchronous messaging with message-driven beans

5.1 + With WBI Server Foundation, J2EE applications can use another level of functionality for asynchronous messaging, called extended messaging. The application server manages the messaging infrastructure, and extra standard types of messaging beans are provided to add functionality to that provided by message-driven beans. This level of functionality enables application developers to concentrate on the business logic to be implemented by the enterprise beans and to leave the messaging usage to standard messaging objects and configuration of the extended messaging service.

WebSphere JMS support - components

The main components of WebSphere JMS support are shown in the following figure and described after the figure:

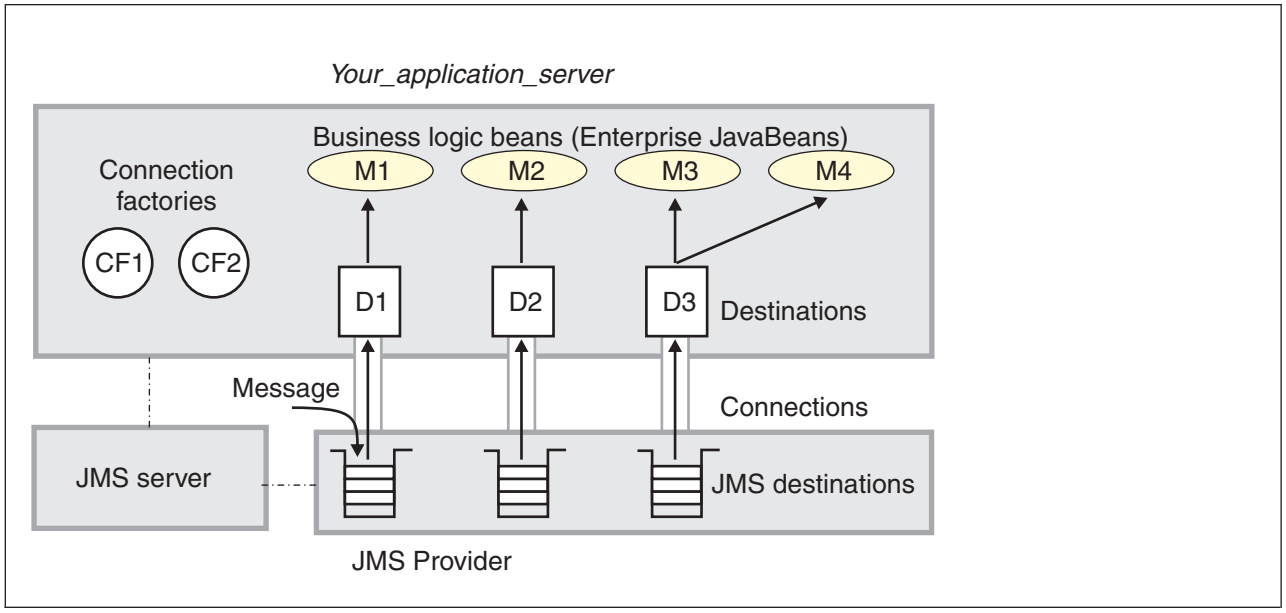

Figure 3. The main components of WebSphere JMS support. This figure shows the main components of WebSphere JMS support, from JMS provider through a connection to a destination, then to a WebSphere enterprise application (acting as a JMS client) that processes the message retrieved from the destination. For more information, see the text that accompanies this figure.

WebSphere Application Server supports asynchronous messaging based on the Java Messaging Service (JMS) of a JMS provider that conforms to the JMS specification version 1.0.2 and supports the Application Server Facility (ASF) function defined within that specification. WebSphere Application Server provides an embedded JMS provider and administration objects for WebSphere MQ as the JMS provider. You can use the embedded JMS provider, install WebSphere MQ JMS on top of the embedded WebSphere JMS, or install and configure another JMS provider.

JMS functions (of JMS providers) within the WebSphere Application Server administration domain are served by one or more JMS servers. There can be at most one JMS server on each node in the administration domain, and any application server within the domain can access JMS resources served by any JMS server on any node in the domain.

A connection factory is used to create connections with the JMS provider for a specific JMS queue or topic destination. Each connection factory encapsulates the configuration parameters needed to create a connection to a JMS destination.

A WebSphere J2EE application can explicitly poll for messages on a destination then retrieve messages for processing by business logic beans (enterprise beans).

The WebSphere Application Server support for message-driven beans and extended messaging builds on this base JMS support. For more information, see the related topics.

WebSphere MQ JMS connection pooling

To improve the overall performance of JMS within the system, the message listener service enables the connection pooling facility provided by the WebSphere MQ JMS

implementation. This support does not affect the performance of a message listener, because it retains its connections while listening on a destination, but does affect the overall JMS system performance. When a connection is no longer required, WebSphere MQ can pool the connection then reuse it later instead of destroying it.

Note: This support is only available if WebSphere MQ is configured as the JMS provider.

To enable WebSphere MQ connection pooling and configure the characteristics of the WebSphere MQ connection pool, see Enabling WebSphere MQ JMS connection pooling.

Administering WebSphere JMS support

Use these tasks with the WebSphere administrative console to manage JMS providers and their resources, and other runtime components of WebSphere JMS support.

You can use the WebSphere administrative console to configure the embedded WebSphere JMS provider or an WebSphere MQ JMS provider. If you install another JMS provider, you need to configure that JMS provider by using the tools and information provided with the JMS provider. For each JMS provider, you can configure the properties of JMS resources.

You can also use the WebSphere administrative console to configure and control other runtime components of WebSphere JMS support, including the following:
- The WebSphere JMS server
- The message listener service, listener ports, and the listener for each message-driven bean
- Input and output ports for extended messaging.

You can update the configuration data at any time, but if it is updated, the updates only take effect when the appropriate server is next started.

For information about the specific tasks used to administer WebSphere JMS support, see the following topics:
- Installing and configuring a JMS provider
- Moving from the internal JMS provider to WebSphere MQ
- Enabling security for the embedded WebSphere JMS provider
- Displaying administrative lists of JMS resources
- Managing JMS servers on Application Server
- Managing JMS servers in a deployment manager cell
- Configuring JMS provider resources

For more information about JMS resources, see the JMS documentation at http://java.sun.com/products/jms/docs.html.

Installing and configuring a JMS provider

This topic describes the different ways that you can implement a JMS provider for use with WebSphere Application Server.

For IBM WebSphere Application Server to support bean-managed messaging, you need to install and configure one or more JMS providers that conform to the JMS specification version 1.0.2. To use message-driven beans the JMS provider must support the Application Server Facility (ASF) function defined within that specification.

You can install and use the Embedded Messaging Server option of WebSphere Application Server, install WebSphere MQ as the JMS provider, or install another "generic" JMS provider. If you install both embedded messaging and WebSphere MQ as JMS providers, for example, WebSphere applications can use JMS resources provided by both the embedded WebSphere JMS provider and the WebSphere MQ JMS provider.

WebSphere Application Server provides default JMS support with its Embedded Messaging Server component. This function is installed with WebSphere Application Server, administered through the administrative console and managed as part of the WebSphere Application Server runtime. This function is only accessible from WebSphere Application Server Web, EJB and client containers, and is not interoperable with WebSphere MQ. If access is required to heterogeneous non-JMS applications, WebSphere MQ clustering, or other WebSphere MQ functions, you should install WebSphere Application Server without the Embedded Messaging Server component, and should install WebSphere MQ instead as the JMS provider.

The WebSphere Application Server Embedded Messaging Client is the same Java Client feature that ships with WebSphere MQ 5.3.1.

- The Java Client is installed automatically by default with a standalone WebSphere Application Server for z/OS during ISPF dialog customization, regardless of whether you configure an embedded JMS provider.
- The Java Client remains installed in a standalone WebSphere Application Server for z/OS when that node is federated into a deployment manager cell.
- An application server can use the Java Client for point-to-point JMS messaging to full-function 5.2.0 and 5.3.0 WebSphere MQ queue managers in bindings mode. The support for these levels of queue managers does not offer res-auth=application security.
- An application server can use the Java Client for point-to-point JMS messaging to full-function 5.3.1 queue managers in both client or bindings mode. Applications deployed with res-auth=application specifications have that security property honored.
- The Java Client feature has been certified by IBM with WebSphere Application Server for z/OS to be J2EE 1.3 compliant when using a configured embedded JMS provider only. Although a full-function 5.3.1 queue manager can provide J2EE 1.3 compliant point-to-point messaging for WebSphere Application Server for z/OS, IBM has not certified the Java Client feature with this configuration.
- The Java Client feature can be configured to use the install root (SMP/E target) hfs files, by setting the appropriate environment variables as defined in the WebSphere MQ. For more information about configuring WebSphere MQ environment variables, see the *Using Java* publication that is delivered with WebSphere Application Server for z/OS.

To provide the supported levels of messaging, you should apply the latest service for WebSphere Application Server and WebSphere MQ while a production application is in Quality-Assurance or Product-Validation Test.

For more information about WebSphere Application Server messaging scenarios, and the relationship between embedded messaging and WebSphere MQ, see the following articles:

- WebSphere Application Server solutions White paper: Selecting the most appropriate JMS provider for your applications (ftp://ftp.software.ibm.com/software/websphere/resourcefinder/2318-00.pdf).

This white paper describes the different Java Message Service (JMS) options that are available from IBM for WebSphere Application Server applications. It compares the messaging technology built into WebSphere Application Server, Version 5 with IBM WebSphere MQ, IBM WebSphere Business Integration Event Broker and IBM WebSphere Business Integration Message Broker.

- JMS Topologies and Configurations with WebSphere Application Server and WebSphere Studio Version 5 (http://www7b.boulder.ibm.com/wsdd/library/techarticles/0310_barcia/barcia.html). This technical article helps you understand configuration options within WebSphere Application Server for Java Message Service (JMS) applications, including using embedded messaging and WebSphere MQ (formerly MQSeries). It covers setting up your development environment to support various JMS scenarios, development and runtime topologies, and scripting and security.

You can install and configure a JMS provider in one or more of the following ways:

- Installing WebSphere embedded messaging as the JMS provider.

  Note:
  - WebSphere embedded messaging as the JMS provider supports both queues (for point-to-point messaging) and topics (for publish/subscribe messaging).
  - You can install IBM WebSphere Application Server with embedded messaging on the same host as an existing WebSphere MQ installation, which must be at a supported level of WebSphere MQ features.
  - You can install IBM WebSphere Application Server with embedded messaging then later install WebSphere MQ for use as a JMS provider.
  - After installing WebSphere MQ and WebSphere Application Server on the same host, you should only service WebSphere MQ with its CSD updates, and separately service WebSphere Application Server with its fix packs. This approach helps to avoid potential failures if you later decide to uninstall WebSphere Application Server fix packs.

- Installing WebSphere MQ as the JMS provider.

  Note:
  - You can install WebSphere MQ before IBM WebSphere Application Server. If you then want to install embedded messaging, you must ensure that the WebSphere MQ installation is at a supported level of MQ features.
  - If you do not want to use the embedded WebSphere JMS provider, you can install IBM WebSphere Application Server without the **Embedded Messaging** options.
  - You can install WebSphere MQ for use as a JMS provider on top of WebSphere Application Server embedded messaging; this automatically uninstalls the installed WebSphere Embedded Messaging Server component, and results in a single WebSphere MQ Server installation that is used to provide JMS resources for both embedded messaging and WebSphere MQ.
    - For point-to-point messaging WebSphere applications can continue to use WebSphere queue resources (through the embedded messaging JMS provider) or WebSphere MQ queue resources that you define to IBM WebSphere Application Server.
    - For publish/subscribe messaging, WebSphere applications can continue to use WebSphere topic resources (through the embedded messaging JMS provider) or WebSphere MQ topic resources that you

define to IBM WebSphere Application Server, and which are provided by a Publish/Subscribe broker installed in addition to the base WebSphere MQ.

The preferred solution for publish/subscribe messaging with WebSphere MQ as the JMS provider is a full broker such as WebSphere MQ Event Broker. Failing this, you can install the MA0C SupportPac with WebSphere MQ.

- If you install WebSphere MQ as the JMS provider, you can use the WebSphere administrative console to administer the WebSphere MQ JMS provider resources, such as queue connection factories. However, you cannot administer MQ security, which is administered through WebSphere MQ.
- You can change from using WebSphere embedded messaging to WebSphere MQ as the JMS provider for WebSphere enterprise applications, as described in Moving from the embedded WebSphere JMS provider to WebSphere MQ.

For more information about scenarios and considerations for using WebSphere MQ with IBM WebSphere Application Server, see the White Papers and Red books provided by WebSphere MQ; for example, through the WebSphere MQ library Web page at http://www-3.ibm.com/software/ts/mqseries/library/

- Installing another JMS provider, which must conform to the JMS specification and, to use message-driven beans, support the ASF function. If you want to use a JMS provider other than the embedded WebSphere JMS provider or a WebSphere MQ JMS provider, you should complete the following steps:
  1. Installing and configuring the JMS provider and its resources by using the tools and information provided with the JMS provider.
  2. Defining the JMS provider to WebSphere Application Server as a generic JMS provider.

  Note: You cannot use the WebSphere administrative console to administer the JMS provider or its resources.

To install a JMS provider for IBM WebSphere Application Server, consider the following scenarios:

A new IBM WebSphere Application Server server machine, hostA.

This scenario starts with adding embedded messaging as the JMS provider, then optionally adding WebSphere MQ as an alternative JMS provider. Each stage summarizes the messaging functions that can be added.

1. Installing embedded messaging as the only JMS provider.

   You want to be able to run WebSphere applications that use the WebSphere JMS resources for both point-to-point and publish/subscribe messaging.

   a. Install and customize WebSphere Application Server with the **Embedded Messaging Server** and **Embedded Messaging Client** options.
   b. Use the administrative console to configure WebSphere JMS resources; for example, WebSphere Queue Connection Factories and WebSphere Topic Connection Factories.
   c. On any client machines that are to use the WebSphere JMS resources, install and customize WebSphere Application Server with the **Embedded Messaging Client** option.
2. Adding WebSphere MQ as an alternative JMS provider for point-to-point messaging.

Besides the point-to-point and publish/subscribe messaging that uses the embedded WebSphere JMS resources (from the preceding step), you want to use WebSphere MQ Queue resources for point-to-point messaging.

a. Install WebSphere MQ 5.3 with the required features. For more information about installing WebSphere MQ, see the WebSphere MQ Quick Beginnings book for your machine's platform. When installing WebSphere MQ, perform the following actions:
   1) When asked if you want to remove or modify the version of WebSphere MQ currently installed, choose to remove the Server. (The WebSphere MQ install program recognizes the installed WebSphere embedded messaging as an installed version of WebSphere MQ.)
   2) Select the option to keep existing queue managers.
   3) Proceed with the installation of the full WebSphere MQ Server by running the install program again, after the old WebSphere MQ server (the Embedded Messaging Server) has been removed.
b. Use the administrative console to configure WebSphere MQ Queue Connection Factories and WebSphere MQ Queue Destinations.

3. Adding WebSphere MQ Event Broker for alternative publish/subscribe messaging.

   For publish/subscribe messaging, you want to be able to run WebSphere applications that use the WebSphere MQ Topic resources or the embedded WebSphere Topic resources (such as those configured in preceding steps).

   a. Install WebSphere MQ Event Broker. For more information about installing WebSphere MQ Event Broker, see the WebSphere MQ Event Broker Installation Guide for your machine's platform.
   b. Use the administrative console to configure WebSphere MQ Topic Connection Factories and WebSphere MQ Topic Destinations.

An existing WebSphere MQ 5.2 server and broker machine, hostA, where you want to install embedded messaging as the JMS provider.

1. Upgrade to WebSphere MQ 5.3 with the required features. For more information about upgrading to WebSphere MQ 5.3, see "Migrating from an earlier version" in the WebSphere MQ Quick Beginnings book for your machine's platform.
2. To continue using publish/subscribe messaging, upgrade to a supported broker such as WebSphere MQ Event Broker.
3. Install and customize WebSphere Application Server with the **Embedded Messaging Server**. You do not need to install the Embedded Messaging Client, which is the same as installed with WebSphere MQ.
4. Use the administrative console to configure WebSphere JMS resources; for example, WebSphere Queue Connection Factories and WebSphere Topic Connection Factories.
5. If you want WebSphere applications to use the WebSphere MQ resources, use the administrative console to configure WebSphere MQ JMS resources; for example, WebSphere MQ Queue Connection Factories and WebSphere MQ Destinations.
6. On any client machines that are to use the WebSphere JMS resources, install and customize WebSphere Application Server. If the WebSphere MQ client is not installed, install **Embedded Messaging Client** option.

You can run WebSphere applications that use both the WebSphere JMS resources and WebSphere MQ JMS resources for messaging.

An existing WebSphere MQ server machine, hostA, where you want to use WebSphere MQ as the only JMS provider.

1. For point-to-point messaging, ensure that you have installed WebSphere MQ 5.3 with required features. For publish/subscribe messaging, ensure that you have also installed a supported broker such as WebSphere MQ Event Broker. For more information about installing WebSphere MQ 5.3 and Event Broker, see the WebSphere MQ Quick Beginnings book and WebSphere MQ Event Broker Installation Guide for your machine's platform.
2. Install and customize WebSphere Application Server without any of the **Embedded Messaging Server** and **Embedded Messaging Client** options.
3. Use the administrative console to configure WebSphere MQ JMS resources; for example, WebSphere MQ Queue Connection Factories and WebSphere MQ Topic Connection Factories.
4. On any client machines that are to use the WebSphere JMS resources, install and customize WebSphere Application Server. If the WebSphere MQ client is not installed, install the **Embedded Messaging Client** option.

   You can run WebSphere applications that use the WebSphere MQ JMS resources for point-to-point or (with a supported broker installed) publish/subscribe messaging.

Installing WebSphere MQ as the JMS provider:

Use this task to install and configure WebSphere MQ with support for the Java Message Service (JMS) for use with the WebSphere Application Server.

To install and configure WebSphere MQ (MQSeries) for use as a JMS provider to IBM WebSphere Application Server, complete the following steps:

1. Install WebSphere MQ 5.3, with the required MQ features, as described in the installation instructions provided with WebSphere MQ.

   If you are installing WebSphere MQ on top of WebSphere Application Server embedded messaging, perform the following actions when installing WebSphere MQ:

   a. When asked if you want to remove or modify the version of WebSphere MQ currently installed, choose to remove the Server. (The WebSphere MQ install program recognizes the installed WebSphere embedded messaging as an installed version of WebSphere MQ.)
   b. Select the option to keep existing queue managers.
   c. Proceed with the installation of the full WebSphere MQ Server by running the install program again, after the old WebSphere MQ server (the Embedded Messaging Server) has been removed.

   If you want to use the original WebSphere MQ 5.3 release, ensure that you install the CSD04 update.

   If you want to use WebSphere MQ 5.3 on the same machine as WebSphere Application Server embedded messaging, ensure that you install the following WebSphere MQ features:
   - For a WebSphere Application Server **Embedded Messaging Server** installation, the required MQ features are "Server" and "Java Messaging".
   - For a WebSphere Application Server **Embedded Messaging Client** installation, the only required MQ feature is "Java Messaging".

For information about installing WebSphere MQ 5.3, or migrating to WebSphere MQ 5.3 from an earlier release, see the appropriate WebSphere MQ *Quick Beginnings* book, as listed above.

2. If you want to use WebSphere MQ - Publish/Subscribe support, you need to provide a Publish/Subscribe broker.

   For example, you can do this by using either WebSphere MQ Event Broker or WebSphere MQ Integrator (formerly MQSeries Integrator). For more information about these products, see the following Web sites:
   - WebSphere MQ Event Broker Web site at http://www-4.ibm.com/software/ts/mqseries/platforms/#eventb
   - WebSphere MQ Integrator Web site at http://www-4.ibm.com/software/ts/mqseries/platforms/#integrator
3. Follow the WebSphere MQ 5.3 instructions for verifying your installation setup.
4. For AIX, see the WebSphere MQ 5.3 readme.txt for additional steps.
5. If you want to install IBM WebSphere Application Server on the same host as WebSphere MQ, and have not yet done so, install and customize WebSphere Application Server. If you do not want to use the embedded WebSphere JMS provider, you can install WebSphere MQ then install WebSphere Application Server without the **Embedded Messaging** options.
6. Set the MQJMS_LIB_ROOT environment variable to the directory where WebSphereMQ\Java\lib is installed. IBM WebSphere Application Server uses the MQJMS_LIB_ROOT to locate the WebSphere MQ libraries for the WebSphere MQ JMS Provider.

This task has installed WebSphere MQ for use as the JMS provider with WebSphere Application Server.

You can configure JMS resources to be provided by WebSphere MQ, by using the WebSphere administrative console to define WebSphere MQ resources.

After installing WebSphere MQ and WebSphere Application Server on the same host, you should only service WebSphere MQ with its CSD updates, and separately service WebSphere Application Server with its fix packs. This approach helps to avoid potential failures if you later decide to uninstall WebSphere Application Server fix packs. Also, if you apply a fix pack to WebSphere Application Server, specify *not to update* the Embedded Messaging feature.

Defining a generic JMS provider:

Use this task to define a new JMS provider to WebSphere Application Server, for use instead of the embedded WebSphere JMS provider or a WebSphere MQ JMS provider.

Before starting this task, you should have installed and configured the JMS provider and its resources by using the tools and information provided with the JMS provider.

To define a new generic JMS provider to WebSphere Application Server, use the administrative console to complete the following steps:

1. In the navigation pane, expand **Resources-> Generic JMS Providers** This displays the existing generic JMS providers in the content pane.

2. To define a new generic JMS provider, click **New** in the content pane. Otherwise, to change the definition of an existing JMS provider, click the JMS provider. This displays the properties used to define the JMS provider in the content pane.
3. Specify appropriate properties for the JMS provider.
4. Click **OK**.
5. To save your configuration, click **Save** on the task bar of the Administrative console window.
6. To have the changed configuration take effect, stop then restart the application server.

Installing WebSphere embedded messaging as the JMS provider:

Use this task to install the Embedded Messaging options of IBM WebSphere Application Server for use as the JMS provider.

Installing the WebSphere embedded messaging feature for WebSphere Application Server for z/OS V5 is done in two parts:
- The first part consists of the SMP/E install process and as such is documented in detail in the IBM WebSphere Application Server for z/OS V5 Program Directory.
- The second part consists of the customization process which is completed using the Customization Dialog. This process is documented in detail in the IBM WebSphere Application Server for z/OS V5 Installation and Customization Guide.

Both of these publications are readily available on the WebSphere Application Server for z/OS library page.

If you have installed the Embedded Messaging Server option on top of WebSphere MQ, the MQ command **setmqcap** is set to use parameter 0 instead of -1, which results in:
- Issuing a license-unit message to the MQ console window whenever a queue manager starts
- Writing a message to the MQ error log

To prevent this, after you have completed the installation of IBM WebSphere Application Server, issue the `setmqcap -1` command from a command line.

After installing WebSphere MQ and WebSphere Application Server on the same host, you should service WebSphere MQ independently of the WebSphere Application Server fix packs. This approach helps to avoid potential failures if you later decide to uninstall WebSphere Application Server fix packs.

Moving from the embedded WebSphere JMS provider to WebSphere MQ

Use this task to move from the embedded WebSphere JMS provider to WebSphere MQ as the provider of messaging services and resources for WebSphere enterprise applications.

To move from the embedded WebSphere JMS provider to WebSphere MQ as the provider of messaging services and resources for WebSphere enterprise applications, you need to install and configure a supported level of WebSphere MQ with the required MQ features.

Existing JMS resource definitions for the embedded WebSphere JMS provider continue to work with WebSphere MQ as the JMS provider, so you do not need to redefine those JMS resources. However, to take advantage of the extra configuration options for WebSphere MQ, you can use the administrative console to define new JMS resources as WebSphere MQ resources; for example, to define MQ Queue Connection Factories.

1. For WebSphere MQ point-to-point messaging, install the base WebSphere MQ product.
2. Configure WebSphere MQ queue resources to IBM WebSphere Application Server.
   - For point-to-point messaging WebSphere applications can continue to use WebSphere queue resources (through the embedded messaging JMS provider) or WebSphere MQ queue resources.
   - For publish/subscribe messaging, WebSphere applications can continue to use WebSphere topic resources (through the embedded messaging JMS provider).
3. For WebSphere MQ publish/subscribe messaging, install a Publish/Subscribe broker, such as WebSphere MQ Event Broker.

   For more information about installing WebSphere MQ Event Broker or WebSphere MQ Integrator (formerly MQSeries Integrator), see the following Web sites:
   - WebSphere MQ Event Broker Web site at http://www-4.ibm.com/software/ts/mqseries/platforms/#eventb
   - WebSphere MQ Integrator Web site at http://www-4.ibm.com/software/ts/mqseries/platforms/#integrator
4. Configure WebSphere MQ topic resources to IBM WebSphere Application Server. For publish/subscribe messaging, WebSphere applications can continue to use WebSphere topic resources (through the embedded messaging JMS provider) or WebSphere MQ topics.

Managing JMS servers on an Application Server node

Use this task to manage JMS servers on an Application Server node that is not part of a deployment manager cell.

On an Application Server node that is not part of a deployment manager cell, each application server has an internal JMS server that runs as part of the application server process and is administered as additional properties of the application server. A JMS server enables the application server to access JMS resources.

If you want to start a JMS server that is not part of a deployment manager cell, you start the associated application server.

A JMS server makes use of separate WebSphere MQ processes. If a WebSphere MQ process fails and restarts while the JMS server is running, you must restart the application server. This shutdown also can affect other installed applications.

Before starting the application server, ensure that the required WebSphere MQ messages are not being suppressed by the message processing facility (MPF). For more information about starting application servers, see the Information Center topic Starting servers.

You can use the WebSphere administrative console to configure a general set of JMS server properties, which add to the default values of properties configured automatically for the embedded WebSphere JMS provider.

To configure JMS server properties, use the administrative console to complete the following steps:

1. In the navigation pane, select **Servers-> Application Servers** This displays a table of the application servers in the administrative domain.
2. In the content pane, click the name of the application server. This displays the properties of the application server in the content pane.
3. In the content pane, under Additional Properties, select **Server components-> JMS Server** This displays the JMS server properties in the content pane.
4. Specify appropriate properties for the JMS server.

   If you want the JMS server to be started automatically when the application server is next started, set the **Initial state** property to `started`.

   If you want to add a new queue to be hosted by the JMS server, add the administrative name of the queue to the Queue Names field. (The name must match the name of a WebSphere Queue administrative object, including the use of upper- and lowercase.) Similarly, if you want to remove a queue from the JMS server, remove its name from that field.
5. Click **OK**.
6. To save your configuration, click **Save** on the task bar of the administrative console window.
7. To have the changed configuration take effect, stop then restart the application server.

Managing JMS servers in a deployment manager cell

Use this task to manage JMS servers on nodes in a WebSphere Application Server deployment manager cell.

Before starting a JMS server, ensure that the following WebSphere MQ messages are not being suppressed by the message processing facility (MPF):

```
CSQV086E CSQY022I CSQY003I CSQX022I CSQM132I CSQ9022I CSQX017I CSQ3104I CSQ3106E
```

In a WebSphere Application Server deployment manager cell, each node can have at most one JMS server, and any application server within the cell can access JMS resources served by any of those JMS servers.

You can use the WebSphere administrative console to display a list of all JMS servers, to show and control their runtime status. You can also configure a general set of JMS server properties, which add to the default values of WebSphere MQ properties configured automatically for the embedded WebSphere JMS provider.

Note: In general, the default values of WebSphere MQ properties are adequate for WebSphere internal JMS servers. However, if you are running high messaging loads, you may need to change some WebSphere MQ properties; for example, WebSphere MQ properties for log file locations, file pages, and buffer pages. For more information about configuring WebSphere MQ properties, see the *WebSphere MQ System Administration* book, SC33-1873, which is available from the IBM Publications Center or from the WebSphere MQ collection kit, SK2T-0730.

To manage a WebSphere internal JMS server, use the administrative console to complete the following steps:

1. In the navigation pane, select **Servers-> JMS Servers** This displays a table of the JMS servers, showing their runtime status.

2. If you want to change the runtime status of a JMS server, complete the following steps:
   a. In the table of JMS servers, select the JMS servers that you want to act on.
      - To act on one or more specific JMS servers, select the checkbox next to the JMS server name.
      - To act on all JMS servers, select the checkbox next to the JMS servers title of the table.
   b. Click one of the actions displayed to change the status of the JMS servers; for example, click **Stop** to stop a JMS server.

   The status of the JMS servers that you have acted on is updated to show the result of your actions.
3. If you want to change the properties of a JMS server, click the name of the JMS server. This displays the properties of the JMS server in the content pane.
4. Specify appropriate properties for the JMS server.

   If you want to add a new queue to be hosted by the JMS server, add the administrative name of the queue to the Queue Names field. (The name must match the name of a WebSphere Queue administrative object, including the use of upper- and lowercase.) Similarly, if you want to remove a queue from the JMS server, remove its name from that field.
5. Click **OK**.
6. To save your configuration, click **Save** on the task bar of the Administrative console window.
7. To have the changed configuration take effect, stop then restart the JMS Server.

Configuring JMS provider resources

Use the following tasks to configure JMS provider resources needed to support enterprise beans that exploit JMS services.

- Configuring resources for the embedded WebSphere JMS provider
  - Configuring a queue connection factory
  - Configuring a topic connection factory
  - Configuring a queue destination
  - Configuring a topic destination
- Configuring resources for the WebSphere MQ JMS provider
  - Configuring a queue connection factory
  - Configuring a topic connection factory
  - Configuring a queue destination
  - Configuring a topic destination
- Configuring resources for a generic JMS provider
  - Configuring a JMS connection factory
  - Configuring a JMS destination

Configuring resources for the embedded WebSphere JMS provider:

Use the following tasks to configure the connection factories and destinations for the embedded WebSphere JMS provider.

You only need to complete these tasks if your WebSphere Application Server environment uses the embedded WebSphere JMS provider to support enterprise applications that use JMS.

- Configuring a queue connection factory
- Configuring a topic connection factory
- Configuring a queue destination
- Configuring a topic destination

Configuring a queue connection factory, embedded WebSphere JMS provider:

Use this task to configure the properties of a queue connection factory for use with the embedded WebSphere JMS provider. This task contains an optional step for you to create a new queue connection factory.

To configure the properties of a queue connection factory for use with the embedded WebSphere JMS provider, use the administrative console to complete the following steps:

1. In the navigation pane, click **Resources-> WebSphere JMS Provider** This displays in the content pane a table of properties for the WebSphere JMS provider, including links to the types of JMS resources supported by the JMS provider.
2. Optional: Change the **Scope** check box to Cell, Node, or Server, according to your needs.
3. In the content pane, under Additional Properties, click **WebSphere Queue Connection Factories** This displays any existing queue connection factories for the WebSphere JMS provider in the content pane.
4. To create a new queue connection factory, click **New** in the content pane. Otherwise, to change the properties of an existing queue connection factory, click one of the connection factories displayed. This displays the properties for the queue connection factory in the content pane.
5. Specify appropriate properties for the queue connection factory.
6. Click **OK**.
7. To save your configuration, click **Save** on the task bar of the Administrative console window.
8. To have the changed configuration take effect, stop then restart the application server.

Configuring a topic connection factory, embedded WebSphere JMS provider:

Use this task to configure the properties of a topic connection factory for use with the embedded WebSphere JMS provider. This task contains an optional step for you to create a new topic connection factory.

To configure the properties of a topic connection factory for use with the embedded WebSphere JMS provider, use the administrative console to complete the following steps:

1. In the navigation pane, click **Resources-> WebSphere JMS Provider** This displays in the content pane a table of properties for the WebSphere JMS provider, including links to the types of JMS resources supported by the JMS provider.
2. Optional: Change the **Scope** check box to Cell, Node, or Server, according to your needs.
3. In the content pane, under Additional Properties, click **WebSphere Topic Connection Factories** This displays any existing topic connection factories for the WebSphere JMS provider in the content pane.
4. To create a new topic connection factory, click **New** in the content pane. Otherwise, to change the properties of an existing topic connection factory, click one of the connection factories displayed. This displays the properties for the topic connection factory in the content pane.
5. Specify appropriate properties for the topic connection factory.
6. Click **OK**.

7. To save your configuration, click **Save** on the task bar of the Administrative console window.
8. To have the changed configuration take effect, stop then restart the application server.

Configuring a queue destination, embedded WebSphere JMS provider:

Use this task to configure the properties of a queue destination for use with the embedded WebSphere JMS provider. This task contains an optional step for you to create a new queue destination.

To optimize performance, configure the queue destination properties to best fit your applications. You should also consider queue attributes of the internal JMS server that are associated with the queue name. For more information, see the Information Center topic "Performance considerations for WebSphere queue destinations."

To configure the properties of a queue destination for use with the embedded WebSphere JMS provider, use the administrative console to complete the following steps:

1. In the navigation pane, expand **Resources-> WebSphere JMS Provider** This displays in the content pane a table of properties for the WebSphere JMS provider, including links to the types of JMS resources supported by the JMS provider.
2. In the content pane, under Additional Properties, click **WebSphere Queue Destinations** This displays any existing queue destinations for the WebSphere JMS provider in the content pane.
3. To create a new queue destination, click **New** in the content pane. Otherwise, to change the properties of an existing queue destination, click one of the destinations displayed. This displays the properties for the queue destination in the content pane.
4. Specify appropriate properties for the queue destination.
5. Click **OK**.
6. To save your configuration, click **Save** on the task bar of the Administrative console window.
7. To make a queue destination available to applications, you need to host the queue on a JMS server. To add a new queue to a JMS server or to change an existing queue on a JMS server, you define the administrative name of the queue to the JMS server, as described in Managing JMS servers in a deployment manager cell.
8. To have the changed configuration take effect, stop then restart the application server.

Performance considerations for WebSphere queue destinations: To optimize performance, configure the queue destination properties to best fit your applications. For example, setting the Expiry property to SPECIFIED and the Specified Expiry property to 30000 milliseconds for the expiry timeout, reduces the number of messages that can be queued. To ensure that there are enough underlying WebSphere MQ resources available for the queue, you must ensure that you configure the queue destination properties adequately for your application usage.

You should also consider queue attributes of the internal JMS server that are associated with the queue name. Inappropriate queue attributes can reduce the performance of WebSphere operations.

BOQNAME
The excessive backout requeue name. This can be set to a local queue name that can hold the messages which were rolled back by the WebSphere applications. This queue name can be a system dead letter queue.

BOTHRESH
The backout threshold and can be set to a number once the threshold is reached, the message will be moved to the queue name specified in BOQNAME.

For more information about using these properties, see:

- "Handling poison messages" in the *WebSphere MQ Using Java* book
- The *WebSphere MQ Script (MQSC) Command Reference* book

Configuring a topic destination, embedded WebSphere JMS provider:

Use this task to configure the properties of a topic destination for use with the embedded WebSphere JMS provider. This task contains an optional step for you to create a new topic destination.

To optimize performance, configure the topic destination properties to best fit your applications. For more information, see the topic "Performance considerations for WebSphere topic destinations."

To configure the properties of a topic destination for use with the embedded WebSphere JMS provider, use the administrative console to complete the following steps:

1. In the navigation pane, click **Resources-> WebSphere JMS Provider** This displays in the content pane a table of properties for the WebSphere JMS provider, including links to the types of JMS resources supported by the JMS provider.
2. Optional: Change the **Scope** check box to Cell, Node, or Server, according to your needs.
3. In the content pane, under Additional Properties, click **WebSphere Topic Destinations** This displays any existing topic destinations for the WebSphere JMS provider in the content pane.
4. To create a new topic destination, click **New** in the content pane. Otherwise, to change the properties of an existing topic destination, click one of the destinations displayed. This displays the properties for the topic destination in the content pane.
5. Specify appropriate properties for the topic destination.
6. Click **OK**.
7. To save your configuration, click **Save** on the task bar of the Administrative console window.
8. To have the changed configuration take effect, stop then restart the application server.

Performance considerations for WebSphere topic destinations: To optimize performance, configure the queue destination properties to best fit your applications. For example, setting the Expiry property to SPECIFIED and the Specified Expiry property to 30000 milliseconds for the expiry timeout, reduces the number of

messages that can be queued. To ensure that there are enough underlying WebSphere MQ resources available for the queue, you must ensure that you configure the queue destination properties adequately for your application usage.

- Ensure the queue attribute, INDXTYPE is set to MSGID for the following system queues:
  - SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE
  - SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE
- Ensure the queue attribute, INDXTYPE is set to CORRELID for the following system queues:
  - SYSTEM.JMS.ND.SUBSCRIBER.QUEUE
  - SYSTEM.JMS.D.SUBSCRIBER.QUEUE

For more information about using these properties, see:

- The *WebSphere MQ Using Java* book
- The *WebSphere MQ Script (MQSC) Command Reference* book

Configuring resources for the WebSphere MQ JMS provider:

Use the following tasks to configure the connection factories and destinations for the WebSphere MQ JMS provider.

You only need to complete these tasks if your WebSphere Application Server environment uses the WebSphere MQ JMS provider to support enterprise applications that use JMS. To enable use of the WebSphere MQ JMS provider, you must have installed and configured WebSphere MQ JMS support, as described in Installing and configuring WebSphere MQ as the JMS provider.

- Configuring a queue connection factory
- Configuring a topic connection factory
- Configuring a queue destination
- Configuring a topic destination
- Enabling WebSphere MQ JMS connection pooling

Configuring a queue connection factory, WebSphere MQ JMS provider:

Use this task to configure the properties of a queue connection factory for use with the WebSphere MQ JMS provider. This task contains an optional step for you to create a new queue connection factory.

To configure the properties of a queue connection factory for use with the WebSphere MQ JMS provider, use the administrative console to complete the following steps:

1. In the navigation pane, click **Resources-> WebSphere MQ JMS Provider** This displays in the content pane a table of properties for the WebSphere JMS provider, including links to the types of JMS resources supported by the JMS provider.
2. Optional: Change the **Scope** check box to Cell, Node, or Server, according to your needs.
3. In the contents pane, under Additional Properties, click **WebSphere MQ Queue Connection Factories** This displays a table listing any existing queue connection factories, with a summary of their properties.
4. To create a new queue connection factory, click **New** in the content pane. Otherwise, to change the properties of an existing queue connection factory,

click one of the connection factories displayed. This displays the properties for the queue connection factory in the content pane.

5. Specify appropriate properties for the queue connection factory.
6. 5.1+ Optional: Specify any of the following WebSphere MQ Secure Sockets Layer (SSL) properties that you need, as **Custom properties** of the connection factory: SSLPEERNAME, SSLCRL, and SSLCIPHERSUITE.

   For more information about these custom properties, see Custom properties.
7. Click **OK**.
8. To save your configuration, click **Save** on the taskbar of the Administrative console window.
9. To have the changed configuration take effect, stop then restart the application server.

Configuring a topic connection factory, WebSphere MQ JMS provider:

Use this task to configure the properties of a topic connection factory for use with the WebSphere MQ JMS provider. This task contains an optional step for you to create a new topic connection factory.

To configure the properties of a topic connection factory for use with the WebSphere MQ JMS provider, use the administrative console to complete the following steps:

1. In the navigation pane, click **Resources-> WebSphere MQ JMS Provider** This displays in the content pane a table of properties for the WebSphere JMS provider, including links to the types of JMS resources supported by the JMS provider.
2. Optional: Change the **Scope** check box to Cell, Node, or Server, according to your needs.
3. In the contents pane, under Additional Properties, click **WebSphere MQ Topic Connection Factories** This displays a table listing any existing topic connection factories, with a summary of their properties.
4. To create a new topic connection factory, click **New** in the content pane. Otherwise, to change the properties of an existing topic connection factory, click one of the connection factories displayed. This displays the properties for the topic connection factory in the content pane.
5. Specify appropriate properties for the topic connection factory.
6. 5.1+ Optional: Specify any of the following WebSphere MQ properties that you need, as **Custom properties** of the connection factory: SSLPEERNAME, SSLCRL, SSLCIPHERSUITE, MSGSELECTION, and SUBSTORE.

   For more information about these custom properties, see Custom properties.
7. Click **OK**.
8. To save your configuration, click **Save** on the taskbar of the Administrative console window.
9. To have the changed configuration take effect, stop then restart the application server.

Configuring a queue destination, WebSphere MQ JMS provider:

Use this task to configure the properties of a queue destination for use with the WebSphere MQ JMS provider. This task contains an optional step for you to create a new queue destination.

To optimize performance, configure the queue destination properties to best fit your applications. You should also consider queue attributes of the internal JMS server that are associated with the queue name. For more information, see "Performance considerations for WebSphere MQ queue destinations."

To configure the properties of a queue destination for use with the WebSphere MQ JMS provider, use the administrative console to complete the following steps:

1. In the navigation pane, click **Resources-> WebSphere MQ JMS Provider** This displays in the content pane a table of properties for the WebSphere JMS provider, including links to the types of JMS resources supported by the JMS provider.
2. Optional: Change the **Scope** check box to Cell, Node, or Server, according to your needs.
3. In the contents pane, under Additional Properties, click **WebSphere MQ Queue Destinations** This displays a table listing any existing queue destinations, with a summary of their properties.
4. To define a new queue destination, click **New** in the content pane. Otherwise, to change the properties of an existing queue destination, click one of the destinations displayed. This displays the properties for the queue destination in the content pane.
5. Configure appropriate properties for the queue destination.
6. Optional: If you want WebSphere Application Server to try to use the WebSphere MQ queue manager's remote administration utilities to create the queue, configure the WebSphere MQ Queue Connection properties.

   If you have already created your underlying queue in WebSphere MQ using its administration tools (such as runmqsc or MQ Explorer), you do not need to configure any of the WebSphere MQ Queue Connection properties. You only need to configure these properties if you want WebSphere Application Server to try to use the WebSphere MQ queue manager's remote administration utilities to create the queue.

   Note: For any changes to these properties to take effect on the queue manager, the WebSphere MQ Queue Manager on which the queue resides (or will reside) must be configured for remote administration and be running.

   For more details about these properties, see the Information Center topic "WebSphere MQ config properties for the queue destination."
7. Click **Apply**.
8. To save your configuration, click **Save** on the task bar of the Administrative console window.
9. To have the changed configuration take effect, stop then restart the application server.

Performance considerations for WebSphere MQ queue destinations: To optimize performance, configure the queue destination properties to best fit your applications. For example, setting the Expiry property to SPECIFIED and the Specified Expiry property to 30000 milliseconds for the expiry timeout, reduces the number of messages that can be queued. To ensure that there are enough underlying WebSphere MQ resources available for the queue, you must ensure that you configure the queue destination properties adequately for your application usage.

You should also consider queue attributes of the internal JMS server that are associated with the queue name. Inappropriate queue attributes can reduce the performance of WebSphere operations.

You should also consider the queue attributes associated with the queue name you created with WebSphere MQ. Inappropriate queue attributes can reduce the performance of WebSphere operations. You can use WebSphere MQ commands to change queue attributes for the queue name.

BOQNAME
The excessive backout requeue name. This can be set to a local queue name that can hold the messages which were rolled back by the WebSphere applications. This queue name can be a system dead letter queue.

BOTHRESH
The backout threshold and can be set to a number once the threshold is reached, the message will be moved to the queue name specified in BOQNAME.

INDXTYPE
Set this to MSGID. This causes an index of message identifiers to be maintained, which can improve WebSphere MQ retrieval of messages.

DEFSOPT
Set this to SHARED (for shared input from the queue).

SHARE
This must be specified (so that multiple applications can get messages from this queue).

For more information about using these properties, see:

- For BOQNAME and BOTHRESH, see "Handling poison messages" in the *WebSphere MQ Using Java* book
- The *WebSphere MQ Script (MQSC) Command Reference* book

Configuring a topic destination, WebSphere MQ JMS provider:

Use this task to configure the properties of a topic destination for use with the WebSphere MQ JMS provider. This task contains an optional step for you to create a new topic destination.

To optimize performance, configure the topic destination properties to best fit your applications. For more information, see "Performance considerations for WebSphere MQ topic destinations" on page 221.

To configure the properties of a topic destination for use with the WebSphere MQ JMS provider, use the administrative console to complete the following steps:

1. In the navigation pane, click **Resources-> WebSphere MQ JMS Provider** This displays in the content pane a table of properties for the WebSphere JMS provider, including links to the types of JMS resources supported by the JMS provider.
2. Optional: Change the **Scope** check box to Cell, Node, or Server, according to your needs.
3. In the content pane, under Additional Properties, click **WebSphere MQ Topic Destinations** This displays a table listing any existing topic destinations, with a summary of their properties.

4. To create a new topic destination, click **New** in the content pane. Otherwise, to change the properties of an existing topic destination, click one of the destinations displayed. This displays the properties for the topic destination in the content pane.
5. Specify appropriate properties for the topic destination.
6. Click **OK**.
7. To save your configuration, click **Save** on the task bar of the Administrative console window.
8. To have the changed configuration take effect, stop then restart the application server.

Performance considerations for WebSphere MQ topic destinations: To optimize performance, configure the topic destination properties to best fit your applications. For example, setting the Expiry property to SPECIFIED and the Specified Expiry property to 30000 milliseconds for the expiry timeout, reduces the number of messages that can be queued. To ensure that there are enough underlying WebSphere MQ resources available for the queue, you must ensure that you configure the queue destination properties adequately for your application usage.

- Ensure the queue attribute, INDXTYPE is set to MSGID for the following system queues:
  - SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE
  - SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE
- Ensure the queue attribute, INDXTYPE is set to CORRELID for the following system queues:
  - SYSTEM.JMS.ND.SUBSCRIBER.QUEUE
  - SYSTEM.JMS.D.SUBSCRIBER.QUEUE

For more information about using these properties, see:

- The *WebSphere MQ Using Java* book
- The *WebSphere MQ Script (MQSC) Command Reference* book

Configuring WebSphere MQ JMS connection pooling:

Use this task to configure properties of WebSphere MQ JMS connection pooling.

To enable WebSphere MQ JMS connection pooling, complete the following steps:

1. Start the WebSphere Administrative console.
2. In the navigation pane, select **Servers-> Application Servers-> *your_app_server*** This displays the properties of the application server, *your_app_server*, in the content pane.
3. In the Additional Properties table, select **Message Listener Service properties** This displays the Message Listener Service properties in the content pane.
4. Select Custom Properties, then add the following properties:
   mqjms.pooling.threshold
   The maximum number of unused connections in the pool.
   mqjms.pooling.timeout
   The timeout in milliseconds for unused connections in the pool.
5. Click **OK**.
6. To save your configuration, click **Save** on the task bar of the administrative console window.

7. To have the changed configuration take effect, stop then restart the application server.

Configuring resources for a generic JMS provider:

Use the following tasks to configure the connection factories and destinations for a generic JMS provider (not the embedded WebSphere JMS provider or the WebSphere MQ JMS provider).

You only need to complete these tasks if your WebSphere Application Server environment uses another JMS provider to support enterprise applications that use JMS. To enable use of another JMS provider, you must have installed and configured the JMS provider, as described in Defining a new JMS provider to WebSphere Application Server.

- Configuring a JMS connection factory
- Configuring a JMS destination

Configuring a JMS connection factory, generic JMS provider:

Use this task to configure the properties of a JMS connection factory for use with a generic JMS provider other than the embedded WebSphere JMS provider or WebSphere MQ.

To configure the properties of a JMS connection factory for use with a generic JMS provider, use the administrative console to complete the following steps:

1. In the navigation pane, click **Resources-> Generic JMS Provider** This displays in the content pane a table of properties for the WebSphere JMS provider, including links to the types of JMS resources supported by the JMS provider.
2. Optional: Change the **Scope** check box to Cell, Node, or Server, according to your needs.
3. In the Additional Properties list in the contents pane, select **JMS Connection Factories** This displays a table listing any existing JMS connection factories, with a summary of their properties.
4. To create a new JMS connection factory, click **New** in the content pane. Otherwise, to change the properties of an existing JMS connection factory, click one of the connection factories displayed. This displays the properties for the JMS connection factory in the content pane.
5. Specify appropriate properties for the JMS connection factory.
6. Click **OK**.
7. To save your configuration, click **Save** on the task bar of the Administrative console window.
8. To have the changed configuration take effect, stop then restart the application server.

Configuring a JMS destination, a generic JMS provider:

Use this task to configure the properties of a JMS destination for use with a generic JMS provider (other than the embedded WebSphere JMS provider or the WebSphere MQ JMS provider).

To configure the properties of a JMS destination for use with a generic JMS provider, use the administrative console to complete the following steps:

1. In the navigation pane, click **Resources-> Generic JMS Providers** This displays in the content pane a list of any existing generic JMS providers.

2. Optional: Change the **Scope** check box to Cell, Node, or Server, according to your needs.
3. In the content pane, click the JMS provider that you want to support the JMS destination. This displays in the content pane a table of properties for the JMS provider, including links to the types of JMS resources supported by the JMS provider.
4. In the Additional Properties list in the contents pane, select **JMS Destinations** This displays a table listing any existing JMS destinations, with a summary of their properties.
5. To create a new JMS destination, click **New** in the content pane. Otherwise, to change the properties of an existing queue destination, click one of the destinations displayed. This displays the properties for the JMS destination in the content pane.
6. Specify appropriate properties for the JMS destination.
7. Click **OK**.
8. To save your configuration, click **Save** on the task bar of the Administrative console window.
9. To have the changed configuration take effect, stop then restart the application server.

Configuring authorization security for the embedded WebSphere JMS provider

Use this task to configure authorization security for the embedded WebSphere JMS provider.

To configure authorization security for the embedded WebSphere JMS provider complete the following steps.

Note: Security for the embedded WebSphere JMS provider is enabled when you enable global security for WebSphere Application Server. For more information about enabling global security, see the Information Center topic Managing secured applications.

1. Configure authorization settings to access JMS resources owned by the embedded WebSphere JMS provider. Authorization to access JMS resources owned by the embedded WebSphere JMS provider is controlled by settings in the *WAS_PUBSUB_ROOT*/WMQX/config/integral-jms-authorizations.xml file.

   Where *WAS_PUBSUB_ROOT* is the directory where the WebSphere Embedded Publish and Subscribe broker (WEMPS) is installed.

   The settings grant or deny authenticated userids access to internal JMS provider resources (queues or topics). As supplied, the integral-jms-authorisations.xml file grants the following permissions:
   - Read and write permissions to all queues.
   - Publish, subscribe, and persist to all topics.

   To configure authorization settings, edit the integral-jms-authorisations.xml file according to the information in this topic and in that file.
2. Edit the queue-admin-userids element to create a list of userids with administrative access to all queues. Administrative access is needed to create queues and perform other administrative activities on queues. For example, consider the following queue-admin-userids section:

```
<queue-admin-userids>
    <userid>adminid1</userid>
    <userid>adminid2</userid>
</queue-admin-userids>
```

In this example the userids adminid1 and adminid2 are defined to have administrative access to all queues.

3. Edit the queue-default-permissions element to define the default queue access permissions. These permissions are used for queues for which you do not define specific permissions (in queue sections). If this section is not specified, then access permissions exist only for those queues for which you have specifically created queue elements.

   For example, consider the following queue-default-permissions element:

   ```
   <queue-default-permissions>
     <permission>write</permission>
   </queue-default-permissions>
   ```

   In this example the default access permission for all queues is **write**. This can be overridden for a specific queue by creating a queue element that sets its access permission to **read**.

4. If you want to define specific access permissions for a queue, create a queue element, then define the following elements:

   For example, consider the following queue element:

   ```
   <queue>
     <name>q1</name>
     <public>
     </public>
     <authorize>
       <userid>useridr</userid>
       <permission>read</permission>
     </authorize>
     <authorize>
       <userid>useridw</userid>
       <permission>write</permission>
     </authorize>
     <authorize>
       <userid>useridrw</userid>
       <permission>read</permission>
       <permission>write</permission>
     </authorize>
   </queue>
   ```

   In this example for the queue q1, the userid useridr has read permission, the userid useridw has write permission, the userid useridrw has both read and write permissions, and all other userids have no access permissions (<public></public>).

5. Edit topic elements to define the access permissions for publish/subscribe topic destinations.

   For topics, you can grant and deny access permissions. Full permission inheritance is supported on topics. If you do not define specific access permissions for a userid on a specific topic then permissions are inherited first from the public permissions on that topic then from the parent topic. The inheritance of access permissions continues until the root topic from which the root permissions are assumed.

   a. If you want to define default access permissions for the root topic, edit a topic element with an empty name element. If you omit such a topic section, topics have no default topic permissions other than those defined by specific topic elements. For example, consider the following topic element for the root topic:

```
<topic>
   <name></name>
   <public>
     <permission>+pub</permission>
   </public>
 </topic>
```

In this example, the default access permission for all topics is set to publish. This can be overridden by other topic elements for specific topic names.

b. If you want to define access permissions for a specific topic, create a topic element with the name for the topic then define the access permissions in the public and authorize elements of the topic element. For example, consider the following topic section:

```
<topic>
  <name>a/b/c</name>
  <public>
    <permission>+sub</permission>
  </public>
  <authorize>
    <userid>useridpub</userid>
    <permission>+pub</permission>
  </authorize>
</topic>
```

In this example, the subscribe permission is granted to anyone accessing any topic whose name starts with `a/b/c`. Also, the userid useridpub is granted publish permission for any topic whose name starts with `a/b/c`.

6. Save the integral-jms-authorizations.xml file.

If the dynamic update setting is selected, changes to the integral-jms-authorizations.xml file become active when the changed file is saved, so there is no need to stop and restarted the JMS server. If the dynamic update setting is not selected, you need to stop and restart the JMS server to make changes active.

Dynamic updating is available, by ensuring proper tagging in the integral-jms-authorizations.xml file `<dyanmic-update>true</dynamic-update>`.

Authorization settings for embedded WebSphere JMS provider resources:

Use the integral-jms-authorisations.xml file to view or change the authorization settings for JMS resources owned by the embedded WebSphere JMS provider.

Authorization to access JMS resources owned by the embedded WebSphere JMS provider is controlled by the following settings in the *wempspath*/*wempsname*/`config/integral-jms-authorizations.xml` file.

This structure of the settings in integral-jms-authorisations.xml is shown in the following example. Descriptions of these settings are provided after the example. To configure authorization settings, follow the instructions provided in Configuring authorization security for the embedded WebSphere JMS provider

```
<integral-jms-authorizations>

  <dynamic-update>true</dynamic-update>

  <queue-admin-userids>
    <userid>adminid1</userid>
    <userid>adminid2</userid>
  </queue-admin-userids>

  <queue-default-permissions>
    <permission>write</permission>
```

```
  </queue-default-permissions>

  <queue>
    <name>q1</name>
    <public>
    </public>
    <authorize>
      <userid>useridr</userid>
      <permission>read</permission>
    </authorize>
    <authorize>
      <userid>useridw</userid>
      <permission>write</permission>
    </authorize>
  </queue>

  <queue>
    <name>q2</name>
    <public>
      <permission>write</permission>
    </public>
    <authorize>
      <userid>useridr</userid>
      <permission>read</permission>
    </authorize>
  </queue>

  <topic>
    <name></name>
    <public>
      <permission>+pub</permission>
    </public>
  </topic>

  <topic>
    <name>a/b/c</name>
    <public>
      <permission>+sub</permission>
    </public>
    <authorize>
      <userid>useridpub</userid>
      <permission>+pub</permission>
    </authorize>
  </topic>

</integral-jms-authorizations>
```

dynamic-update: Controls whether or not the JMS Server checks dynamically for updates to this file.

true (Default) Enables dynamic update support.

false Disables dynamic update checking and improves authorization performance.

queue-admin-userids: This element lists those userids with administrative access to all WebSphere queue destinations. Administrative access is needed to create queues and perform other administrative activities on queues. You define each userid within a separate userid sub element:

<userid>*adminid***</userid>**

Where *adminid* is a user ID that can be authenticated by IBM WebSphere Application Server.

queue-default-permissions: This element defines the default queue access permissions that are assumed if no permissions are specified for a specific queue name. These permissions are used for queues for which you do not define specific

permissions (in queue elements). If this element is not specified, then no access permissions exist unless explicitly authorized for individual queues.

You define the default permission within a separate permission sub element:
<permission>*read-write***</permission>**

Where *read-write* is one of the following keywords:

read By default, userids have read access to WebSphere queue destinations.

write By default, userids have write access to WebSphere queue destinations.

queue: This element contains the following authorization settings for a single queue destination:

name The name of the queue.

public The default public access permissions for the queue. This is used only for those userids that have no specific authorize element. If you leave this element empty, or do not define it at all, only those userids with authorize elements can access the queue.

You define each default permission within a separate permission element.

authorize

The access permissions for a specific userid. Within each authorize element, you define the following elements:

userid The userid that you want to assign a specific access permission.

permission

An access permission for the associated userid.

You define each permission within a separate permission element. Each permission element can contain the keyword read or write to define the access permission.

For example, consider the following queue element:

```
<queue>
  <name>q1</name>
  <public>
  </public>
  <authorize>
    <userid>useridr</userid>
    <permission>read</permission>
  </authorize>
  <authorize>
    <userid>useridw</userid>
    <permission>write</permission>
  </authorize>
  <authorize>
    <userid>useridrw</userid>
    <permission>read</permission>
    <permission>write</permission>
  </authorize>
</queue>
```

topic: This element contains the following authorization settings for a single topic destination:

Each topic element has the following sub elements:

name The name of the topic, without wildcards or other substitution characters.

public The default public access permissions for the topic. This is used only for those userids that have no specific authorize element. If you leave this element empty, or do not define it at all, only those userids with authorize elements can access the topic.

You define each default permission within a separate permission element.

authorize
The access permissions for a specific userid. Within each authorize element, you define the following elements:

userid The userid that you want to assign a specific access permission.

permission
An access permission for the associated userid.

You define each permission within a separate permission element. Each permission element can contain one of the following keywords to define the access permission:

+pub Grant publish permission
+sub Grant subscribe permission
+persist
Grant persist permission
-pub Deny publish permission
-sub Deny subscribe permission
-persist
Deny persist permission

Displaying administrative lists of JMS resources

Use this task with the WebSphere administrative console to display administrative lists of JMS resources.

You can use the WebSphere administrative console to display lists of the following types of JMS resources. You can use the panels displayed to select JMS resources to administer, or to create or delete JMS resources (where appropriate).

To display administrative lists of JMS resources, complete the following general steps:

1. Start the WebSphere administrative console.
2. In the navigation pane, expand the appropriate path to select the type of JMS provider (as shown in the following table).
3. If appropriate, in the content pane, select a specific JMS provider. This displays the properties for the JMS provider, and an Additional Properties list of links to the types of JMS resources provided.
4. In the content pane, under Additional Resources, select the link for the type of JMS resource. This displays a list of the selected JMS resource type in the content pane.

Table 4. Network Deployment - Administrative panels for JMS resources

Path	Panel	Description
Servers-> JMS servers	JMS servers	List all JMS servers within the administration domain
Servers-> JMS servers-> *server_name*	JMS servers	List properties of the selected JMS server *server_name*
Embedded WebSphere JMS providers		
Resources-> WebSphere JMS Provider	WebSphere JMS providers	List all WebSphere JMS providers within the administration domain

Table 4. Network Deployment - Administrative panels for JMS resources (continued)

Path	Panel	Description
Resources-> WebSphere JMS Provider	WebSphere JMS providers	List properties and resources at the selected scope
Resources-> WebSphere JMS Provider-> (In content pane, under Additional Properties) WebSphere Queue Connection Factories	WebSphere queue connection factories	List all queue connection factories at the selected scope
Resources-> WebSphere JMS Provider-> (In content pane, under Additional Properties) WebSphere Topic Connection Factories	WebSphere topic connection factories	List all topic connection factories at the selected scope
Resources-> WebSphere JMS Provider-> (In content pane, under Additional Properties) WebSphere Queue Destinations	WebSphere queue destinations	List all queue destinations at the selected scope
Resources-> WebSphere JMS Provider-> (In content pane, under Additional Properties) WebSphere Topic Destinations	WebSphere topic destinations	List all topic destinations at the selected scope
WebSphere MQ JMS provider		
Resources-> WebSphere MQ JMS Provider	WebSphere MQ JMS Provider	List properties and resource types for the WebSphere MQ JMS provider at the selected scope
Resources-> WebSphere MQ JMS Provider	WebSphere MQ JMS provider	List properties and resources of the selected WebSphere MQ JMS provider at the selected scope
Resources-> WebSphere MQ JMS Provider-> (In content pane, under Additional Properties) WebSphere MQ Queue Connection Factories	WebSphere MQ queue connection factories	List all queue connection factories of the selected WebSphere MQ JMS provider at the selected scope
Resources-> WebSphere MQ JMS Provider-> (In content pane, under Additional Properties) WebSphere MQ Topic Connection Factories	WebSphere MQ topic connection factories	List all topic connection factories of the selected WebSphere MQ JMS provider at the selected scope
Resources-> WebSphere MQ JMS Provider-> (In content pane, under Additional Properties) WebSphere MQ Queue Destinations	WebSphere MQ queue destinations	List all queue destinations of the selected WebSphere MQ JMS provider at the selected scope
Resources-> WebSphere MQ JMS Provider-> (In content pane, under Additional Properties) WebSphere MQ Topic Destinations	WebSphere MQ topic destinations	List all topic destinations of the selected WebSphere MQ JMS provider at the selected scope

Table 4. Network Deployment - Administrative panels for JMS resources (continued)

Path	Panel	Description
Generic JMS providers A JMS provider other than the embedded WebSphere JMS provider or the WebSphere MQ JMS provider		
Resources-> Generic JMS Providers	Generic JMS providers	List all generic JMS providers within the administration domain
Resources-> Generic JMS Providers-> (In content pane) *provider_name*	Generic JMS provider *provider_name*	List properties and resources of the selected generic JMS provider *provider_name* at the selected scope
Resources-> Generic JMS Providers-> (In content pane) *provider_name*->(Under Additional Properties) JMS Connection Factories	Generic JMS connection factories	List all JMS connection factories of the selected generic JMS provider *provider_name* at the selected scope
Resources-> Generic JMS Providers-> (In content pane) *provider_name*-> (Under Additional Properties) JMS Destinations	Generic JMS destinations	List all JMS destinations (queues and topics) of the selected generic JMS provider *provider_name*

JMS server collection:

Each JMS server provides the functions of the JMS provider for a node in your administrative domain. Use this panel to list the JMS servers within the administration domain, or to select a JMS server to view or change its configuration properties.

There can be at most one JMS server on each node in the administration domain, and any application server within the domain can access JMS resources served by any JMS server on any node in the domain.

To view this administrative console page, use the administrative console to complete the following steps:
1. In the navigation pane, select **Servers-> JMS Servers**.

To view or change the properties of a JMS server, select its name in the list displayed.

To act on one or more of the JMS servers listed, click the check boxes next to the names of the objects that you want to act on, then use the buttons provided.

JMS provider collection:

Use this panel to list JMS providers, or to select a JMS provider to view or change its configuration properties.

To view this administrative console page, use the administrative console to complete the following steps:
1. In the navigation pane, expand one of the following paths:
   - **Resources-> WebSphere JMS Provider**

- **Resources-> WebSphere MQ JMS Provider**
- **Resources-> Generic JMS Providers**

To view or change the properties of a JMS provider or its resources, select its name in the list displayed.

To define a new generic JMS provider, on the **Resources-> Generic JMS Providers** page click **New**.

To act on one or more of the JMS providers listed, click the check boxes next to the names of the objects that you want to act on, then use the buttons provided.

Name The name by which this JMS provider is known for administrative purposes.

Description
A description of this JMS provider for administrative purposes.

WebSphere JMS provider settings:

Use this panel to view the configuration properties of the embedded WebSphere JMS provider that is installed with WebSphere Application Server. *You cannot change these properties.*

To view this administrative console page, click **Resources-> WebSphere JMS Provider**

Name:

The name by which the JMS provider is known for administrative purposes.

Data type	String
Default	WebSphereJMSProvider

Description:

A description of the JMS provider, for administrative purposes within IBM WebSphere Application Server.

Data type	String
Default	Built-in WebSphere JMS Provider

WebSphere MQ JMS provider settings:

Use this panel to view the configuration properties of the WebSphere MQ JMS provider. These properties apply only if you have installed WebSphere MQ as the JMS provider over the internal JMS provider installed with WebSphere Application Server. *You cannot change these properties.*

To view this administrative console page, click **Resources-> WebSphere MQ JMS Provider**

Name:

The name by which the WebSphere MQ JMS provider is known for administrative purposes.

Data type String
Default WebSphereMQJMSProvider

Description:

A description of the JMS provider, for administrative purposes within IBM WebSphere Application Server.

Data type String
Default WebSphere MQ JMS provider

Classpath:

The Java classpath for the JMS provider. The list of paths or JAR file names that together form the location for the JMS provider classes.

Data type String
Default $MQJMS_LIB_ROOT

Native Library Path:

The native library path for the JMS provider. An optional path to any native libraries needed by the JMS provider.

Data type String
Default $MQJMS_LIB_ROOT

The Native Library Path property is set to the directory where the WebSphere MQ Java feature is installed.

JMS provider settings:

If you want to use a JMS provider other than the embedded WebSphere JMS provider or the WebSphere MQ JMS provider, use this panel to configure properties of the JMS provider.

To view this administrative console page, click **Resources-> Generic JMS Providers->** ***provider_name***

Name:

The name by which the JMS provider is known for administrative purposes.

Data type String
Default Null

Description:

A description of the JMS provider, for administrative purposes

Data type String
Default Null

Classpath:

The Java classpath for the JMS provider. The list of paths or JAR file names that together form the location for the JMS provider classes.

Data type	String
Default	Null

Native Library Path:

The native library path for the JMS provider. An optional path to any native libraries needed by the JMS provider.

Data type	String
Default	Null

External initial context factory:

The Java classname of the initial context factory for the JMS provider.

For example, for an LDAP service provider the value has the form: `com.sun.jndi.ldap.LdapCtxFactory`.

Data type	String
Default	Null

External provider URL:

The JMS provider URL for external JNDI lookups.

For example, an LDAP URL for a JMS provider has the form: `ldap://hostname.company.com/contextName`.

Data type	String
Default	Null

WebSphere Queue connection factory collection:

The queue connection factories configured in the embedded WebSphere JMS provider for point-to-point messaging with JMS queues.

This panel shows a list of the WebSphere queue connection factories with a summary of their configuration properties.

To view this administrative console page, use the administrative console to complete the following steps:
1. In the navigation pane, expand **Resources-> WebSphere JMS Provider**.
2. Change the **Scope** check box to Cell, Node (for a JMS provider on a specific node), or Server, according to your needs.
3. In the Additional Properties list in the contents pane, select **WebSphere Queue Connection Factory**.

To view or change the properties of a connection factory, select its name in the list displayed.

To act on one or more of the connection factories listed, click the check boxes next to the names of the objects that you want to act on, then use the buttons provided.

WebSphere queue connection factory settings:

Use this panel to view or change the configuration properties of the selected queue connection factory for use with the embedded WebSphere JMS provider that is installed with WebSphere Application Server. These configuration properties control how connections are created to the associated JMS queue destination.

A queue connection factory is used to create JMS connections to queue destinations. The queue connection factory is created by the embedded WebSphere JMS provider. A queue connection factory for the embedded WebSphere JMS provider has the following properties:

To view this administrative console page, click **Resources-> WebSphere JMS Provider-> (In content pane, under Additional Properties) WebSphere Queue Connection Factories->** *connection_factory*

Scope:

Specifies the level to which this resource definition is visible -- the cell, node, or server level.

Resources such as JMS Providers, Namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

Note that no matter what the scope of a defined resource, the resource's properties only apply at an individual server level. For example, if you define the scope of a data source at the Cell level, all users in that Cell can look up and use that data source, which is unique within that Cell. However, resource property settings are local to each server in the Cell. For example, if you set **Max Connections** to 10, then each server in that Cell can have 10 connections.

Cell The most general scope. Resources defined at the Cell scope are visible from all Nodes and servers, unless they are overridden. To view resources defined in the cell scope, do not specify a server or a node name in the scope selection form.

Node The default scope for most resource types. Resources defined at the Node scope override any duplicates defined at the Cell scope and are visible to all servers on the same node, unless they are overridden at a server scope on that node. To view resources defined in a node scope, do not specify a server, but select a node name in the scope selection form.

Server The most specific scope for defining resources. Resources defined at the Server scope override any duplicate resource definitions defined at the Cell scope or parent Node scope and are visible only to a specific server. To view resources defined in a server scope, specify a server name as well as a node name in the scope selection form.

When resources are created, they are always created into the current scope selected in the panel. To view resources in other scopes, specify a different node or server in the scope selection form.

Data type String

Name:

The name by which this queue connection factory is known for administrative purposes within IBM WebSphere Application Server. The name must be unique within the JMS connection factories across the WebSphere administrative domain.

Data type String
Default Null

JNDI name:

The JNDI name that is used to bind the connection factory into the application server's name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Description:

A description of this connection factory for administrative purposes within IBM WebSphere Application Server.

Data type String
Default Null

Category:

A category used to classify or group this connection factory, for your IBM WebSphere Application Server administrative records.

Data type String

Component-managed Authentication Alias:

This alias specifies a user ID and password to be used to authenticate connection to a JMS provider for application-managed authentication.

This property provides a list of the J2C authentication data entry aliases that have been defined to WebSphere Application Server. You can select a data entry alias to be used to authenticate the creation of a new connection to the JMS provider.

If you have enabled global security for WebSphere Application Server, select the alias that specifies the user ID and password used to authenticate the creation of a new connection to the JMS provider. The use of this alias depends on the resource authentication (res-auth) setting declared in the connection factory resource reference of an application component's deployment descriptors.

Note: User IDs longer than 12 characters cannot be used for authentication with the embedded WebSphere JMS provider. For example, the default Windows NT user ID, **Administrator**, is not valid for use with embedded WebSphere

messaging, because it contains 13 characters. Therefore, an authentication alias for a WebSphere JMS provider connection factory must specify a user ID no longer than 12 characters.

Data type Pick-list

Container-managed Authentication Alias:

This alias specifies a user ID and password to be used to authenticate connection to a JMS provider for container-managed authentication.

This property provides a list of the J2C authentication data entry aliases that have been defined to WebSphere Application Server. You can select a data entry alias to be used to authenticate the creation of a new connection to the JMS provider.

If you have enabled global security for WebSphere Application Server, select the alias that specifies the user ID and password used to authenticate the creation of a new connection to the JMS provider. The use of this alias depends on the resource authentication (res-auth) setting declared in the connection factory resource reference of an application component's deployment descriptors.

Note: User IDs longer than 12 characters cannot be used for authentication with the embedded WebSphere JMS provider. For example, the default Windows NT user ID, **Administrator**, is not valid for use with embedded WebSphere messaging, because it contains 13 characters. Therefore, an authentication alias for a WebSphere JMS provider connection factory must specify a user ID no longer than 12 characters.

Data type Pick-list

Mapping-Configuration Alias:

Allows users to select from the **Security** > **JAAS Configuration** > **Application Logins Configuration** list.

The **DefaultPrincipalMapping** JAAS configuration maps the authentication alias to the userid and password. You may define and use other mapping configurations. For more information about the mapping configurations, see the Information Center topic ″Java Authentication and Authorization service configuration entry settings.″

Data type Pick-list

Node:

The WebSphere node name of the administrative node where the JMS server runs for this connection factory. Connections created by this factory connect to that JMS server.

Data type String
Units Enum
Default Null
Range Pull-down list of nodes in the WebSphere administrative domain.

XA Enabled:

Specifies whether the connection factory is for XA or non-XA coordination of messages and controls if the application server uses XA QCF/TCF. Enable XA if multiple resources are not used in the same transaction.

If you clear this checkbox property (for non-XA coordination), the JMS session is still enlisted in a transaction, but uses the resource manager local transaction calls (session.commit and session.rollback) instead of XA calls. This can lead to an improvement in performance. However, this means that only a single resource can be enlisted in a transaction in WebSphere Application Server.

5.1 + In WBI Server Foundation the last participant support enables you to enlist one non-XA resource with other XA-capable resources.

For a WebSphere Topic Connection Factory with the **Port** property set to `DIRECT` this property does not apply, and always adopts non-XA coordination.

Data type	Checkbox
Default	Selected (enabled for XA coordination)
Range	**Selected** The connection factory is enabled for XA-coordination of messages **Cleared** The connection factory is not enabled for XA coordination of messages
Recommended	Do not enable XA coordination when the message queue or topic received is the only resource in the transaction. Enable XA coordination when other resources, including other queues or topics, are involved.

Connection pool:

Specifies an optional set of connection pool settings.

Connection pool properties are common to all J2C connectors.

The application server pools connections and sessions with the JMS provider to improve performance. You need to configure the connection and session pool properties appropriately for your applications, otherwise you may not get the connection and session behavior that you want.

Change the size of the connection pool if concurrent server-side access to the JMS resource exceeds the default value.The size of the connection pool is set on a per queue or topic basis. See the following table for details.

Session pool:

An optional set of session pool settings.

This link provides a panel of optional connection pool properties, common to all J2C connectors.

The application server pools connections and sessions with the JMS provider to improve performance. You need to configure the connection and session pool properties appropriately for your applications, otherwise you may not get the connection and session behavior that you want.

Session pool settings:

Use this page to configure session pool settings.

This administrative console page is common to a range of resource types; for example, JMS queue connection factories. To view this page, the path depends on the type of resource, but generally you select an instance of the resource provider, then an instance of the resource type, then click **Session Pool**. For example: click **Resources** > **WebSphere JMS Provider** > **WebSphere Queue Connection Factories** > *connection_factory* > **Session Pool**.

Scope:

Specifies the level to which this resource definition is visible -- the cell, node, or server level.

Resources such as JDBC Providers, Namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

Note that no matter what the scope of a defined resource, the resource's properties only apply at an individual server level. For example, if you define the scope of a data source at the Cell level, all users in that Cell can look up and use that data source, which is unique within that Cell. However, resource property settings are local to each server in the Cell. For example, if you set **Max Connections** to 10, then each server in that Cell can have 10 connections.

Cell The most general scope. Resources defined at the Cell scope are visible from all Nodes and servers, unless they are overridden. To view resources defined in the cell scope, do not specify a server or a node name in the scope selection form.

Node The default scope for most resource types. Resources defined at the Node scope override any duplicates defined at the Cell scope and are visible to all servers on the same node, unless they are overridden at a server scope on that node. To view resources defined in a node scope, do not specify a server, but select a node name in the scope selection form.

Server The most specific scope for defining resources. Resources defined at the Server scope override any duplicate resource definitions defined at the Cell scope or parent Node scope and are visible only to a specific server. To view resources defined in a server scope, specify a server name as well as a node name in the scope selection form.

When resources are created, they are always created into the current scope selected in the panel. To view resources in other scopes, specify a different node or server in the scope selection form.

Data type String

Connection Timeout:

Specifies the interval, in seconds, after which a connection request times out and a ConnectionWaitTimeoutException is thrown.

The wait is necessary when the maximum value of connections (**Max Connections**) to a particular connection pool is reached . For example, if *Connection Timeout* is set to 300 and the maximum number of connections is reached, the Pool Manager waits for 300 seconds for an available physical connection. If a physical connection is *not* available within this time, the Pool Manager throws a ConnectionWaitTimeoutException. It usually does not make sense to retry the getConnection() method, because if a longer wait time is required, you should set the **Connection Timeout** setting to a higher value. Therefore, if this exception is caught by the application, the administrator should review the expected usage of the application and tune the connection pool and the database accordingly.

If Connection Timeout is set to 0, the Pool Manager waits as long as necessary until a connection is allocated (which happens when the number of connections falls below the value of **Max Connections**).

If Max Connections is set to 0, which enables an infinite number of physical connections, then the Connection Timeout value is ignored.

Data type	Integer
Units	Seconds
Default	180
Range	0 to max int

Max Connections:

Specifies the maximum number of physical connections that you can create in this pool.

These are the physical connections to the backend resource. Once this number is reached, no new physical connections are created and the requester waits until a physical connection that is currently in use returns to the pool, or a ConnectionWaitTimeoutException is thrown.

For example, if the Max Connections value is set to 5, and there are five physical connections in use, the pool manager waits for the amount of time specified in Connection Timeout for a physical connection to become free.

If Max Connections is set to 0, the Connection Timeout value is ignored.

For better performance, set the value for the connection pool lower than the value for the MaxKeepAliveConnections option in the Web container. Lower settings, such as 10-30 connections, perform better than higher settings, such as 100.

If clones are used, one data pool exists for each clone. Knowing the number of data pools is important when configuring the database maximum connections.

Min Connections:

Specifies the minimum number of physical connections to maintain.

Until this number is reached, the pool maintenance thread does not discard physical connections. However, no attempt is made to bring the number of connections up to this number. If you set a value for Aged Timeout, the minimum is not maintained. All connections with an expired age are discarded.

For example if the **Min Connections** value is set to 3, and one physical connection is created, the Unused Timeout thread does not discard that connection. By the same token, the thread does not automatically create two additional physical connections to reach the **Min Connections** setting.

Data type	Integer
Default	1
Range	0 to max int

Reap Time:

Specifies the interval, in seconds, between runs of the pool maintenance thread.

For example, if **Reap Time** is set to 60, the pool maintenance thread runs every 60 seconds. The Reap Time interval affects the accuracy of the **Unused Timeout** and **Aged Timeout** settings. The smaller the interval, the greater the accuracy. If the pool maintenance thread is enabled, set the Reap Time value less than the values of Unused Timeout and Aged Timeout. When the pool maintenance thread runs, it discards any connections remaining unused for longer than the time value specified in Unused Timeout, until it reaches the number of connections specified in **Min Connections**. The pool maintenance thread also discards any connections that remain active longer than the time value specified in Aged Timeout.

The Reap Time interval also affects performance. Smaller intervals mean that the pool maintenance thread runs more often and degrades performance.

To disable the pool maintenance thread set Reap Time to 0, or set both Unused Timeout and Aged Timeout to 0. The recommended way to disable the pool maintenance thread is to set Reap Time to 0, in which case Unused Timeout and Aged Timeout are ignored. However, if Unused Timeout and Aged Timeout are set to 0, the pool maintenance thread runs, but only physical connections which timeout due to non-zero timeout values are discarded.

Data type	Integer
Units	Seconds
Default	180
Range	0 to max int

Unused Timeout:

Specifies the interval in seconds after which an unused or idle connection is discarded.

Set the Unused Timeout value higher than the Reap Timeout value for optimal performance. Unused physical connections are only discarded if the current number of connections not in use exceeds the **Min Connections** setting. For example, if the unused timeout value is set to 120, and the pool maintenance thread is enabled (Reap Time is not 0), any physical connection that remains unused for two minutes is discarded. Note that accuracy of this timeout, as well as performance, is affected by the **Reap Time** value. For more information, see the topic "Reap Time."

Data type	Integer
Units	Seconds
Default	1800

Range	0 to max int

Aged Timeout:

Specifies the interval in seconds before a physical connection is discarded.

Setting **Aged Timeout** to 0 supports active physical connections remaining in the pool indefinitely. Set the Aged Timeout value higher than the **Reap Timeout** value for optimal performance. For example, if the Aged Timeout value is set to 1200, and the Reap Time value is not 0, any physical connection that remains in existence for 1200 seconds (20 minutes) is discarded from the pool. Note that accuracy of this timeout, as well as performance, are affected by the Reap Time value. For more information, see Reap Time.

Data type	Integer
Units	Seconds
Default	0
Range	0 to max int

Purge Policy:

Specifies how to purge connections when a *stale connection* or *fatal connection error* is detected.

Valid values are **EntirePool** and **FailingConnectionOnly**. JCA data sources can have either option. WebSphere Version 4.0 data sources always have a purge policy of **EntirePool**.

Data type	String
Default	FailingConnectionOnly

Range

EntirePool
All connections in the pool are marked stale. Any connection not in use is immediately closed. A connection in use is closed and throws a *StaleConnectionException* during the next operation on that connection. Subsequent *getConnection* requests from the application result in new connections to the database opening. When using this purge policy, there is a slight possibility that some connections in the pool are closed unnecessarily when they are not stale. However, this is a rare occurrence. In most cases, a purge policy of EntirePool is the best choice.

FailingConnectionOnly
Only the connection that caused the *StaleConnectionException* is closed. Although this setting eliminates the possibility that valid connections are closed unnecessarily, it makes recovery from an application perspective more complicated. Because only the currently failing connection is closed, there is a good possibility that the next *getConnection* request from the application can return a connection from the pool that is also stale, resulting in more stale connection exceptions.

WebSphere topic connection factory collection:

The topic connection factories configured in the embedded WebSphere JMS provider for publish/subscribe messaging with JMS topics.

This panel shows a list of the WebSphere topic connection factories with a summary of their configuration properties.

To view this administrative console page, use the administrative console to complete the following steps:
1. In the navigation pane, expand **Resources-> WebSphere JMS Provider**.
2. Change the **Scope** check box to Cell, Node (for a JMS provider on a specific node), or Server, according to your needs.
3. In the Additional Properties list in the contents pane, select **WebSphere Topic Connection Factory**.

To view or change the properties of a connection factory, select its name in the list displayed.

To act on one or more of the connection factories listed, click the check boxes next to the names of the objects that you want to act on, then use the buttons provided.

WebSphere topic connection factory settings:

Use this panel to view or change the configuration properties of the selected topic connection factory for use with the embedded WebSphere JMS provider. These configuration properties control how connections are created to the associated JMS topic destination.

A topic connection factory is used to create JMS connections to topic destinations. The topic connection factory is created by the associated JMS provider. A topic connection factory for the embedded WebSphere JMS provider has the following properties.

To view this administrative console page, click **Resources-> WebSphere JMS Provider-> (In content pane, under Additional Properties) WebSphere Topic Connection Factories->** ***connection_factory***

Scope:

Specifies the level to which this resource definition is visible -- the cell, node, or server level.

Resources such as JMS Providers, Namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

Note that no matter what the scope of a defined resource, the resource's properties only apply at an individual server level. For example, if you define the scope of a data source at the Cell level, all users in that Cell can look up and use that data source, which is unique within that Cell. However, resource property settings are local to each server in the Cell. For example, if you set **Max Connections** to 10, then each server in that Cell can have 10 connections.

Cell The most general scope. Resources defined at the Cell scope are visible from all Nodes and servers, unless they are overridden. To view resources defined in the cell scope, do not specify a server or a node name in the scope selection form.

Node The default scope for most resource types. Resources defined at the Node scope override any duplicates defined at the Cell scope and are visible to all servers on the same node, unless they are overridden at a server scope on that node. To view resources defined in a node scope, do not specify a server, but select a node name in the scope selection form.

Server The most specific scope for defining resources. Resources defined at the Server scope override any duplicate resource definitions defined at the Cell scope or parent Node scope and are visible only to a specific server. To view resources defined in a server scope, specify a server name as well as a node name in the scope selection form.

When resources are created, they are always created into the current scope selected in the panel. To view resources in other scopes, specify a different node or server in the scope selection form.

Data type String

Name:

The name by which this queue connection factory is known for administrative purposes within IBM WebSphere Application Server. The name must be unique within the JMS connection factories across the WebSphere administrative domain.

Data type	String
Default	Null

JNDI name:

The JNDI name that is used to bind the topic connection factory into the application server's name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type	String

Description:

A description of this topic connection factory for administrative purposes within IBM WebSphere Application Server.

Data type	String
Default	Null

Category:

A category used to classify or group this topic connection factory, for your IBM WebSphere Application Server administrative records.

Data type	String

Node:

The WebSphere node name of the administrative node where the JMS server runs for this connection factory. Connections created by this factory connect to that JMS server.

Data type	Enum
Default	Null
Range	Pull-down list of nodes in the WebSphere administrative domain.

Port:

Which of the two ports that connections use to connect to the JMS Server. The QUEUED port is for full-function JMS publish/subscribe support, the DIRECT port is for non-persistent, non-transactional, non-durable subscriptions only.

Note: Message-driven beans cannot use the direct listener port for publish/subscribe support. Therefore, any topic connection factory configured with **Port** set to `Direct` cannot be used with message-driven beans.

Data type	Enum
Units	Not applicable
Default	QUEUED
Range	**QUEUED** The listener port used for full-function JMS-compliant, publish/subscribe support. **DIRECT** The listener port used for direct TCP/IP connection (non-transactional, non-persistent, and non-durable subscriptions only) for publish/subscribe support. The TCP/IP port numbers for these ports are defined on the WebSphere Internal JMS Server.

Component-managed Authentication Alias:

This alias specifies a user ID and password to be used to authenticate connection to a JMS provider for application-managed authentication.

This property provides a list of the J2C authentication data entry aliases that have been defined to WebSphere Application Server. You can select a data entry alias to be used to authenticate the creation of a new connection to the JMS provider.

If you have enabled global security for WebSphere Application Server, select the alias that specifies the user ID and password used to authenticate the creation of a new connection to the JMS provider. The use of this alias depends on the resource authentication (res-auth) setting declared in the connection factory resource reference of an application component's deployment descriptors.

Note: User IDs longer than 12 characters cannot be used for authentication with the embedded WebSphere JMS provider. For example, the default Windows NT user ID, **Administrator**, is not valid for use with embedded WebSphere messaging, because it contains 13 characters. Therefore, an authentication alias for a WebSphere JMS provider connection factory must specify a user ID no longer than 12 characters.

Container-managed Authentication Alias:

This alias specifies a user ID and password to be used to authenticate connection to a JMS provider for container-managed authentication.

This property provides a list of the J2C authentication data entry aliases that have been defined to WebSphere Application Server. You can select a data entry alias to be used to authenticate the creation of a new connection to the JMS provider.

If you have enabled global security for WebSphere Application Server, select the alias that specifies the user ID and password used to authenticate the creation of a new connection to the JMS provider. The use of this alias depends on the resource

authentication (res-auth) setting declared in the connection factory resource reference of an application component's deployment descriptors.

Note: User IDs longer than 12 characters cannot be used for authentication with the embedded WebSphere JMS provider. For example, the default Windows NT user ID, **Administrator**, is not valid for use with embedded WebSphere messaging, because it contains 13 characters. Therefore, an authentication alias for a WebSphere JMS provider connection factory must specify a user ID no longer than 12 characters.

Mapping-Configuration Alias:

Allows users to select from the **Security** > **JAAS Configuration** > **Application Logins Configuration** list.

The **DefaultPrincipalMapping** JAAS configuration maps the authentication alias to the userid and password. You may define and use other mapping configurations. For more information about the mapping configurations, see the Information Center topic ″Java Authentication and Authorization service configuration entry settings.″

Data type Pick-list

Clone Support:

Select this checkbox to enable clone support to allow the same durable subscription across topic clones.

Data type Enum
Default Cleared
Range
Selected
Clone support is enabled.
Cleared
Clone support is disabled.

If you select this property, you must also specify a value for the **Client ID** property.

Client ID:

The JMS client identifier used for connections to the queue manager.

Data type String
Range A valid JMS client ID

XA Enabled:

Specifies whether the connection factory is for XA or non-XA coordination of messages and controls if the application server uses XA QCF/TCF. Enable XA if multiple resources are not used in the same transaction.

If you clear this checkbox property (for non-XA coordination), the JMS session is still enlisted in a transaction, but uses the resource manager local transaction calls (session.commit and session.rollback) instead of XA calls. This can lead to an

improvement in performance. However, this means that only a single resource can be enlisted in a transaction in WebSphere Application Server.

5.1 + In WBI Server Foundation the last participant support enables you to enlist one non-XA resource with other XA-capable resources.

For a WebSphere Topic Connection Factory with the **Port** property set to `DIRECT` this property does not apply, and always adopts non-XA coordination.

Data type	Checkbox
Default	Selected (enabled for XA coordination)
Range	**Selected** The connection factory is enabled for XA-coordination of messages **Cleared** The connection factory is not enabled for XA coordination of messages
Recommended	Do not enable XA coordination when the message queue or topic received is the only resource in the transaction. Enable XA coordination when other resources, including other queues or topics, are involved.

Connection pool:

Specifies an optional set of connection pool settings.

Connection pool properties are common to all J2C connectors.

The application server pools connections and sessions with the JMS provider to improve performance. You need to configure the connection and session pool properties appropriately for your applications, otherwise you may not get the connection and session behavior that you want.

Change the size of the connection pool if concurrent server-side access to the JMS resource exceeds the default value.The size of the connection pool is set on a per queue or topic basis. See the following table for details.

Session pool:

An optional set of session pool settings.

This link provides a panel of optional connection pool properties, common to all J2C connectors.

The application server pools connections and sessions with the JMS provider to improve performance. You need to configure the connection and session pool properties appropriately for your applications, otherwise you may not get the connection and session behavior that you want.

WebSphere Queue destination collection:

The queue destinations configured in the embedded WebSphere JMS provider for point-to-point messaging with JMS queues.

This panel shows a list of the WebSphere queue destinations with a summary of their configuration properties.

To view this administrative console page, use the administrative console to complete the following steps:
1. In the navigation pane, expand **Resources-> WebSphere JMS Provider**.
2. Change the **Scope** check box to Cell, Node (for a JMS provider on a specific node), or Server, according to your needs.
3. In the Additional Properties list in the contents pane, select **WebSphere Queue Destination**.

To view or change the properties of a queue destination, select its name in the list displayed.

To act on one or more of the queue destinations listed, click the check boxes next to the names of the objects that you want to act on, then use the buttons provided.

WebSphere queue settings:

Use this panel to view or change the configuration properties of the selected queue destination for use with the WebSphere JMS provider.

A queue destination is used to configure the properties of a JMS queue. Connections to the queue are created by the associated queue connection factory for the embedded WebSphere JMS provider. A queue for use with the internal WebSphere JMS provider has the following properties.

To view this administrative console page, click **Resources-> WebSphere JMS Provider-> (In content pane, under Additional Properties) WebSphere Queue Destinations->** ***destination_name***

Scope:

Specifies the level to which this resource definition is visible -- the cell, node, or server level.

Resources such as JMS Providers, Namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

Note that no matter what the scope of a defined resource, the resource's properties only apply at an individual server level. For example, if you define the scope of a data source at the Cell level, all users in that Cell can look up and use that data source, which is unique within that Cell. However, resource property settings are local to each server in the Cell. For example, if you set **Max Connections** to 10, then each server in that Cell can have 10 connections.

Cell The most general scope. Resources defined at the Cell scope are visible from all Nodes and servers, unless they are overridden. To view resources defined in the cell scope, do not specify a server or a node name in the scope selection form.

Node The default scope for most resource types. Resources defined at the Node scope override any duplicates defined at the Cell scope and are visible to all servers on the same node, unless they are overridden at a server scope on that node. To view resources defined in a node scope, do not specify a server, but select a node name in the scope selection form.

Server The most specific scope for defining resources. Resources defined at the

Server scope override any duplicate resource definitions defined at the Cell scope or parent Node scope and are visible only to a specific server. To view resources defined in a server scope, specify a server name as well as a node name in the scope selection form.

When resources are created, they are always created into the current scope selected in the panel. To view resources in other scopes, specify a different node or server in the scope selection form.

Data type String

Name:

The name by which the queue is known for administrative purposes within IBM WebSphere Application Server.

To enable applications to use this queue, you must add the queue name to the list of queue names in the configuration of the JMS servers that host the queue.

Data type String

JNDI name:

The JNDI name that is used to bind the queue into the application server's name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Description:

A description of the queue, for administrative purposes

Data type String
Default Null

Category:

A category used to classify or group this queue, for your IBM WebSphere Application Server administrative records.

Data type String

Persistence:

Whether all messages sent to the destination are persistent, non-persistent, or have their persistence defined by the application

Data type Enum
Default APPLICATION DEFINED
Range
APPLICATION DEFINED
Messages on the destination have their persistence defined by the application that put them onto the queue.
NON-PERSISTENT
Messages on the destination are not persistent.
PERSISTENT
Messages on the destination are persistent. When a persistent message is put to a queue, all of the message data is written to the messaging log (under the *embedded_messaging_install*\log directory) to make recovery of the message possible.

Priority:

Whether the message priority for this destination is defined by the application or the **Specified priority** property

Data type Enum
Units Not applicable
Default APPLICATION DEFINED
Range
APPLICATION DEFINED
The priority of messages on this destination is defined by the application that put them onto the destination.
QUEUE DEFINED
[WebSphere MQ destination only] Messages on the destination have their persistence defined by the WebSphere MQ queue definition properties.
SPECIFIED
The priority of messages on this destination is defined by the **Specified priority** property.*If you select this option, you must define a priority on the **Specified priority** property.*

Specified priority:

If the **Priority** property is set to `Specified`, type here the message priority for this queue, in the range 0 (lowest) through 9 (highest)

If the **Priority** property is set to `Specified`, messages sent to this queue have the priority value specified by this property.

Data type Integer
Units Message priority level

Default 0
Range 0 (lowest priority) through 9 (highest priority)

Expiry:

Whether the expiry timeout for this queue is defined by the application or the **Specified expiry** property, or messages on the queue never expire (have an unlimited expiry timeout)

Data type Enum
Units Not applicable
Default APPLICATION DEFINED
Range

APPLICATION DEFINED
The expiry timeout for messages on this queue is defined by the application that put them onto the queue.

UNLIMITED
Messages on this queue have no expiry timeout, so those messages never expire.

SPECIFIED
The expiry timeout for messages on this queue is defined by the **Specified expiry** property.*If you select this option, you must define a timeout on the **Specified expiry** property.*

Specified expiry:

If the **Expiry timeout** property is set to `Specified`, type here the number of milliseconds (greater than 0) after which messages on this queue expire

Data type Integer
Units Milliseconds
Default 0
Range Greater than or equal to 0
- 0 indicates that messages never timeout
- Other values are an integer number of milliseconds

WebSphere topic destination collection:

The topic destinations configured in the embedded WebSphere JMS provider for publish/subscribe messaging with JMS topics. Use this panel to create or delete topic destinations, or to select a topic destination to view or change its configuration properties.

This panel shows a list of the WebSphere topic destinations with a summary of their configuration properties.

To view this administrative console page, use the administrative console to complete the following steps:
1. In the navigation pane, expand **Resources-> WebSphere JMS Provider**.

2. Change the **Scope** check box to Cell, Node (for a JMS provider on a specific node), or Server, according to your needs.
3. In the Additional Properties list in the contents pane, select **WebSphere Topic Destination**.

To view or change the properties of a topic destination, select its name in the list displayed.

To act on one or more of the topic destinations listed, click the check boxes next to the names of the objects that you want to act on, then use the buttons provided.

WebSphere topic settings:

Use this panel to view or change the configuration properties of the selected topic destination for use with the embedded WebSphere JMS provider.

A topic destination is used to configure the properties of a JMS topic for the associated JMS provider. Connections to the topic are created by the associated topic connection factory. A topic for use with the embedded WebSphere JMS provider has the following properties.

To view this administrative console page, click **Resources-> WebSphere JMS Provider-> (In content pane, under Additional Properties) WebSphere Topic destinations->** ***destination_name***

Scope:

Specifies the level to which this resource definition is visible -- the cell, node, or server level.

Resources such as JMS Providers, Namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

Note that no matter what the scope of a defined resource, the resource's properties only apply at an individual server level. For example, if you define the scope of a data source at the Cell level, all users in that Cell can look up and use that data source, which is unique within that Cell. However, resource property settings are local to each server in the Cell. For example, if you set **Max Connections** to 10, then each server in that Cell can have 10 connections.

Cell The most general scope. Resources defined at the Cell scope are visible from all Nodes and servers, unless they are overridden. To view resources defined in the cell scope, do not specify a server or a node name in the scope selection form.

Node The default scope for most resource types. Resources defined at the Node scope override any duplicates defined at the Cell scope and are visible to all servers on the same node, unless they are overridden at a server scope on that node. To view resources defined in a node scope, do not specify a server, but select a node name in the scope selection form.

Server The most specific scope for defining resources. Resources defined at the Server scope override any duplicate resource definitions defined at the Cell scope or parent Node scope and are visible only to a specific server. To view resources defined in a server scope, specify a server name as well as a node name in the scope selection form.

When resources are created, they are always created into the current scope selected in the panel. To view resources in other scopes, specify a different node or server in the scope selection form.

Data type String

Name:

The name by which the topic is known for administrative purposes within IBM WebSphere Application Server.

Data type String

JNDI name:

The JNDI name that is used to bind the topic into the application server's name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Description:

A description of the topic, for administrative purposes within IBM WebSphere Application Server.

Data type String
Default Null

Category:

A category used to classify or group this topic, for your IBM WebSphere Application Server administrative records.

Data type String

Topic:

The name of the topic as defined to the JMS provider.

Data type String
Default Null
Range The topic value can be dot notation and include wildcard characters.

Persistence:

Whether all messages sent to the destination are persistent, non-persistent, or have their persistence defined by the application

Data type Enum
Default APPLICATION DEFINED
Range
APPLICATION DEFINED
Messages on the destination have their persistence defined by the application that put them onto the queue.
NON-PERSISTENT
Messages on the destination are not persistent.
PERSISTENT
Messages on the destination are persistent.
QUEUE DEFINED
[WebSphere MQ destination only] Messages on the destination have their persistence defined by the WebSphere MQ queue definition properties.

Priority:

Whether the message priority for this destination is defined by the application or the **Specified priority** property

Data type Enum
Units Not applicable
Default APPLICATION DEFINED
Range
APPLICATION DEFINED
The priority of messages on this destination is defined by the application that put them onto the destination.
QUEUE DEFINED
[WebSphere MQ destination only] Messages on the destination have their persistence defined by the WebSphere MQ queue definition properties.
SPECIFIED
The priority of messages on this destination is defined by the **Specified priority** property.*If you select this option, you must define a priority on the **Specified priority** property.*

Specified priority:

If the **Priority** property is set to `Specified`, type here the message priority for this queue, in the range 0 (lowest) through 9 (highest)

If the **Priority** property is set to `Specified`, messages sent to this queue have the priority value specified by this property.

Data type	Integer
Units	Message priority level
Default	0
Range	0 (lowest priority) through 9 (highest priority)

Expiry:

Whether the expiry timeout for this queue is defined by the application or the **Specified expiry** property, or messages on the queue never expire (have an unlimited expiry timeout)

Data type	Enum
Units	Not applicable
Default	APPLICATION DEFINED
Range	**APPLICATION DEFINED** The expiry timeout for messages on this queue is defined by the application that put them onto the queue. **UNLIMITED** Messages on this queue have no expiry timeout, so those messages never expire. **SPECIFIED** The expiry timeout for messages on this queue is defined by the **Specified expiry** property.*If you select this option, you must define a timeout on the **Specified expiry** property.*

Specified expiry:

If the **Expiry timeout** property is set to `Specified`, type here the number of milliseconds (greater than 0) after which messages on this queue expire

Data type	Integer
Units	Milliseconds
Default	0
Range	Greater than or equal to 0 • 0 indicates that messages never timeout • Other values are an integer number of milliseconds

WebSphere MQ queue connection factory collection:

The queue connection factories configured in the WebSphere MQ JMS provider for point-to-point messaging with JMS queues.

This panel shows a list of the WebSphere MQ queue connection factories with a summary of their configuration properties.

To view this administrative console page, use the administrative console to complete the following steps:
1. In the navigation pane, expand **Resources-> WebSphere MQ JMS Provider**.

2. Change the **Scope** check box to Cell, Node (for a JMS provider on a specific node), or Server, according to your needs.
3. In the Additional Properties list in the contents pane, select **WebSphere MQ Queue Connection Factory**.

To view or change the properties of a connection factory, select its name in the list displayed.

To act on one or more of the connection factories listed, click the check boxes next to the names of the objects that you want to act on, then use the buttons provided.

WebSphere MQ queue connection factory settings:

Use this panel to view or change the configuration properties of the selected queue connection factory for use with the WebSphere MQ JMS provider. These configuration properties control how connections are created to the associated JMS queue destination.

A queue connection factory is used to create JMS connections to queue destinations. The queue connection factory is created by the WebSphere MQ JMS provider. A queue connection factory for the WebSphere MQ JMS provider has the following properties.

Note:

- The property values that you specify must match the values that you specified when configuring WebSphere MQ for JMS resources. For more information about configuring WebSphere MQ JMS resources, see the WebSphere MQ *Using Java* book. and the *WebSphere MQ System Administration* book, SC33-1873, which are available from the WebSphere MQ messaging platform-specific books Web page at http://www-3.ibm.com/software/ts/mqseries/library/manualsa/manuals/platspecific.html, the IBM Publications Center, or from the WebSphere MQ collection kit, SK2T-0730.
- In WebSphere MQ, names can have a maximum of 48 characters, with the exception of channels which have a maximum of 20 characters.
- **5.1 +** You can use the Custom properties page to define WebSphere MQ Secure Sockets Layer (SSL) properties. For more information about setting those properties, see Custom properties.

To view this administrative console page, click **Resources-> WebSphere MQ JMS Provider-> (In content pane, under Additional Properties) WebSphere MQ Queue Connection Factories->** ***connection_factory***

Scope:

Specifies the level to which this resource definition is visible -- the cell, node, or server level.

Resources such as JMS Providers, Namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

Note that no matter what the scope of a defined resource, the resource's properties only apply at an individual server level. For example, if you define the scope of a data source at the Cell level, all users in that Cell can look up and use that data source, which is unique within that Cell. However, resource property settings are

local to each server in the Cell. For example, if you set **Max Connections** to 10, then each server in that Cell can have 10 connections.

Cell The most general scope. Resources defined at the Cell scope are visible from all Nodes and servers, unless they are overridden. To view resources defined in the cell scope, do not specify a server or a node name in the scope selection form.

Node The default scope for most resource types. Resources defined at the Node scope override any duplicates defined at the Cell scope and are visible to all servers on the same node, unless they are overridden at a server scope on that node. To view resources defined in a node scope, do not specify a server, but select a node name in the scope selection form.

Server The most specific scope for defining resources. Resources defined at the Server scope override any duplicate resource definitions defined at the Cell scope or parent Node scope and are visible only to a specific server. To view resources defined in a server scope, specify a server name as well as a node name in the scope selection form.

When resources are created, they are always created into the current scope selected in the panel. To view resources in other scopes, specify a different node or server in the scope selection form.

Data type String

Name:

The name by which this queue connection factory is known for administrative purposes within IBM WebSphere Application Server. The name must be unique within the JMS connection factories across the WebSphere administrative domain.

Data type String

JNDI name:

The JNDI name that is used to bind the connection factory into the application server's name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Description:

A description of this connection factory for administrative purposes within IBM WebSphere Application Server.

Data type String
Default Null

Category:

A category used to classify or group this connection factory, for your IBM WebSphere Application Server administrative records.

Data type String

Component-managed Authentication Alias:

This alias specifies a user ID and password to be used to authenticate connection to a JMS provider for application-managed authentication.

This property provides a list of the J2C authentication data entry aliases that have been defined to WebSphere Application Server. You can select a data entry alias to be used to authenticate the creation of a new connection to the JMS provider.

If you have enabled global security for WebSphere Application Server, select the alias that specifies the user ID and password used to authenticate the creation of a new connection to the JMS provider. The use of this alias depends on the resource authentication (res-auth) setting declared in the connection factory resource reference of an application component's deployment descriptors.

Restriction:

1. User IDs longer than 12 characters cannot be used for authentication with the embedded WebSphere JMS provider. For example, the default Windows NT user ID, **Administrator**, is not valid for use with embedded WebSphere messaging, because it contains 13 characters. Therefore, an authentication alias for a WebSphere JMS provider connection factory must specify a user ID no longer than 12 characters.
2. If you want to use a WebSphere MQ JMS Provider JMS connection when using Bindings transport mode, you set the property **Transport type**=BINDINGS on the WebSphere MQ Queue Connection Factory. You must also choose one of the following options:
   - To use security credentials, ensure that the user specified is the currently logged on user for the WebSphere Application Server process. If the user specified is not the current logged on user for the WebSphere Application Server process, then the WebSphere MQ JMS Bindings authentication throws the error `MQJMS2013 invalid security authentication supplied for MQQueueManager`.
   - Do not specify security credentials. On the WebSphere MQ Connection Factory, ensure that both the **Component-managed Authentication Alias** and the **Container-managed Authentication Alias** properties are not set.

Container-managed Authentication Alias:

This alias specifies a user ID and password to be used to authenticate connection to a JMS provider for container-managed authentication.

This property provides a list of the J2C authentication data entry aliases that have been defined to WebSphere Application Server. You can select a data entry alias to be used to authenticate the creation of a new connection to the JMS provider.

If you have enabled global security for WebSphere Application Server, select the alias that specifies the user ID and password used to authenticate the creation of a new connection to the JMS provider. The use of this alias depends on the resource authentication (res-auth) setting declared in the connection factory resource reference of an application component's deployment descriptors.

Restriction:

1. User IDs longer than 12 characters cannot be used for authentication with the embedded WebSphere JMS provider. For example, the default Windows NT user ID, **Administrator**, is not valid for use with embedded WebSphere messaging, because it contains 13 characters. Therefore, an authentication alias for a WebSphere JMS provider connection factory must specify a user ID no longer than 12 characters.
2. If you want to use a WebSphere MQ JMS Provider JMS connection when using Bindings transport mode, you set the property **Transport type**=BINDINGS on the WebSphere MQ Queue Connection Factory. You must also choose one of the following options:
   - To use security credentials, ensure that the user specified is the currently logged on user for the WebSphere Application Server process. If the user specified is not the current logged on user for the WebSphere Application Server process, then the WebSphere MQ JMS Bindings authentication throws the error `MQJMS2013 invalid security authentication supplied for MQQueueManager`.
   - Do not specify security credentials. On the WebSphere MQ Connection Factory, ensure that both the **Component-managed Authentication Alias** and the **Container-managed Authentication Alias** properties are not set.

Queue manager:

The name of the WebSphere MQ queue manager for this connection factory. Connections created by this factory connect to that queue manager.

Data type	String
Default	Null
Range	A valid WebSphere MQ queue manager name, as 1 through 48 ASCII characters

Host:

The name of the host on which the WebSphere MQ queue manager runs, for client connection only.

Data type	String
Default	Null
Range	A valid TCP/IP hostname

Port:

The TCP/IP port number used for connection to the WebSphere MQ queue manager, for client connection only.

This port must be configured on the WebSphere MQ queue manager.

Data type	Integer
Default	Null
Range	A valid TCP/IP port number, configured on the WebSphere MQ queue manager.

Channel:

The name of the channel used for connection to the WebSphere MQ queue manager, for client connection only.

Data type	String
Default	Null
Range	1 through 20 ASCII characters

Transport type:

Specifies whether the WebSphere MQ client connection or JNI bindings are used for connection to the WebSphere MQ queue manager. The external JMS provider controls the communication protocols between JMS clients and JMS servers. Tune the transport type when you are using non-ASF non-persistent, non-durable, non-transactional messaging or when you want to satisfy security issues and the client is local to the queue manager node.

Data type	Enum
Units	Not applicable
Default	BINDINGS
Range	**BINDINGS** JNI bindings are used to connect to the queue manager. BINDINGS is a shared memory protocol and can only be used when the queue manager is on the same node as the JMS client and comes at some security risks that should be addressed through the use of EJB roles. **CLIENT** WebSphere MQ client connection is used to connect to the queue manager. CLIENT is a typical TCP-based protocol.
Recommended	BINDINGS is faster by 30% or more, but it lacks security. When you have security concerns, BINDINGS is more desirable than CLIENT.

Model queue definition:

The name of the model queue definition that can be used by the queue manager to create temporary queues if a queue requested does not already exist.

Data type	String
Default	Null
Range	1 through 48 ASCII characters

Client ID:

The JMS client identifier used for connections to the WebSphere MQ queue manager.

Data type	String
Default	Null

CCSID:

The coded character set identifier for use with the WebSphere MQ queue manager.

This coded character set identifier (CCSID) must be one of the CCSIDs supported by WebSphere MQ.

Data type	String
Units	Integer
Default	Null
Range	1 through 65535

For more information about supported CCSIDs, and about converting between message data from one coded character set to another, see the *WebSphere MQ System Administration* and the *WebSphere MQ Application Programming Reference* books. These are available from the WebSphere MQ messaging multiplatform and platform-specific books Web pages; for example, at http://www-3.ibm.com/software/ts/mqseries/library/manualsa/manuals/platspecific.html, the IBM Publications Center, or from the WebSphere MQ collection kit, SK2T-0730.

Message retention:

Select this check box to specify that unwanted messages are to be left on the queue. Otherwise, unwanted messages are dealt with according to their disposition options.

Data type	Enum
Units	Not applicable
Default	Cleared
Range	**Selected** Unwanted messages are left on the queue. **Cleared** Unwanted messages are dealt with according to their disposition options.

XA Enabled:

Specifies whether the connection factory is for XA or non-XA coordination of messages and controls if the application server uses XA QCF/TCF. Enable XA if multiple resources are not used in the same transaction.

If you set this property to `NON_XA`, the JMS session is still enlisted in a transaction, but uses the resource manager local transaction calls (session.commit and

session.rollback) instead of XA calls. This can lead to an improvement in performance. However, this means that only a single resource can be enlisted in a transaction in WebSphere Application Server.

5.1+ In WBI Server Foundation the last participant support enables you to enlist one non-XA resource with other XA-capable resources.

Data type	Enum
Units	Not applicable
Default	XA enabled
Range	**XA** The connection factory is for XA-coordination of messages **NON_XA** The connection factory is for non-XA coordination of messages
Recommended	Do not enable XA when the message queue received is the only resource in the transaction. Enable XA when other resources, including other queues or topics, are involved.

Connection pool:

Specifies an optional set of connection pool settings.

Connection pool properties are common to all J2C connectors.

The application server pools connections and sessions with the JMS provider to improve performance. This is independent from any WebSphere MQ connection pooling. You need to configure the connection and session pool properties appropriately for your applications, otherwise you may not get the connection and session behavior that you want.

Change the size of the connection pool if concurrent server-side access to the JMS resource exceeds the default value. The size of the connection pool is set on a per queue or topic basis.

Session pool:

An optional set of session pool settings.

This link provides a panel of optional connection pool properties, common to all J2C connectors.

The application server pools connections and sessions with the JMS provider to improve performance. This is independent from any WebSphere MQ connection pooling. You need to configure the connection and session pool properties appropriately for your applications, otherwise you may not get the connection and session behavior that you want.

Custom properties:

An optional set of name and value pairs for custom properties passed to WebSphere MQ.

You can use the Custom properties page to define the following WebSphere MQ Secure Sockets Layer (SSL) properties. These properties apply only if you set the property **Transport type**=CLIENT.

- SSLCIPHERSUITE
- SSLCRL
- SSLPEERNAME

For more information about setting these properties, see the section SSL properties in the *WebSphere MQ Using Java* book.

WebSphere MQ queue connection factory custom properties:

Use this panel to view or change an optional set of name and value pairs for custom properties of the selected queue connection factory for use with the WebSphere MQ JMS provider. These custom properties are passed to WebSphere MQ.

To view this administrative console page, click **Resources-> WebSphere MQ JMS Provider-> (In content pane, under Additional Properties) WebSphere MQ Queue Connection Factories->** ***connection_factory*****-> Custom Properties**

You can use the Custom properties page to define the following WebSphere MQ Secure Sockets Layer (SSL) properties. These properties apply only if you set the property **Transport type**=CLIENT on the connection factory.

- SSLCIPHERSUITE
- SSLCRL
- SSLPEERNAME

For more information about setting these properties for WebSphere MQ, see the section SSL properties in the *WebSphere MQ Using Java* book.

SSLCIPHERSUITE:

The cipher suite to use for SSL connection.

Set this property to a valid cipher suite provided by your JSSE provider; it must match the CipherSpec named on the SVRCONN channel named by the **Channel** property.

You must set this property if the SSLPEERNAME is to be set.

SSLCRL:

A list of zero or more CRL (Certificate Revocation List) servers used to check for SSL certificate revocation. (Use of this property requires a WebSphere MQ JVM at Java 2 version 1.4.)

The value is a space-delimited list of entries of the form:

`ldap://`*`hostname`*`:[`*`port`*`]`

optionally followed by a single / (forward slash). If *port* is omitted, the default LDAP port of 389 is assumed. At connect-time, the SSL certificate presented by the server is checked against the specified CRL servers. For more information about CRL security, see the section “Working with Certificate Revocation Lists” in the

WebSphere MQ Security book; for example at:
http://publibfp.boulder.ibm.com/epubs/html/csqzas01/csqzas012w.htm#IDX2254.

SSLPEERNAME:

For SSL, a *distinguished name* skeleton that must match the name provided by the WebSphere MQ queue manager. The distinguished name is used to check the identifying certificate presented by the server at connect-time.

If SSLPEERNAME is not set, such checking is performed. SSLPEERNAME is ignored if SSLCIPHERSUITE is not specified.

The SSLPEERNAME property is a list of attribute name and value pairs separated by commas or semicolons. For example:

```
SSLPEERNAME(CN=QMGR.*, OU=IBM, OU=WEBSPHERE)
```

The example given checks the identifying certificate presented by the server at connect-time. For the connection to succeed, the certificate must have a Common Name beginning QMGR., and must have at least two Organizational Unit names, the first of which is IBM and the second WEBSPHERE. Checking is not case-sensitive.

For more details about distinguished names and their use with WebSphere MQ, see the *WebSphere MQ Security* book; for example, the section "Distinguished Names" at
http://publibfp.boulder.ibm.com/epubs/html/csqzas01/csqzas010p.htm#HDRDCDN.

WebSphere MQ topic connection factory collection:

The topic connection factories configured in the WebSphere MQ JMS provider for publish/subscribe messaging with JMS topics.

This panel shows a list of the WebSphere MQ topic connection factories with a summary of their configuration properties.

To view this administrative console page, use the administrative console to complete the following steps:
1. In the navigation pane, expand **Resources-> WebSphere MQ JMS Provider**.
2. Change the **Scope** check box to Cell, Node (for a JMS provider on a specific node), or Server, according to your needs.
3. In the Additional Properties list in the contents pane, select **WebSphere MQ Topic Connection Factory**.

To view or change the properties of a connection factory, select its name in the list displayed.

To act on one or more of the connection factories listed, click the check boxes next to the names of the objects that you want to act on, then use the buttons provided.

WebSphere MQ topic connection factory settings:

Use this panel to view or change the configuration properties of the selected topic connection factory for use with the WebSphere MQ JMS provider. These configuration properties control how connections are created to the associated JMS topic destination.

A topic connection factory is used to create JMS connections to topic destinations. The topic connection factory is created by the WebSphere MQ JMS provider. A topic connection factory for the WebSphere MQ JMS provider has the following properties.

Note:

- The property values that you specify must match the values that you specified when configuring WebSphere MQ JMS resources. For more information about configuring WebSphere MQ JMS resources, see the WebSphere MQ *Using Java* book.
- In WebSphere MQ, names can have a maximum of 48 characters, with the exception of channels which have a maximum of 20 characters.
- 5.1 + You can use the Custom properties page to define a subset of WebSphere MQ properties. For more information about setting those properties, see Custom properties.

To view this administrative console page, click **Resources-> WebSphere MQ JMS Provider-> (In content pane, under Additional Properties) WebSphere MQ Topic Connection Factories->** ***connection_factory***

Scope:

Specifies the level to which this resource definition is visible -- the cell, node, or server level.

Resources such as JMS Providers, Namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

Note that no matter what the scope of a defined resource, the resource's properties only apply at an individual server level. For example, if you define the scope of a data source at the Cell level, all users in that Cell can look up and use that data source, which is unique within that Cell. However, resource property settings are local to each server in the Cell. For example, if you set **Max Connections** to 10, then each server in that Cell can have 10 connections.

Cell The most general scope. Resources defined at the Cell scope are visible from all Nodes and servers, unless they are overridden. To view resources defined in the cell scope, do not specify a server or a node name in the scope selection form.

Node The default scope for most resource types. Resources defined at the Node scope override any duplicates defined at the Cell scope and are visible to all servers on the same node, unless they are overridden at a server scope on that node. To view resources defined in a node scope, do not specify a server, but select a node name in the scope selection form.

Server The most specific scope for defining resources. Resources defined at the Server scope override any duplicate resource definitions defined at the Cell scope or parent Node scope and are visible only to a specific server. To view resources defined in a server scope, specify a server name as well as a node name in the scope selection form.

When resources are created, they are always created into the current scope selected in the panel. To view resources in other scopes, specify a different node or server in the scope selection form.

Data type String

Name:

The name by which this topic connection factory is known for administrative purposes within IBM WebSphere Application Server. The name must be unique within the JMS provider.

Data type String

JNDI name:

The JNDI name that is used to bind the topic connection factory into the application server's name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Description:

A description of this topic connection factory for administrative purposes within IBM WebSphere Application Server.

Data type String
Default Null

Category:

A category used to classify or group this topic connection factory, for your IBM WebSphere Application Server administrative records.

Data type String

Component-managed Authentication Alias:

This alias specifies a user ID and password to be used to authenticate connection to a JMS provider for application-managed authentication.

This property provides a list of the J2C authentication data entry aliases that have been defined to WebSphere Application Server. You can select a data entry alias to be used to authenticate the creation of a new connection to the JMS provider.

If you have enabled global security for WebSphere Application Server, select the alias that specifies the user ID and password used to authenticate the creation of a new connection to the JMS provider. The use of this alias depends on the resource authentication (res-auth) setting declared in the connection factory resource reference of an application component's deployment descriptors.

Note: User IDs longer than 12 characters cannot be used for authentication with the embedded WebSphere JMS provider. For example, the default Windows

NT user ID, **Administrator**, is not valid for use with embedded WebSphere messaging, because it contains 13 characters. Therefore, an authentication alias for a WebSphere JMS provider connection factory must specify a user ID no longer than 12 characters.

Container-managed Authentication Alias:

This alias specifies a user ID and password to be used to authenticate connection to a JMS provider for container-managed authentication.

This property provides a list of the J2C authentication data entry aliases that have been defined to WebSphere Application Server. You can select a data entry alias to be used to authenticate the creation of a new connection to the JMS provider.

If you have enabled global security for WebSphere Application Server, select the alias that specifies the user ID and password used to authenticate the creation of a new connection to the JMS provider. The use of this alias depends on the resource authentication (res-auth) setting declared in the connection factory resource reference of an application component's deployment descriptors.

Note: User IDs longer than 12 characters cannot be used for authentication with the embedded WebSphere JMS provider. For example, the default Windows NT user ID, **Administrator**, is not valid for use with embedded WebSphere messaging, because it contains 13 characters. Therefore, an authentication alias for a WebSphere JMS provider connection factory must specify a user ID no longer than 12 characters.

Queue manager:

The name of the WebSphere MQ queue manager for this connection factory. Connections created by this factory connect to that queue manager.

Data type	String
Default	Null
Range	A valid WebSphere MQ queue manager name, as 1 through 48 ASCII characters

Host:

The name of the host on which the WebSphere MQ queue manager runs, for client connection only.

Data type	String
Default	Null
Range	A valid TCP/IP hostname

Port:

The TCP/IP port number used for connection to the WebSphere MQ queue manager, for client connection only.

This port must be configured on the WebSphere MQ queue manager.

Data type	Integer
Default	Null

Range A valid TCP/IP port number, configured on the WebSphere MQ queue manager.

Channel:

The name of the channel used for connection to the WebSphere MQ queue manager, for client connection only.

Data type String
Default Null
Range 1 through 20 ASCII characters

Transport type:

Specifies whether the WebSphere MQ client connection or JNI bindings are used for connection to the WebSphere MQ queue manager. The external JMS provider controls the communication protocols between JMS clients and JMS servers. Tune the transport type when you are using non-ASF nonpersistent, non-durable, non-transactional messaging or when you want to satisfy security issues and the client is local to the queue manager node.

Data type Enum
Units Not applicable
Default BINDINGS
Range

BINDINGS
JNI bindings are used to connect to the queue manager. BINDINGS is a shared memory protocol and can only be used when the queue manager is on the same node as the JMS client and comes at some security risks that should be addressed through the use of EJB roles.

CLIENT
WebSphere MQ client connection is used to connect to the queue manager. CLIENT is a typical TCP-based protocol.

DIRECT
For WebSphere MQ Event Broker using DIRECT mode. DIRECT is a lightweight sockets protocol used in non-transactional, non-durable and non-persistent Publish/Subscribe messaging. DIRECT works only for clients and message-driven beans using the non-ASF protocol.

Recommended	DIRECT is the fastest transport type and should be used where possible. Use BINDINGS when you want to satisfy additional security tasks and the queue manager is local to the JMS client. QUEUED is fallback for all other cases. **Note:** WebSphere MQ 5.3 before CSD2 with the DIRECT setting can lose messages when used with message-driven beans and under load. This also happens with client-side based applications unless the broker's maxClientQueueSize is set to 0. You can set this to 0 with the command `#wempschangeproperties WAS_nodeName_server1 -e default -o DynamicSubscriptionEngine -n maxClientQueueSize -v 0 -x executionGroupUUID`, where executionGroupUUID can be found by starting the broker and looking in the Event Log/Applications for event 2201. This value is usually ffffffff-0000-0000-000000000000.

Broker control queue:

The name of the broker's control queue, to which all command messages (except publications and requests to delete publications) are sent

The name of the broker's control queue. Publisher and subscriber applications, and other brokers, send all command messages (except publications and requests to delete publications) to this queue.

Data type	String
Units	En_US ASCII characters
Default	Null
Range	1 through 48 ASCII characters

Broker queue manager:

The name of the WebSphere MQ queue manager that provides the publish/subscribe message broker.

Data type	String
Units	En_US ASCII characters
Default	Null
Range	1 through 48 ASCII characters

Broker publication queue:

The name of the broker's input queue that receives all publication messages for the default stream

The name of the broker's input queue (stream queue) that receives all publication messages for the default stream. Applications can also send requests to delete publications on the default stream to this queue.

Data type	String
Units	En_US ASCII characters
Default	Null
Range	1 through 48 ASCII characters

Broker subscription queue:

The name of the broker's queue from which non-durable subscription messages are retrieved

The name of the broker's queue from which non-durable subscription messages are retrieved. The subscriber specifies the name of the queue when it registers a subscription.

Data type	String
Units	En_US ASCII characters
Default	Null
Range	1 through 48 ASCII characters

Broker CC subscription queue:

The name of the broker's queue from which non-durable subscription messages are retrieved for a ConnectionConsumer. This property applies only for use of the Web container.

The name of the broker's queue from which non-durable subscription messages are retrieved for a ConnectionConsumer. This property applies only for use of the Web container.

Data type	String
Units	En_US ASCII characters
Default	Null
Range	1 through 48 ASCII characters

Broker version:

Whether the message broker is provided by the WebSphere MQ MA0C Supportpac or newer versions of WebSphere message broker products

Data type	Enum
Units	Not applicable
Default	Advanced
Range	**Advanced** The message broker is provided by newer versions of WebSphere message broker products, such as WebsSphere MQ Integrator and EventBroker. **Basic** The message broker is provided by the WebSphere MQ MA0C SupportPac (MQSeries - Publish/Subscribe) or MQSI working in MA0C compatibility mode.

Model queue definition:

The name of the model queue definition that the broker can use to create dynamic queues for non-default streams if the stream queue does not already exist

The name of the model queue definition that the broker can use to create dynamic queues to receive publications for streams other than the default stream. This is only used if the stream queue does not already exist. If this model queue definition does not exist, all stream queues must be defined by the administrator.

Data type	String
Units	En_US ASCII characters
Default	Null
Range	1 through 48 ASCII characters

CCSID:

The coded character set identifier for use with the WebSphere MQ queue manager.

This coded character set identifier (CCSID) must be one of the CCSIDs supported by WebSphere MQ.

Data type	String
Units	Integer
Default	Null
Range	1 through 65535

For more information about supported CCSIDs, and about converting between message data from one coded character set to another, see the *WebSphere MQ System Administration* and the *WebSphere MQ Application Programming Reference* books. These are available from the WebSphere MQ messaging multiplatform and platform-specific books Web pages; for example, at the IBM Publications Center, or from the WebSphere MQ collection kit, SK2T-0730.

Clone Support:

Select this checkbox to enable clone support to allow the same durable subscription across topic clones.

Data type	Enum
Units	Not applicable
Default	Cleared
Range	**Selected** Clone support is enabled. **Cleared** Clone support is disabled.

If you select this property, you must also specify a value for the **Client ID** property.

Client ID:

The JMS client identifier used for connections to the queue manager.

Data type	String

Range A valid JMS client ID, as ASCII characters

XA Enabled:

Specifies whether the connection factory is for XA or non-XA coordination of messages and controls if the application server uses XA QCF/TCF. Enable XA if multiple resources are not used in the same transaction.

If you set this property to NON_XA, the JMS session is still enlisted in a transaction, but uses the resource manager local transaction calls (session.commit and session.rollback) instead of XA calls. This can lead to an improvement in performance. However, this means that only a single resource can be enlisted in a transaction in WebSphere Application Server.

5.1 + In WBI Server Foundation the last participant support enables you to enlist one non-XA resource with other XA-capable resources.

For a WebSphere Topic Connection Factory with the **Port** property set to DIRECT this property does not apply, and always takes the value NON_XA.

Data type	Enum
Units	Not applicable
Default	XA enabled
Range	**XA** The connection factory is for XA-coordination of messages **NON_XA** The connection factory is for non-XA coordination of messages
Recommended	Do not enable XA when the message queue or topic received is the only resource in the transaction. Enable XA when other resources, including other queues or topics, are involved.

Connection pool:

Specifies an optional set of connection pool settings.

Connection pool properties are common to all J2C connectors.

The application server pools connections and sessions with the JMS provider to improve performance. This is independent from any WebSphere MQ connection pooling. You need to configure the connection and session pool properties appropriately for your applications, otherwise you may not get the connection and session behavior that you want.

Change the size of the connection pool if concurrent server-side access to the JMS resource exceeds the default value.The size of the connection pool is set on a per queue or topic basis. See the following table for details.

Session pool:

An optional set of session pool settings.

This link provides a panel of optional connection pool properties, common to all J2C connectors.

The application server pools connections and sessions with the JMS provider to improve performance. This is independent from any WebSphere MQ connection pooling. You need to configure the connection and session pool properties appropriately for your applications, otherwise you may not get the connection and session behavior that you want.

Custom properties:

An optional set of name and value pairs for custom properties passed to WebSphere MQ.

You can use the Custom properties page to define the following WebSphere MQ properties. These properties apply only if you set the property **Transport type**=CLIENT.

- SSLCIPHERSUITE
- SSLCRL
- SSLPEERNAME
- MSGSELECTION
- SUBSTORE

WebSphere MQ topic connection factory custom properties:

Use this panel to view or change an optional set of name and value pairs for custom properties of the selected topic connection factory for use with the WebSphere MQ JMS provider. These custom properties are passed to WebSphere MQ.

To view this administrative console page, click **Resources-> WebSphere MQ JMS Provider-> (In content pane, under Additional Properties) WebSphere MQ Topic Connection Factories->** ***connection_factory*****-> Custom Properties**

You can use the Custom properties page to define the following WebSphere MQ properties. These properties apply only if you set the property **Transport type**=CLIENT.

- SSLCIPHERSUITE
- SSLCRL
- SSLPEERNAME
- MSGSELECTION
- SUBSTORE

For more information about setting the SSL properties for WebSphere MQ, see the section SSL properties in the *WebSphere MQ Using Java* book.

SSLCIPHERSUITE:

The cipher suite to use for SSL connection.

Set this property to a valid cipher suite provided by your JSSE provider; it must match the CipherSpec named on the SVRCONN channel named by the **Channel** property.

You must set this property if the SSLPEERNAME is to be set.

SSLCRL:

A list of zero or more CRL (Certificate Revocation List) servers used to check for SSL certificate revocation. (Use of this property requires a WebSphere MQ JVM at Java 2 version 1.4.)

The value is a space-delimited list of entries of the form:
`ldap://hostname:[port]`

optionally followed by a single / (forward slash). If *port* is omitted, the default LDAP port of 389 is assumed. At connect-time, the SSL certificate presented by the server is checked against the specified CRL servers. For more information about CRL security, see the section "Working with Certificate Revocation Lists" in the *WebSphere MQ Security book*; for example at:
http://publibfp.boulder.ibm.com/epubs/html/csqzas01/csqzas012w.htm#IDX2254.

SSLPEERNAME:

For SSL, a *distinguished name* skeleton that must match the name provided by the WebSphere MQ queue manager. The distinguished name is used to check the identifying certificate presented by the server at connect-time.

If SSLPEERNAME is not set, such checking is performed. SSLPEERNAME is ignored if SSLCIPHERSUITE is not specified.

The SSLPEERNAME property is a list of attribute name and value pairs separated by commas or semicolons. For example:
`SSLPEERNAME(CN=QMGR.*, OU=IBM, OU=WEBSPHERE)`

The example given checks the identifying certificate presented by the server at connect-time. For the connection to succeed, the certificate must have a Common Name beginning QMGR., and must have at least two Organizational Unit names, the first of which is IBM and the second WEBSPHERE. Checking is not case-sensitive.

For more details about distinguished names and their use with WebSphere MQ, see the *WebSphere MQ Security* book; for example, the section "Distinguished Names" at
http://publibfp.boulder.ibm.com/epubs/html/csqzas01/csqzas010p.htm#HDRDCDN.

MSGSELECTION:

Determines whether message selection is done by the JMS Client or by the Broker.

If you set the property **Transport type**=BINDINGS, message selection is always done by the Broker and the setting of this property is ignored.

Data type	String
Default	CLIENT

Range

CLIENT
Message selection is done by the JMS Client.

BROKER
Message selection is done by the Broker.

SUBSTORE:

Where WebSphere MQ JMS should store persistent data relating to active subscriptions.

Data type String
Default MIGRATE

Range

BROKER
Subscription information is stored by the publish/subscribe broker used by the application.

If a non-durable subscriber fails, the subscription is deregistered from the broker as soon as possible. The broker adds a response to this deregistration onto the SYSTEM.JMS.REPORT.QUEUE, which is used to clean up after the failed subscriber. With SUBSTORE(BROKER), a separate cleanup thread is run regularly in the background of each JMS publish/subscribe application.

MIGRATE
This option dynamically selects the queue-based or broker-based subscription store based on the levels of queue manager and publish/subscribe broker installed. If both queue manager and broker are capable of supporting SUBSTORE(BROKER), this behaves as SUBSTORE(BROKER); otherwise it behaves as SUBSTORE(QUEUE). Additionally, SUBSTORE(MIGRATE) transfers durable subscription information from the queue-based subscription store to the broker-based store.

This provides an easy migration path from older versions of WebSphere MQ JMS, WebSphere MQ, and publish/subscribe broker.

QUEUE
Subscription information is stored on SYSTEM.JMS.ADMIN.QUEUE and SYSTEM.JMS.PS.STATUS.QUEUE on the local queue manager. SUBSTORE(QUEUE) is provided for compatibility with versions of MQSeries JMS.

For more information about the use of this property in WebSphere MQ, see the section Subscription stores in the *WebSphere MQ Using Java* book.

WebSphere MQ queue destination collection:

The queue destinations configured in the WebSphere MQ JMS provider for point-to-point messaging with JMS queues.

This panel shows a list of the WebSphere MQ queue destinations with a summary of their configuration properties.

To view this administrative console page, use the administrative console to complete the following steps:
1. In the navigation pane, expand **Resources-> WebSphere MQ JMS provider**.
2. Change the **Scope** check box to Cell, Node (for a JMS provider on a specific node), or Server, according to your needs.
3. In the Additional Properties list in the contents pane, select **WebSphere MQ Queue Destination**.

To view or change the properties of a queue destination, select its name in the list displayed.

To act on one or more of the queue destinations listed, click the check boxes next to the names of the objects that you want to act on, then use the buttons provided.

WebSphere MQ queue settings:

Use this panel to view or change the configuration properties of the selected queue destination for use with the WebSphere MQ JMS provider.

A queue destination is used to configure the properties of a JMS queue. Connections to the queue are created by the associated queue connection factory for the WebSphere MQ JMS provider. A queue for use with the WebSphere MQ JMS provider has the following properties.

Note:
- The property values that you specify must match the values that you specified when configuring WebSphere MQ JMS resources. For more information about configuring WebSphere MQ JMS resources, see the WebSphere MQ *Using Java* book.
- In WebSphere MQ, names can have a maximum of 48 characters, with the exception of channels which have a maximum of 20 characters.

To view this administrative console page, click **Resources-> WebSphere MQ JMS Provider-> (In content pane, under Additional Properties) WebSphere MQ Queue Destinations->** ***destination_name***

Scope:

Specifies the level to which this resource definition is visible -- the cell, node, or server level.

Resources such as JMS Providers, Namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

Note that no matter what the scope of a defined resource, the resource's properties only apply at an individual server level. For example, if you define the scope of a data source at the Cell level, all users in that Cell can look up and use that data source, which is unique within that Cell. However, resource property settings are local to each server in the Cell. For example, if you set **Max Connections** to 10, then each server in that Cell can have 10 connections.

Cell The most general scope. Resources defined at the Cell scope are visible from all Nodes and servers, unless they are overridden. To view resources defined in the cell scope, do not specify a server or a node name in the scope selection form.

Node The default scope for most resource types. Resources defined at the Node

scope override any duplicates defined at the Cell scope and are visible to all servers on the same node, unless they are overridden at a server scope on that node. To view resources defined in a node scope, do not specify a server, but select a node name in the scope selection form.

Server The most specific scope for defining resources. Resources defined at the Server scope override any duplicate resource definitions defined at the Cell scope or parent Node scope and are visible only to a specific server. To view resources defined in a server scope, specify a server name as well as a node name in the scope selection form.

When resources are created, they are always created into the current scope selected in the panel. To view resources in other scopes, specify a different node or server in the scope selection form.

Data type String

Name:

The name by which the queue is known for administrative purposes within IBM WebSphere Application Server.

Data type String

JNDI name:

The JNDI name that is used to bind the queue into the application server's name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Description:

A description of the queue, for administrative purposes

Data type String
Default Null

Category:

A category used to classify or group this queue, for your IBM WebSphere Application Server administrative records.

Data type String

Persistence:

Whether all messages sent to the destination are persistent, non-persistent, or have their persistence defined by the application

Data type Enum
Units Not applicable
Default APPLICATION_DEFINED
Range
Application defined
Messages on the destination have their persistence defined by the application that put them onto the queue.
Queue defined
[WebSphere MQ destination only] Messages on the destination have their persistence defined by the WebSphere MQ queue definition properties.
Persistent
Messages on the destination are persistent.
Non persistent
Messages on the destination are not persistent.

Priority:

Whether the message priority for this destination is defined by the application or the **Specified priority** property

Data type Enum
Units Not applicable
Default APPLICATION_DEFINED
Range
Application defined
The priority of messages on this destination is defined by the application that put them onto the destination.
Queue defined
[WebSphere MQ destination only] Messages on the destination have their persistence defined by the WebSphere MQ queue definition properties.
Specified
The priority of messages on this destination is defined by the **Specified priority** property.*If you select this option, you must define a priority on the **Specified priority** property.*

Specified priority:

If the **Priority** property is set to `Specified`, type here the message priority for this queue, in the range 0 (lowest) through 9 (highest)

If the **Priority** property is set to `Specified`, messages sent to this queue have the priority value specified by this property.

Data type	Integer
Units	Message priority level
Default	Null
Range	0 (lowest priority) through 9 (highest priority)

Expiry:

Whether the expiry timeout for this queue is defined by the application or the **Specified expiry** property, or messages on the queue never expire (have an unlimited expiry timeout)

Data type	Enum
Units	Not applicable
Default	APPLICATION_DEFINED
Range	**Application defined** The expiry timeout for messages on this queue is defined by the application that put them onto the queue. **Specified** The expiry timeout for messages on this queue is defined by the **Specified expiry** property.*If you select this option, you must define a timeout on the **Specified expiry** property.* **Unlimited** Messages on this queue have no expiry timeout, so those messages never expire.

Specified expiry:

If the **Expiry timeout** property is set to `Specified`, type here the number of milliseconds (greater than 0) after which messages on this queue expire

Data type	Integer
Units	Milliseconds
Default	Null
Range	Greater than or equal to 0 • 0 indicates that messages never timeout • Other values are an integer number of milliseconds

Base queue name:

The name of the queue to which messages are sent, on the queue manager specified by the **Base queue manager name** property

Data type	String
Default	Null
Range	1 through 48 ASCII characters

Base queue manager name:

The name of the WebSphere MQ queue manager to which messages are sent

This queue manager provides the queue specified by the **Base queue name** property.

Data type	String
Units	En_US ASCII characters
Default	Null
Range	A valid WebSphere MQ Queue Manager name, as 1 through 48 ASCII characters

CCSID:

The coded character set identifier for use with the WebSphere MQ queue manager.

This coded character set identifier (CCSID) must be one of the CCSIDs supported by WebSphere MQ.

Data type	String
Units	Integer
Default	Null
Range	1 through 65535

For more information about supported CCSIDs, and about converting between message data from one coded character set to another, see the *WebSphere MQ System Administration* and the *WebSphere MQ Application Programming Reference* books. These are available from the WebSphere MQ messaging multiplatform and platform-specific books Web pages; for example, at http://www-3.ibm.com/software/ts/mqseries/library/manualsa/manuals/platspecific.html, the IBM Publications Center, or from the WebSphere MQ collection kit, SK2T-0730.

Use native encoding:

Select this checkbox to indicate that the queue destination should use native encoding (appropriate encoding values for the Java platform).

Data type	Enum
Units	Not applicable
Default	Cleared
Range	**Cleared** Native encoding is not used, so specify the properties below for integer, decimal, and floating point encoding. **Selected** Native encoding is used (to provide appropriate encoding values for the Java platform). For more information about encoding properties, see the WebSphere MQ *Using Java* document.

Integer encoding:

If native encoding is not enabled, select whether integer encoding is normal or reversed.

Data type Enum
Units Not applicable
Default NORMAL
Range
NORMAL
Normal integer encoding is used.
REVERSED
Reversed integer encoding is used.

For more information about encoding properties, see the WebSphere MQ *Using Java* document.

Decimal encoding:

If native encoding is not enabled, select whether decimal encoding is normal or reversed.

Data type Enum
Units Not applicable
Default NORMAL
Range
NORMAL
Normal decimal encoding is used.
REVERSED
Reversed decimal encoding is used.

For more information about encoding properties, see the WebSphere MQ *Using Java* document.

Floating point encoding:

If native encoding is not enabled, select the type of floating point encoding.

Data type Enum
Units Not applicable
Default IEEENORMAL
Range
IEEENORMAL
IEEE normal floating point encoding is used.
IEEEREVERSED
IEEE reversed floating point encoding is used.
S390 S390 floating point encoding is used.

For more information about encoding properties, see the WebSphere MQ *Using Java* document.

Target client:

Whether the receiving application is JMS-compliant or is a traditional WebSphere MQ application

Data type Enum

Units Not applicable
Default MQSeries
Range **MQSeries**
The target is a non-JMS, traditional WebSphere MQ application.
JMS The target is a JMS-compliant application.

Queue manager host:

The name of host for the queue manager on which the queue destination is created.

Data type String
Default Null
Range A valid TCP/IP hostname

Queue manager port:

The number of the port used by the queue manager on which this queue is defined.

Data type String
Units A valid TCP/IP port number.
Default Null
Range A valid TCP/IP port number. This port must be configured on the WebSphere MQ queue manager.

Server connection channel name:

The name of the channel used for connection to the WebSphere MQ queue manager.

Data type String
Default Null
Range 1 through 20 ASCII characters

User name:

The user ID used, with the **Password** property, for authentication when connecting to the queue manager to define the queue destination.

If you specify a value for the **User name** property, you must also specify a value for the **Password** property.

Data type String
Default Null

Password:

The password, used with the **User name** property, for authentication when connecting to the queue manager to define the queue destination.

If you specify a value for the **User name** property, you must also specify a value for the **Password** property.

Data type String
Default Null

WebSphere MQ queue settings (MQ Config):

Use this panel to view or change the configuration properties defined to WebSphere MQ for the selected queue destination.

A queue destination is used to configure the properties of a JMS queue. A queue for use with the WebSphere MQ JMS provider has the following extra properties defined to WebSphere MQ.

Notes

Note:

- Some properties displayed are read-only and cannot be changed.
- These MQ Config properties can be used only to view or change the properties of local queues. You cannot use MQ Config to administer alias or remote queues.
- To be able to view or change properties, the WebSphere MQ Queue Manager on which the queue resides must be configured for remote administration and be running.
- The property values that you specify must match the values that you specified when configuring WebSphere MQ JMS resources. For more information about configuring WebSphere MQ JMS resources, see the *WebSphere MQ: Using Java* book; for example from the WebSphere MQ multiplatform library Web page at http://www.ibm.com/software/ts/mqseries/library/manualsa/manuals/crosslatest.html.
- In WebSphere MQ, names can have a maximum of 48 characters, with the exception of channels which have a maximum of 20 characters.

To view this administrative console page, click **Resources-> WebSphere MQ JMS Provider-> (In content pane, under Additional Properties) WebSphere MQ Queue Destinations->** ***destination_name*****-> (In content pane, under Additional Properties) MQ Config**

Base queue name:

The name of the local queue to which messages are sent, on the queue manager specified by the **Base queue manager name** property.

Data type String

Base queue manager name:

The name of the WebSphere MQ queue manager to which messages are sent.

This queue manager provides the queue specified by the **Base queue name** property.

Data type String

Queue manager host:

The name of host for the queue manager on which the queue destination is created.

Data type String

Queue manager port:

The number of the port used by the queue manager on which this queue is defined.

Data type Integer
Range A valid TCP/IP port number. This port must be configured on the WebSphere MQ queue manager.

Server connection channel name:

The name of the channel used for connection to the WebSphere MQ queue manager.

Data type String
Range 1 through 20 ASCII characters

User ID:

The user ID used, with the **Password** property, for authentication when connecting to the queue manager to define the queue destination.

If you specify a value for the **User name** property, you must also specify a value for the **Password** property.

Data type String

Password:

The password, used with the **User name** property, for authentication when connecting to the queue manager to define the queue destination.

If you specify a value for the **User name** property, you must also specify a value for the **Password** property.

Data type String

Name:

The name of the queue defined to the WebSphere MQ queue manager.

Data type String
Range 1 through 48 ASCII characters.

Description:

The WebSphere MQ queue description, for administrative purposes within WebSphere MQ.

Data type String
Default Null
Range 1 through 64 ASCII characters.

Inhibit Put:

Whether or not put operations are allowed for this queue.

Data type Enum
Units Not applicable
Default Cleared
Range
Allowed
Put operations are allowed for this queue.
Not allowed
Put operations are not allowed for this queue.

Persistence:

Whether all messages sent to the destination are persistent, non-persistent, or have their persistence defined by the application

Data type Enum
Units Not applicable
Default APPLICATION_DEFINED
Range
Application defined
Messages on the destination have their persistence defined by the application that put them onto the queue.
Queue defined
[WebSphere MQ destination only] Messages on the destination have their persistence defined by the WebSphere MQ queue definition properties.
Persistent
Messages on the destination are persistent.
Non persistent
Messages on the destination are not persistent.

Cluster name:

The name of the cluster to which the WebSphere MQ queue manager belongs.

If you specify a value for **Cluster name**, you cannot specify a value for **Cluster name list**. Cluster names must conform to the rules described in the *WebSphere MQ MQSC Command Reference* book.

Data type String

Default Null
Range A valid WebSphere MQ name for a queue manager cluster, as 1 through 48 ASCII characters

Cluster name list:

The name of the cluster namelist to which the WebSphere MQ queue manager belongs.

If you specify a value for **Cluster name**, you cannot specify a value for **Cluster name list**.

Data type String
Default Null
Range A valid WebSphere MQ name for a list of queue manager clusters, as 1 through 48 ASCII characters

Default Binding:

The default binding to be used when the queue is defined as a cluster queue.

Data type Enum
Default Null
Range A
On open
The queue handle is bound to a specific instance of the cluster queue when the queue is opened.
Not fixed
The queue handle is not bound to any particular instance of the cluster queue. This allows the queue manager to select a specific queue instance when the message is put, and to change that selection subsequently should the need arise.

Inhibit Get:

Whether or not get operations are allowed for this queue.

Data type Enum
Units Not applicable
Default Cleared
Range **Allowed**
Get operations are allowed for this queue.
Not allowed
Get operations are not allowed for this queue.

Maximum queue depth:

The maximum number of messages allowed on the queue.

Data type Integer
Units Messages
Default
Range A value greater than or equal to zero, and less than or equal to:
- On AIX, Compaq OpenVMS, HP-UX, Linux, OS/400, Solaris, Windows, and z/OS, specify a value in the range zero through 999 999 999.
- On any other WebSphere MQ platform, specify a value in the range zero through 640 000.

For more information about the maximum value allowed, see the *WebSphere MQ MQSC Command Reference*.

If this value is reduced, any message that is already on the queue are not affected, even if the number of messages exceeds the new maximum.

Maximum Message Length:

The maximum length, in bytes, of messages on this queue.

Data type Integer
Units Bytes
Default
Range A value greater than or equal to zero, and less than or equal to the maximum message length for the queue manager and WebSphere MQ platform. For more information about the maximum value allowed, see the *WebSphere MQ MQSC Command Reference*.

If this value is reduced, any message that is already on the queue are not affected, even if the message length exceeds the new maximum.

Shareability:

Whether multiple applications can get messages from this queue.

Data type Enum
Units Not applicable
Default Not shareable
Range
Not shareable
Only one application instance can get messages from the queue.
Shareable
More than one application instance can get messages from the queue.

Input Open Option:

The default share option for applications opening this queue for input

Data type Enum
Units Not applicable
Default Cleared
Range
Exclusive The open request is for exclusive input from the queue.
Shared The open request is for shared input from the queue.

Message Delivery Sequence:

The order in which messages are delivered from the queue in response to get requests.

Data type Enum
Units Not applicable
Default Priority
Range
Priority Messages are delivered in first-in-first-out (FIFO) order within priority. This is the default supplied with WebSphere MQ, but your installation might have changed it.
FIFO Messages are delivered in FIFO order. Priority is ignored for messages on this queue.

Backout threshold:

The maximum number of times that a message can be backed out. If this threshold is reached, the message is requeued on the backout queue specified by the **Backout Requeue name** property.

The WebSphere MQ queue manager keeps a record of the number of times that each message has been backed out. When this number reaches a configurable threshold, the connection consumer requeues the message on a named backout queue. If this requeue fails for any reason, the message is removed from the queue and either requeued to the dead-letter queue, or discarded.

Data type Integer
Default 0
Range
0 Never requeue messages
1 or more The number of times that a message has been backed, at which the message is requeued on a named backout queue.

Backout Requeue name:

The name of the backout queue to which messages are requeued if they have been backed out more than the backout threshold.

The WebSphere MQ queue manager keeps a record of the number of times that each message has been backed out. When this number reaches a configurable threshold, the connection consumer requeues the message on a named backout queue. If this requeue fails for any reason, the message is removed from the queue and either requeued to the dead-letter queue, or discarded.

Data type String
Default Null
Range 1 through 48 characters.

Harden Get Backout:

Whether hardening should be used to ensure that the count of the number of times that a message has been backed out is accurate.

Data type Enum
Units Not applicable
Default Cleared
Range
Not hardened
The count is not hardened. This is the default supplied with WebSphere MQ, but your installation might have changed it.
Hardened
The count is hardened.

WebSphere MQ topic destination collection:

The topic destinations configured in the WebSphere MQ JMS provider for publish/subscribe messaging with JMS topics. Use this panel to create or delete topic destinations, or to select a topic destination to view or change its configuration properties.

This panel shows a list of the WebSphere MQ topic destinations with a summary of their configuration properties.

To view this administrative console page, use the administrative console to complete the following steps:
1. In the navigation pane, select **Resources-> WebSphere MQ JMS Provider**.
2. Change the **Scope** check box to Cell, Node (for a JMS provider on a specific node), or Server, according to your needs.
3. In the Additional Properties list in the contents pane, select **WebSphere MQ Topic Destination**.

To view or change the properties of a topic destination, select its name in the list displayed.

To act on one or more of the topic destinations listed, click the check boxes next to the names of the objects that you want to act on, then use the buttons provided.

WebSphere MQ topic settings:

Use this panel to view or change the configuration properties of the selected topic destination for use with the WebSphere MQ JMS provider.

A topic destination is used to configure the properties of a JMS topic for the associated JMS provider. Connections to the topic are created by the associated topic connection factory. A topic for use with the WebSphere MQ JMS provider has the following properties.

Note:

- The property values that you specify must match the values that you specified when configuring WebSphere MQ JMS resources. For more information about configuring WebSphere MQ JMS resources, see the WebSphere MQ *Using Java* book.
- In WebSphere MQ, names can have a maximum of 48 characters, with the exception of channels which have a maximum of 20 characters.

To view this administrative console page, click **Resources-> WebSphere MQ JMS Provider-> (In content pane, under Additional Properties) WebSphere MQ Topic Destinations->** ***destination_name***

Name:

The name by which the topic is known for administrative purposes within IBM WebSphere Application Server.

Data type String

JNDI name:

The JNDI name that is used to bind the topic into the application server's name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Description:

A description of the topic, for administrative purposes within IBM WebSphere Application Server.

Data type String
Default Null

Category:

A category used to classify or group this topic, for your IBM WebSphere Application Server administrative records.

Data type String

Persistence:

Whether all messages sent to the destination are persistent, non-persistent, or have their persistence defined by the application

Data type Enum
Units Not applicable
Default APPLICATION_DEFINED
Range
Application defined
Messages on the destination have their persistence defined by the application that put them onto the queue.
Queue defined
[WebSphere MQ destination only] Messages on the destination have their persistence defined by the WebSphere MQ queue definition properties.
Persistent
Messages on the destination are persistent.
Non persistent
Messages on the destination are not persistent.

Priority:

Whether the message priority for this destination is defined by the application or the **Specified priority** property

Data type Enum
Units Not applicable
Default APPLICATION_DEFINED
Range
Application defined
The priority of messages on this destination is defined by the application that put them onto the destination.
Queue defined
[WebSphere MQ destination only] Messages on the destination have their persistence defined by the WebSphere MQ queue definition properties.
Specified
The priority of messages on this destination is defined by the **Specified priority** property.*If you select this option, you must define a priority on the **Specified priority** property.*

Specified priority:

If the **Priority** property is set to `Specified`, type here the message priority for this queue, in the range 0 (lowest) through 9 (highest)

If the **Priority** property is set to `Specified`, messages sent to this queue have the priority value specified by this property.

Data type Integer
Units Message priority level
Default Null
Range 0 (lowest priority) through 9 (highest priority)

Expiry:

Whether the expiry timeout for this queue is defined by the application or the **Specified expiry** property, or messages on the queue never expire (have an unlimited expiry timeout)

Data type Enum
Units Not applicable
Default APPLICATION_DEFINED
Range
Application defined
The expiry timeout for messages on this queue is defined by the application that put them onto the queue.
Specified
The expiry timeout for messages on this queue is defined by the **Specified expiry** property.*If you select this option, you must define a timeout on the **Specified expiry** property.*
Unlimited
Messages on this queue have no expiry timeout, so those messages never expire.

Specified expiry:

If the **Expiry timeout** property is set to `Specified`, type here the number of milliseconds (greater than 0) after which messages on this queue expire

Data type Integer
Units Milliseconds
Default Null
Range Greater than or equal to 0
- 0 indicates that messages never timeout
- Other values are an integer number of milliseconds

Base topic name:

The name of the topic to which messages are sent

Data type String
Range Depends on the broker used. For details, see the documentation for your broker; for example the *WebSphere MQ Event Broker* library at http://www-3.ibm.com/software/ts/mqseries/library/manualsa/manuals

CCSID:

The coded character set identifier for use with the WebSphere MQ queue manager.

This coded character set identifier (CCSID) must be one of the CCSIDs supported by WebSphere MQ.

Data type	String
Units	Integer
Default	Null
Range	1 through 65535

For more information about supported CCSIDs, and about converting between message data from one coded character set to another, see the *WebSphere MQ System Administration* and the *WebSphere MQ Application Programming Reference* books. These are available from the WebSphere MQ messaging multiplatform and platform-specific books Web pages; for example, at the IBM Publications Center, or from the WebSphere MQ collection kit, SK2T-0730.

Use native encoding:

Select this checkbox to indicate that the queue destination should use native encoding (appropriate encoding values for the Java platform).

Data type	Enum
Units	Not applicable
Default	Cleared
Range	**Cleared** Native encoding is not used, so specify the properties below for integer, decimal, and floating point encoding. **Selected** Native encoding is used (to provide appropriate encoding values for the Java platform). For more information about encoding properties, see the WebSphere MQ *Using Java* document.

Integer encoding:

If native encoding is not enabled, select whether integer encoding is normal or reversed.

Data type	Enum
Units	Not applicable
Default	NORMAL
Range	**NORMAL** Normal integer encoding is used. **REVERSED** Reversed integer encoding is used. For more information about encoding properties, see the WebSphere MQ *Using Java* document.

Decimal encoding:

If native encoding is not enabled, select whether decimal encoding is normal or reversed.

Data type Enum
Units Not applicable
Default NORMAL
Range **NORMAL**
Normal decimal encoding is used.
REVERSED
Reversed decimal encoding is used.

For more information about encoding properties, see the WebSphere MQ *Using Java* document.

Floating point encoding:

If native encoding is not enabled, select the type of floating point encoding.

Data type Enum
Units Not applicable
Default IEEENORMAL
Range **IEEENORMAL**
IEEE normal floating point encoding is used.
IEEEREVERSED
IEEE reversed floating point encoding is used.
S390 S390 floating point encoding is used.

For more information about encoding properties, see the WebSphere MQ *Using Java* document.

Target client type:

Whether the receiving application is JMS-compliant or is a traditional WebSphere MQ application

Data type Enum
Units Not applicable
Default MQ
Range **MQ** The target is a non-JMS, traditional WebSphere MQ application.
JMS The target is a JMS-compliant application.

Broker Dur Sub Queue:

The name of the broker's queue from which durable subscription messages are retrieved

The name of the broker's queue from which durable subscription messages are retrieved. The subscriber specifies the name of the queue when it registers a subscription.

Data type	String
Units	En_US ASCII characters
Default	Null
Range	1 through 48 ASCII characters

Broker CC Dur Sub Queue:

The name of the broker's queue from which durable subscription messages are retrieved for a ConnectionConsumer. This property applies only for use of the Web container.

The name of the broker's queue from which durable subscription messages are retrieved for a ConnectionConsumer. This property applies only for use of the Web container.

Data type	String
Units	En_US ASCII characters
Default	Null
Range	1 through 48 ASCII characters

JMS connection factory collection:

The JMS connection factories configured in the associated JMS provider for both point-to-point and publish/subscribe messaging. Use this panel to create or delete JMS connection factories, or to select a connection factory to view or change its configuration properties.

This panel shows a list of the generic JMS connection factories with a summary of their configuration properties.

To view this administrative console page, use the administrative console to complete the following steps:
1. In the navigation pane, expand **Resources-> Generic JMS Providers**.
2. Change the **Scope** check box to Cell, Node (for a JMS provider on a specific node), or Server, according to your needs.
3. In the Additional Properties list in the contents pane, select **Generic JMS Connection Factory**.

To view or change the properties of a JMS connection factory, select its name in the list displayed.

To act on one or more of the JMS connection factories listed, click the check boxes next to the names of the objects that you want to act on, then use the buttons provided.

Generic JMS connection factory settings:

Use this panel to view or change the configuration properties of the selected JMS connection factory for use with the associated JMS provider. These configuration properties control how connections are created to the associated JMS destination.

A JMS connection factory is used to create connections to JMS destinations. The JMS connection factory is created by the associated JMS provider. A JMS connection factory for a generic JMS provider (other than the embedded WebSphere JMS provider or the WebSphere MQ JMS provider) has the following properties:

To view this administrative console page, click **Resources-> Generic JMS Providers->** ***provider_name*****-> JMS Connection Factories->** ***connection_factory***

Scope:

Specifies the level to which this resource definition is visible -- the cell, node, or server level.

Resources such as JMS Providers, Namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

Note that no matter what the scope of a defined resource, the resource's properties only apply at an individual server level. For example, if you define the scope of a data source at the Cell level, all users in that Cell can look up and use that data source, which is unique within that Cell. However, resource property settings are local to each server in the Cell. For example, if you set **Max Connections** to 10, then each server in that Cell can have 10 connections.

Cell The most general scope. Resources defined at the Cell scope are visible from all Nodes and servers, unless they are overridden. To view resources defined in the cell scope, do not specify a server or a node name in the scope selection form.

Node The default scope for most resource types. Resources defined at the Node scope override any duplicates defined at the Cell scope and are visible to all servers on the same node, unless they are overridden at a server scope on that node. To view resources defined in a node scope, do not specify a server, but select a node name in the scope selection form.

Server The most specific scope for defining resources. Resources defined at the Server scope override any duplicate resource definitions defined at the Cell scope or parent Node scope and are visible only to a specific server. To view resources defined in a server scope, specify a server name as well as a node name in the scope selection form.

When resources are created, they are always created into the current scope selected in the panel. To view resources in other scopes, specify a different node or server in the scope selection form.

Data type String

Name:

The name by which this JMS connection factory is known for administrative purposes within IBM WebSphere Application Server. The name must be unique within the associated JMS provider.

Data type String

Type:

Whether this connection factory is for creating JMS queue destinations or JMS topic destinations.

Select one of the following options:
Queue
A JMS queue connection factory for point-to-point messaging.
Topic A JMS topic connection factory for publish/subscribe messaging.

JNDI name:

The JNDI name that is used to bind the connection factory into the application server's name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Description:

A description of this connection factory for administrative purposes within IBM WebSphere Application Server.

Data type String
Default Null

Category:

A category used to classify or group this connection factory, for your IBM WebSphere Application Server administrative records.

Data type String

Component-managed Authentication Alias:

This alias specifies a user ID and password to be used to authenticate connection to a JMS provider for application-managed authentication.

This property provides a list of the J2C authentication data entry aliases that have been defined to WebSphere Application Server. You can select a data entry alias to be used to authenticate the creation of a new connection to the JMS provider.

If you have enabled global security for WebSphere Application Server, select the alias that specifies the user ID and password used to authenticate the creation of a new connection to the JMS provider. The use of this alias depends on the resource authentication (res-auth) setting declared in the connection factory resource reference of an application component's deployment descriptors.

Note: User IDs longer than 12 characters cannot be used for authentication with the embedded WebSphere JMS provider. For example, the default Windows NT user ID, **Administrator**, is not valid for use with embedded WebSphere

messaging, because it contains 13 characters. Therefore, an authentication alias for a WebSphere JMS provider connection factory must specify a user ID no longer than 12 characters.

Container-managed Authentication Alias:

This alias specifies a user ID and password to be used to authenticate connection to a JMS provider for container-managed authentication.

This property provides a list of the J2C authentication data entry aliases that have been defined to WebSphere Application Server. You can select a data entry alias to be used to authenticate the creation of a new connection to the JMS provider.

If you have enabled global security for WebSphere Application Server, select the alias that specifies the user ID and password used to authenticate the creation of a new connection to the JMS provider. The use of this alias depends on the resource authentication (res-auth) setting declared in the connection factory resource reference of an application component's deployment descriptors.

Note: User IDs longer than 12 characters cannot be used for authentication with the embedded WebSphere JMS provider. For example, the default Windows NT user ID, **Administrator**, is not valid for use with embedded WebSphere messaging, because it contains 13 characters. Therefore, an authentication alias for a WebSphere JMS provider connection factory must specify a user ID no longer than 12 characters.

Mapping-Configuration Alias:

Allows users to select from the **Security** > **JAAS Configuration** > **Application Logins Configuration** list.

The **DefaultPrincipalMapping** JAAS configuration maps the authentication alias to the userid and password. You may define and use other mapping configurations. For more information about the mapping configurations, see the Information Center topic ″Java Authentication and Authorization service configuration entry settings.″

Data type Pick-list

Connection pool:

Specifies an optional set of connection pool settings.

Connection pool properties are common to all J2C connectors.

The application server pools connections and sessions with the JMS provider to improve performance. This is independent from any WebSphere MQ connection pooling. You need to configure the connection and session pool properties appropriately for your applications, otherwise you may not get the connection and session behavior that you want.

Change the size of the connection pool if concurrent server-side access to the JMS resource exceeds the default value.The size of the connection pool is set on a per queue or topic basis. See the following table for details.

Session pool:

An optional set of session pool settings.

This link provides a panel of optional connection pool properties, common to all J2C connectors.

The application server pools connections and sessions with the JMS provider to improve performance. This is independent from any WebSphere MQ connection pooling. You need to configure the connection and session pool properties appropriately for your applications, otherwise you may not get the connection and session behavior that you want.

Custom properties:

An optional set of name and value pairs for custom properties passed to the JMS provider.

Generic JMS destination collection:

The JMS destinations configured in the associated JMS provider for point-to-point and publish/subscribe messaging. Use this panel to create or delete JMS destinations, or to select a JMS destination to view or change its configuration properties.

To view this administrative console page, use the administrative console to complete the following steps:
1. In the navigation pane, expand **Resources-> Generic JMS Providers**.
2. Change the **Scope** check box to Cell, Node (for a JMS provider on a specific node), or Server, according to your needs.
3. In the Additional Properties list in the contents pane, select **Generic JMS Destination**.

To view or change the properties of a JMS destination, select its name in the list displayed.

To act on one or more of the JMS destinations listed, click the check boxes next to the names of the objects that you want to act on, then use the buttons provided.

Generic JMS destination settings:

Use this panel to view or change the configuration properties of the selected JMS destination for use with the associated JMS provider.

A JMS destination is used to configure the properties of a JMS destination for the associated generic JMS provider. Connections to the JMS destination are created by the associated JMS connection factory. A JMS destination for use with a generic JMS provider (not the embedded WebSphere JMS provider or WebSphere MQ JMS provider) has the following properties.

To view this administrative console page, click **Resources-> Generic JMS Providers->** ***provider_name*****-> JMS Destinations->** ***destination***

Name:

The name by which the queue is known for administrative purposes within IBM WebSphere Application Server.

Data type String

Type:

Whether this JMS destination is a queue (for point-to-point) or topic (for publish/subscribe).

Select one of the following options:
Queue
A JMS queue destination for point-to-point messaging.
Topic A JMS topic destination for publish/subscribe messaging.

JNDI name:

The JNDI name that is used to bind the queue into the application server's name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Description:

A description of the queue, for administrative purposes

Category:

A category used to classify or group this queue, for your IBM WebSphere Application Server administrative records.

Data type String

External JNDI name:

The JNDI name that is used to bind the queue into the application server's name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Asynchronous messaging - security considerations

This topic describes considerations that you should be aware of if you want to use security for asynchronous messaging with WebSphere Application Server.

Security for messaging operates as a part of the WebSphere Application Server global security, and is enabled only when global security is enabled.

When global security is enabled, JMS connections made to the JMS provider are authenticated, and access to JMS resources owned by the JMS provider are controlled by access authorizations. Also, all requests to create new connections to the JMS provider must provide a user ID and password for authentication. The user ID and password do not need to be provided by the application. If authentication is successful, then the JMS connection is created; if the authentication fails then the connection request is ended.

Standard J2C authentication is used for a request to create a new connection to the JMS provider. You can specify a Component-managed Authentication Alias and a Container-managed Authentication Alias for each JMS connection factory. The use of the associated J2C authentication data entries depends on the resource authentication (res-auth) setting, as follows:

- If your resource authentication (res-auth) is set to Application, set the alias in the Component-managed Authentication Alias. If the application that tries to create a connection to the JMS provider specifies a user ID and password, those values are used to authenticate the creation request. If the application does not specify a user ID and password, the values defined by the Component-managed Authentication Alias are used. If the connection factory is not configured with a Component-managed Authentication Alias, then you receive a runtime JMS exception when an attempt is made to connect to the JMS provider.
- If your res-auth is set to Container, set the Container-managed Authentication Alias. The values defined by the Container-managed Authentication Alias are used to authenticate the creation request. If you do not specify an alias, then you receive a runtime JMS exception when an attempt is made to connect to the JMS provider.

Restriction:

1. User IDs longer than 12 characters cannot be used for authentication with the embedded WebSphere JMS provider. For example, the default Windows NT user ID, **Administrator**, is not valid for use with embedded WebSphere messaging, because it contains 13 characters. Therefore, an authentication alias for a WebSphere JMS provider connection factory must specify a user ID no longer than 12 characters.
2. If you want to use a WebSphere MQ JMS Provider JMS connection when using Bindings transport mode, you set the property **Transport type**=BINDINGS on the WebSphere MQ Queue Connection Factory. You must also choose one of the following options:
   - To use security credentials, ensure that the user specified is the currently logged on user for the WebSphere Application Server process. If the user specified is not the current logged on user for the WebSphere Application Server process, then the WebSphere MQ JMS Bindings authentication throws the error `MQJMS2013 invalid security authentication supplied for MQQueueManager` error.
   - Do not specify security credentials. On the WebSphere MQ Connection Factory, ensure that both the **Component-managed Authentication Alias** and the **Container-managed Authentication Alias** properties are not set.

Authorization to access JMS resources owned by the embedded WebSphere JMS provider is controlled by authorization data in the config\integral-jms-authorisations.xml file. For information about editing this file, see Configuring security for the embedded WebSphere JMS provider.

Using WebSphere MQ functions from JMS applications

By default, JMS messages held on WebSphere MQ queues use an MQRFH2 header to hold some of the JMS message header information. Many legacy WebSphere MQ applications cannot process messages with these headers, and require their own characteristic headers, for example the MQCIH for CICS Bridge, or MQWIH for WebSphere MQ Workflow applications. The section ″Mapping JMS to a native WebSphere MQ application″ in the chapter ″JMS Messages″ of covers these special considerations.

Designing an enterprise application to use JMS

This topic describes things to consider when designing an enterprise application to use the JMS API directly for asynchronous messaging.

This topic describes things to consider when designing an enterprise application to use the JMS API directly for asynchronous messaging.

1. The application refers to JMS resources that are predefined, as administered objects, to WebSphere Application Server.

   Details of JMS resources that are used by enterprise applications are defined to WebSphere Application Server and bound into the JNDI namespace by the WebSphere administrative support. An enterprise application can retrieve these objects from the JNDI namespace and use them without needing to know anything about their implementation. This enables the underlying messaging architecture defined by the JMS resources to be changed without requiring changes to the enterprise application. When designing an enterprise application, you need to identify the details of the following types of JMS resources:

Point-to-Point	Publish/Subscribe
QueueConnectionFactory	TopicConnectionFactory
Queue	Topic

   A connection factory is used to create connections with the JMS provider for a specific JMS queue or topic destination. Each connection factory encapsulates the configuration parameters needed to create a connection to a JMS destination.

   For more information about the properties of these JMS resources, see Configuring JMS provider resources.
2. The application server pools connections and sessions with the JMS provider to improve performance. This is independent from any WebSphere MQ connection pooling. You need to configure the connection and session pool properties appropriately for your applications, otherwise you may not get the connection and session behavior that you want.
3. Applications can cache JMS connections, sessions, and producers or consumers. Due to the pooling mentioned above this may not give as much of a performance improvement as you might expect.

   You *must not* cache session handles in stateless session beans that operate in transactions started by a client of the bean. Caching handles in this way

causes the bean to be returned to the pool while the session is still involved in the transaction. Also, you should not cache non-durable subscribers due to the restriction mentioned above.

4. A non-durable subscriber can only be used in the same transactional context (for example, a global transaction or an unspecified transaction context) that existed when the subscriber was created. For more information about this context restriction, see The effect of transaction context on non-durable subscribers.
5. If you want to use authentication with embedded WebSphere messaging, you cannot have user IDs longer than 12 characters. For example, the default Windows NT user ID, **administrator**, is not valid for use with WebSphere internal messaging, because it contains 13 characters.
6. For messaging operations, you should write application programs that use only references to the interfaces defined in Sun's javax.jms package.

   JMS defines a generic view of a messaging that maps onto the underlying transport. An enterprise application that uses JMS, makes use of the following interfaces that are defined in Sun's javax.jms package:

   Connection
   Provides access to the underlying transport, and is used to create Sessions.

   Session
   Provides a context for producing and consuming messages, including the methods used to create MessageProducers and MessageConsumers.

   MessageProducer
   Used to send messages.

   MessageConsumer
   Used to receive messages.

   The generic JMS interfaces are subclassed into the following more specific versions for Point-to-Point and Publish/Subscribe behavior:

Point-to-Point	Publish/Subscribe
QueueConnection	TopicConnection
QueueSession,	TopicSession,
QueueSender	TopicSender
QueueReceiver	TopicReceiver

   The section "J2EE.6.7 Java Message Service (JMS) 1.0 Requirements" of the J2EE specification gives a list of methods that must not be called in Web and EJB containers:

```
javax.jms.Session method setMessageListener
javax.jms.Session method getMessageListener
javax.jms.Session method run
javax.jms.QueueConnection method createConnectionConsumer
javax.jms.TopicConnection method createConnectionConsumer
javax.jms.TopicConnection method createDurableConnectionConsumer
javax.jms.MessageConsumer method getMessageListener
javax.jms.MessageConsumer method setMessageListener
javax.jms.Connection setExceptionListener
javax.jms.Connection stop
javax.jms.Connection setClientID
```

   This method restriction is enforced in IBM WebSphere Application Server by throwing a javax.jms.IllegalStateException.
7. The following points, as defined in the EJB specification, apply to the use of flags on create*xxx*Session calls:
   - The transacted flag passed on create*xxx*Session is ignored inside a global transaction and all work is performed as part of the transaction. Outside of

a transaction the transacted flag is not used and, if set to true, the application should use session.commit() and session.rollback() to control the completion of the work. In an EJB2.0 module, if the transacted flag is set to true and outside of an XA transaction, then the session is involved in the WebSphere local transaction and the unresolved action attribute of the method applies to the JMS work.
- Clients cannot use using Message.acknowledge() to acknowledge messages. If a value of CLIENT_ACKNOWLEDGE is passed on the create*xxx*Session call, then messages are automatically acknowledged by the application server and Message.acknowledge() is not used.

8. Decide what message selectors are needed.

   You can use the JMS message selector mechanism to select a subset of the messages on a queue so that this subset is returned by a receive call. The selector can refer to fields in the JMS message header and fields in the message properties.

9. Acting on messages received.

   When a message is received, you can act on it as needed by the business logic of the application. Some general JMS actions are to check that the message is of the correct type and extract the content of the message. To extract the content from the body of the message, you need to cast from the generic Message class (which is the declared return type of the receive methods) to the more specific subclass, such as TextMessage. It is good practice always to test the message class before casting, so that unexpected errors can be handled gracefully.

   In this example, the instanceof operator is used to check that the message received is of the TextMessage type. The message content is then extracted by casting to the TextMessage subclass.

   ```
          if ( inMessage instanceof TextMessage )

   ...
              String replyString = ((TextMessage) inMessage).getText();
   ```

10. Using a listener to receive messages asynchronously.

    An alternative to making calls to QueueReceiver.receive() is to register a method that is called automatically when a suitable message is available; for example:

    ```
    ...
    MyClass listener =new MyClass();
    queueReceiver.setMessageListener(listener);
    //application continues with other application-specific behavior.
    ...
    ```

    When a message is available, it is retrieved by the onMessage() method on the listener object.

    ```
    import javax.jms.*;
    public class MyClass implements MessageListener
    {
    public void onMessage(Message message)
    {
    System.out.println("message is "+message);
    //application specific processing here
    ...

    }
    }
    ```

    Note: A MessageListener can only be used in the client container. (The J2EE specification forbids the use of the JMS MessageListener mechanism for the asynchronous receipt of messages in the EJB and Web containers.)

For asynchronous message delivery, the application code cannot catch exceptions raised by failures to receive messages. This is because the application code does not make explicit calls to receive() methods. To cope with this situation, you can register an ExceptionListener, which is an instance of a class that implements the onException()method. When an error occurs, this method is called with the JMSException passed as its only parameter.

For more details about using listeners to receive messages asynchronously, see the Java Message Service Documentation.

Note: An alternative to developing your own JMS listener class, you can use a message-driven bean, as described in Implementing WebSphere enterprise applications that use message-driven beans.

11. Warning when receiving messages within a server-side application component.

Take care when performing a JMS receive() from a server-side application component if that receive() invocation is waiting on a message produced by another application component that is deployed in the same server. Such a JMS receive() is synchronous, so blocks until the response message is received.

This type of application design can lead to the consumer/producer problem where the entire set of work threads can be exhausted by the receiving component, which has been blocked waiting for responses, leaving no available worker thread for which to dispatch the application component that would generate the response JMS message.

To illustrate this problem, picture a servlet and a message-driven bean deployed in the same server. When this servlet dispatches a request it sends a message to a queue which is serviced by the message-driven bean (that is, messages produced by the servlet are consumed by the message-driven bean's onMessage() method). The servlet subsequently issues a receive(), waiting for a reply on a temporary ReplyTo queue. The message-driven bean's onMessage() method performs a database query and sends back a reply to the servlet on the temporary queue. If a large number of servlet requests occur at once (relative to the number of server worker threads), then it is likely that all available server worker threads will be used to dispatch a servlet request, send a message, and wait for a reply. The application server enters a deadly-embrace condition whereby no threads remain to process any of the message-driven beans that are now pending. Since the servlets are waiting in blocking recieves, the server hangs, likely leading to application failure.

Possible solutions are:

a. Ensure that the number of worker threads (# of threads per server region * # of server regions per server) exceeds the number of concurrent dispatches of the application component doing the receive() so that there is always a worker thread available to dispatch the message producing component.
b. Use an application topology that places the receiver application component in a separate server than the producer application component. While worker thread usage can still need to be carefully considered under such a deployment scenario, this separation ensures that there are always be threads that cannot be blocked by the message receiving component. There can be other interactions to consider, such as an application server that has multiple applications installed.
c. Refactor your application to do the message receives from a client component, which will not compete with the producer component for worker threads. Furthermore, the client component can do asynchronous (non-blocking) receives, which are prohibited from J2EE servers. So, for

example, the example application above could be refactored to have a client sending messages to a queue and then waiting for a response from the MDB.

The effect of transaction context on non-durable subscribers

A non-durable subscriber can only be used in the same transactional context (for example, a global transaction or an unspecified transaction context) that existed when the subscriber was created. A non-durable subscriber is invalidated whenever a sharing boundary (in general, a local or global transaction boundary) is crossed, resulting in a javax.jms.IllegalStateException with message text `Non-durable subscriber invalidated on transaction boundary`.

For example, in the following scenario the non-durable subscriber is invalidated at the begin user transaction. This is because the local transaction context in which the subscriber was created ends when the user transaction begins:

```
...
create subscriber
...
begin user transaction -
...
complete user transaction -
...
use subscriber
...
```

If you want to cache a subscriber (to wait to receive messages that arrived since it was created), then use a durable subscriber (for which this restriction does not apply). Do not cache non-durable subscribers.

Developing a J2EE application to use JMS

Use this task to develop a J2EE application to use the JMS API directly for asynchronous messaging.

This topic gives an overview of the steps needed to develop a J2EE application (servlet or enterprise bean) to use the JMS API directly for asynchronous messaging.

This topic only describes the JMS-related considerations; it does not describe general J2EE application programming, which you should already be familiar with. For detailed information about these steps, and for examples of developing a J2EE application to use JMS, see the Java Message Service Documentation and the WebSphere MQ *Using Java* book, SC34-5456.

Details of JMS resources that are used by J2EE applications are defined to WebSphere Application Server and bound into the JNDI namespace by the WebSphere administrative support.

To use JMS, a J2EE application completes the following general steps:

1. Import JMS packages. A J2EE application that uses JMS starts with a number of import statements for JMS, which should include at least the following:

   ```
   import javax.jms.*;       //JMS interfaces
   import javax.naming.*;    //Used for JNDI lookup of administered objects
   ```

2. Get an initial context.

   ```
      try     {
           ctx = new InitialContext(env);
   ...
   ```

3. Retrieve administered objects from the JNDI namespace. The InitialContext.lookup() method is used to retrieve administered objects (a queue connection factory and the queue destinations); for example, to receive a message from a queue

```
qcf = (QueueConnectionFactory)ctx.lookup( qcfName );
...
inQueue = (Queue)ctx.lookup( qnameIn );
...
```

4. Create a connection to the messaging service provider. The connection provides access to the underlying transport, and is used to create sessions. The createQueueConnection() method on the factory object is used to create the connection.

```
connection = qcf.createQueueConnection();
```

   The JMS specification defines that connections should be created in the stopped state. Until the connection starts, MessageConsumers that are associated with the connection cannot receive any messages. To start the connection, issue the following command:

```
connection.start();
```

5. Create a session, for sending or receiving messages. The session provides a context for producing and consuming messages, including the methods used to create MessageProducers and MessageConsumers. The createQueueSession method is used on the connection to obtain a session. The method takes two parameters:
   - A boolean that determines whether or not the session is transacted.
   - A parameter that determines the acknowledge mode.

```
boolean transacted = false;
session = connection.createQueueSession( transacted,
                                         Session.AUTO_ACKNOWLEDGE);
```

   In this example, the session is not transacted, and it should automatically acknowledge received messages. With these settings, a message is backed out only after a system error or if the application terminates unexpectedly.

   The following points, as defined in the EJB specification, apply to these flags:
   - The transacted flag passed on createQueueSession is ignored inside a global transaction and all work is performed as part of the transaction. Outside of a transaction the transacted flag is not used and, if set to true, the application should use session.commit() and session.rollback() to control the completion of the work. In an EJB2.0 module, if the transacted flag is set to true and outside of an XA transaction, then the session is involved in the WebSphere local transaction and the unresolved action attribute of the method applies to the JMS work.
   - Clients cannot use using Message.acknowledge() to acknowledge messages. If a value of CLIENT_ACKNOWLEDGE is passed on the create*xxx*Session call, then messages are automatically acknowledged by the application server and Message.acknowledge() is not used.

6. Send a message.
   a. Create MessageProducers to create messages. For point-to-point messaging the MessageProducer is a QueueSender that is created by passing an output queue object (retrieved earlier) into the createSender method on the session. A QueueSender is normally created for a specific queue, so that all messages sent using that sender are sent to the same destination.

```
QueueSender queueSender = session.createSender(inQueue);
```

   b. Create the message. Use the session to create an empty message and add the data passed.

JMS provides several message types, each of which embodies some knowledge of its content. To avoid referencing the vendor-specific class names for the message types, methods are provided on the Session object for message creation.

In this example, a text message is created from the outString property:

```
TextMessage outMessage = session.createTextMessage(outString);
```

c. Send the message.

To send the message, the message is passed to the send method on the QueueSender:

```
queueSender.send(outMessage);
```

7. Receive replies.

a. Create a correlation ID to link the message sent with any replies. In this example, the client receives reply messages that are related to the message that it has sent, by using a provider-specific message ID in a JMSCorrelationID.

```
messageID = outMessage.getJMSMessageID();
```

The correlation ID is then used in a message selector, to select only messages that have that ID:

```
String selector = "JMSCorrelationID = '"+messageID+"'";
```

b. Create a MessageReceiver to receive messages. For point-to-point the MessageReceiver is a QueueReceiver that is created by passing an input queue object (retrieved earlier) and the message selector into the createReceiver method on the session.

```
QueueReceiver queueReceiver = session.createReceiver(outQueue, selector);
```

c. Retrieve the reply message. To retrieve a reply message, the receive method on the QueueReceiver is used:

```
Message inMessage = queueReceiver.receive(2000);
```

The parameter in the receive call is a timeout in milliseconds. This parameter defines how long the method should wait if there is no message available immediately. If you omit this parameter, the call blocks indefinitely. If you do not want any delay, use the receiveNoWait()method. In this example, the receive call returns when the message arrives, or after 2000ms, whichever is sooner.

d. Act on the message received. When a message is received, you can act on it as needed by the business logic of the client. Some general JMS actions are to check that the message is of the correct type and extract the content of the message. To extract the content from the body of the message, it is necessary to cast from the generic Message class (which is the declared return type of the receive methods) to the more specific subclass, such as TextMessage. It is good practice always to test the message class before casting, so that unexpected errors can be handled gracefully.

In this example, the instanceof operator is used to check that the message received is of the TextMessage type. The message content is then extracted by casting ti the TextMessage subclass.

```
if ( inMessage instanceof TextMessage )

...
        String replyString = ((TextMessage) inMessage).getText();
```

8. Closing down. If the application needs to create many short-lived JMS objects at the Session level or lower, it is important to close all the JMS resources

used. To do this, you call the close() method on the various classes (QueueConnection, QueueSession, QueueSender, and QueueReceiver) when the resources are no longer required.

```
       queueReceiver.close();
...
       queueSender.close();
...
       session.close();
       session = null;
...
       connection.close();
       connection = null;
```

9. Publishing and subscribing messages. To use JMS Publish/Subscribe support instead of point-to-point messaging, the general actions are the same; for example, to create a session and connection. The exceptions are that topic resources are used instead of queue resources (such as TopicPublisher instead of QueueSender), as shown in the following example to publish a message:

```
// Creating a TopicPublisher
       TopicPublisher pub = session.createPublisher(topic);
...
       pub.publish(outMessage);
...
       // Closing TopicPublisher
       pub.close();
```

10. Handling errors Any JMS runtime errors are reported by exceptions. The majority of methods in JMS throw JMSExceptions to indicate errors. It is good programming practice to catch these exceptions and display them on a suitable output.

    Unlike normal Java exceptions, a JMSException can contain another exception embedded in it. The implementation of JMSException does not include the embedded exception in the output of its toString()method. Therefore, you need to check explicitly for an embedded exception and print it out, as shown in the following example:

```
catch (JMSException je)
{
   System.out.println("JMS failed with "+je);
   Exception le = je.getLinkedException();
   if (le != null)
   {
       System.out.println("linked exception "+le);
   }
}
```

After you have packaged your application, you can next deploy the application into WebSphere Application Server, as described in Deploying a J2EE application to use JMS.

Developing a JMS client

Use this task to develop a JMS client application to use messages to communicate with enterprise applications.

This topic gives an overview of the steps needed to develop a JMS client application, based on a sample client provided with WebSphere Application Server. This topic only describes the JMS-related considerations; it does not describe general client programming, which you should already be familiar with. For detailed information about these steps, and for examples of developing JMS clients, see the Java Message Service Documentation and the WebSphere MQ *Using Java* book, SC34-5456.

A JMS client assumes that the JMS resources (such as a queue connection factory and queue destination) already exist. A client application can use JMS resources administered by the application server or administered by the client container regardless of whether the client application is running on the same machine as the server or remotely.

- If you want your client application to use server-administered JMS objects, configure the client application to use those resources as Resource Environment References.
- If you want your client application to use client container-administered JMS resources, then configure those resources as Resource References.

For more information about developing client applications and configuring JMS resources for them, see the Information Center topic ″Developing J2EE application client code″ and related tasks.

To use JMS, a typical JMS client program completes the following general steps:

1. Import JMS packages. An enterprise application that uses JMS starts with a number of import statements for JMS; for example:

```
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.rmi.PortableRemoteObject;
import javax.jms.*;
```

2. Get an initial context.

```
   try     {
        ctx = new InitialContext(env);
...
```

3. Define the parameters that the client wants to use; for example, to identify the queue connection factory and to assemble a message to be sent.

```
public class JMSppSampleClient
{
  public static void main(String[] args)
  throws JMSException, Exception

  {
    String  messageID                = null;
    String  outString                = null;
    String  qcfName                  = "java:comp/env/jms/ConnectionFactory";
    String  qnameIn                  = "java:comp/env/jms/Q1";
    String  qnameOut                 = "java:comp/env/jms/Q2";
    boolean verbose                  = false;

    QueueSession           session    = null;
    QueueConnection        connection = null;
    Context                ctx        = null;

    QueueConnectionFactory qcf        = null;
    Queue                  inQueue    = null;
    Queue                  outQueue   = null;

...
```

4. Retrieve administered objects from the JNDI namespace. The InitialContext.lookup() method is used to retrieve administered objects (a queue connection factory and the queue destinations):

```
       qcf = (QueueConnectionFactory)ctx.lookup( qcfName );
...
       inQueue = (Queue)ctx.lookup( qnameIn );
       outQueue = (Queue)ctx.lookup( qnameOut );
...
```

5. Create a connection to the messaging service provider. The connection provides access to the underlying transport, and is used to create sessions. The createQueueConnection() method on the factory object is used to create the connection.

   ```
   connection = qcf.createQueueConnection();
   ```

   The JMS specification defines that connections should be created in the stopped state. Until the connection starts, MessageConsumers that are associated with the connection cannot receive any messages. To start the connection, issue the following command:

   ```
   connection.start();
   ```

6. Create a session, for sending and receiving messages. The session provides a context for producing and consuming messages, including the methods used to create MessageProducers and MessageConsumers. The createQueueSession method is used on the connection to obtain a session. The method takes two parameters:
   - A boolean that determines whether or not the session is transacted.
   - A parameter that determines the acknowledge mode.

   ```
   boolean transacted = false;
   session = connection.createQueueSession( transacted,
                                            Session.AUTO_ACKNOWLEDGE);
   ```

   In this example, the session is not transacted, and it should automatically acknowledge received messages. With these settings, a message is backed out only after a system error or if the client application terminates unexpectedly.

7. Send the message.
   a. Create MessageProducers to create messages. For point-to-point the MessageProducer is a QueueSender that is created by passing an output queue object (retrieved earlier) into the createSender method on the session. A QueueSender is normally created for a specific queue, so that all messages sent using that sender are sent to the same destination.

      ```
      QueueSender queueSender = session.createSender(inQueue);
      ```

   b. Create the message. Use the session to create an empty message and add the data passed.

      JMS provides several message types, each of which embodies some knowledge of its content. To avoid referencing the vendor-specific class names for the message types, methods are provided on the Session object for message creation.

      In this example, a text message is created from the outString property, which could be provided as an input parameter on invocation of the client program or constructed in some other way:

      ```
      TextMessage outMessage = session.createTextMessage(outString);
      ```

   c. Send the message.

      To send the message, the message is passed to the send method on the QueueSender:

      ```
      queueSender.send(outMessage);
      ```

8. Receive replies.
   a. Create a correlation ID to link the message sent with any replies. In this example, the client receives reply messages that are related to the message that it has sent, by using a provider-specific message ID in a JMSCorrelationID.

      ```
      messageID = outMessage.getJMSMessageID();
      ```

      The correlation ID is then used in a message selector, to select only messages that have that ID:

```
String selector = "JMSCorrelationID = '"+messageID+"'";
```

b. Create a MessageReceiver to receive messages. For point-to-point the MessageReceiver is a QueueReceiver that is created by passing an input queue object (retrieved earlier) and the message selector into the createReceiver method on the session.

```
QueueReceiver queueReceiver = session.createReceiver(outQueue, selector);
```

c. Retrieve the reply message. To retrieve a reply message, the receive method on the QueueReceiver is used:

```
Message inMessage = queueReceiver.receive(2000);
```

The parameter in the receive call is a timeout in milliseconds. This parameter defines how long the method should wait if there is no message available immediately. If you omit this parameter, the call blocks indefinitely. If you do not want any delay, use the receiveNoWait()method. In this example, the receive call returns when the message arrives, or after 2000ms, whichever is sooner.

d. Act on the message received. When a message is received, you can act on it as needed by the business logic of the client. Some general JMS actions are to check that the message is of the correct type and extract the content of the message. To extract the content from the body of the message, you need to cast from the generic Message class (which is the declared return type of the receive methods) to the more specific subclass, such as TextMessage. It is good practice always to test the message class before casting, so that unexpected errors can be handled gracefully.

In this example, the instanceof operator is used to check that the message received is of the TextMessage type. The message content is then extracted by casting to the TextMessage subclass.

```
if ( inMessage instanceof TextMessage )

...
        String replyString = ((TextMessage) inMessage).getText();
```

9. Closing down. If the application needs to create many short-lived JMS objects at the Session level or lower, it is important to close all the JMS resources used. To do this, you call the close() method on the various classes (QueueConnection, QueueSession, QueueSender, and QueueReceiver) when the resources are no longer required.

```
queueReceiver.close();
...
queueSender.close();
...
session.close();
session = null;
...
connection.close();
connection = null;
```

10. Publishing and subscribing messages. To use publish/subscribe support instead of point-to-point messaging, the general client actions are the same; for example, to create a session and connection. The exceptions are that topic resources are used instead of queue resources (such as TopicPublisher instead of QueueSender), as shown in the following example to publish a message:

```
// Creating a TopicPublisher
TopicPublisher pub = session.createPublisher(topic);
...
pub.publish(outMessage);
...
// Closing TopicPublisher
pub.close();
```

11. Handling errors Any JMS runtime errors are reported by exceptions. The majority of methods in JMS throw JMSExceptions to indicate errors. It is good programming practice to catch these exceptions and display them on a suitable output.

    Unlike normal Java exceptions, a JMSException can contain another exception embedded in it. The implementation of JMSException does not include the embedded exception in the output of its toString()method. Therefore, you need to check explicitly for an embedded exception and print it out, as shown in the following example:

```
catch (JMSException je)
{
   System.out.println("JMS failed with "+je);
   Exception le = je.getLinkedException();
   if (le != null)
   {
       System.out.println("linked exception "+le);
   }
}
```

Deploying a J2EE application to use JMS

This topic describes how to deploy a J2EE application to use JMS.

This task description assumes that you have an .EAR file, which contains an application enterprise bean with code for JMS, that can be deployed in WebSphere Application Server.

To deploy a J2EE application to use JMS, complete the following steps:

1. 5.1+ Configure the deployment attributes for the application, as described in the Information Center topic ″Assembling applications with the Assembly Toolkit.″
2. Use the WebSphere administrative console to install the application.

   This stage is a standard WebSphere Application Server task, as described in the Information Center topic ″Installing applications.″

Tuning Java messaging service

You can tune Java messaging service (JMS) run-time components, resources, and the embedded messaging server through the administrative console with the following parameters:

- **Configure the message listening service**.
  - Thread pool
  - Custom properties Application Server Facilities and Non-Application Server Facilities
  - Listener port Maximum sessions
  - Listener port Maximum messages
- **Configure JMS resources**.
  - XA enabled
  - Connection pool size
  - WebSphere MQ queue connection factory Transport type
  - WebSphere MQ topic connection factory Transport type
  - Transaction log directory
  - **WebSphere MQ**

- **Description:** Includes the WebSphere MQ folder where WebSphere MQ is installed and indirectly where its data files reside since they are always under the directory *WebSphereMQHome/data*. Use this parameter during installation when extra disks are available.
- **How to view or set:** To view, search your system for a folder named WebSphere MQ. Set during installation.
- **Default value:** *c:\Program Files\IBM\WebSphere MQ*
- **Recommended value:** Locate this directory away from any other disk input or output, on the fastest disk available. Redundant Array of Inexpensive Disks (RAID) is recommended.

– **WebSphere MQ**
- **Description:** Includes the WebSphere MQ folder where WebSphere MQ is installed and indirectly where its data files reside since they are always under the directory *WebSphereMQHome/data*. Use this parameter during installation when extra disks are available.
- **How to view or set:** To view, search your system for a folder named WebSphere MQ. Set during installation.
- **Default value:** *c:\Program Files\IBM\WebSphere MQ*
- **Recommended value:** Locate this directory away from any other disk input or output, on the fastest disk available. Redundant Array of Inexpensive Disks (RAID) is recommended.

– **Log buffer pages**
- **Description:** Controls the size of the queue manager buffer to log file writes in number of pages. Use this parameter when high volumes of messages are being sent through a server.
- **How to view or set:** In the registry, navigate to **HKEY_LOCAL_MACHINE** > **SOFTWARE** > **IBM** > **MQSeries** > **CurrentVision** > **Configuration** > **QueueManager** > *QM_name* > **Log** > **LogBufferPages**. For embedded JMS, edit the `createmq.properties` setting LogFilePages in the *install_root*\properties directory to the desired value. Run the **deletemq** and **createmq** commands to delete or recreate the queue manager. For external JMS, set LogFilePages to the desired value. Do this before creating the queue manager in the registry. Run the **amqmdain regsec** command to secure the registry value and create the queue manager.
- **Default value:** 0
- **Recommended value:** There is no performance penalty for setting this value too high. Therefore, set the value to its maximum of 512 pages.

– **Log primary files**
- **Description:** Controls the number of primary or permanent log files for the queue manager. Use this parameter when high volumes of messages are being sent through a server.
- **How to view or set:** In the registry, navigate to **HKEY_LOCAL_MACHINE** > **SOFTWARE** > **IBM** > **MQSeries** > **CurrentVision** > **Configuration** > **QueueManager** > *QM_name* > **Log** > **LogPrimaryFiles**. For embedded JMS, edit the `createmq.properties` setting LogFilePages in the *install_root*\properties directory to the desired value. Run the **deletemq** and **createmq** commands to delete or recreate the queue manager. For external JMS, set LogPrimaryFiles to the desired value. Do this before creating the queue manager in the registry. Run the **amqmdain regsec** command to secure the registry value and create the queue manager.
- **Default value:** External: 3, embedded: 0
- **Recommended value:** There is no performance penalty for setting this value too high. Set the value to its maximum of 63 pages.

– **Log secondary files**

- **Description:** Controls the number of secondary log files for the queue manager. Secondary files are files created when the primary files are not enough and deleted when they are no longer needed.
- **How to view or set:** In the registry, navigate to **HKEY_LOCAL_MACHINE** > **SOFTWARE** > **IBM** > **MQSeries** > **CurrentVision** > **Configuration** > **QueueManager** > *QM_name* > **Log** > **LogSecondaryFiles**. For embedded JMS, edit the `createmq.properties` setting LogSecondaryFiles in the *install_root*\properties directory to the desired value. Run the **deletemq** and **createmq** commands to delete or recreate the queue manager. For external JMS, set LogPrimaryFiles to the desired value. Do this before creating the queue manager in the registry. Run the **amqmdain regsec** command to secure the registry value and create the queue manager.
- **Default value:** External: 2, embedded: 60
- **Recommended value:** There is a limit of 63 total files from the primary and secondary combined and because secondary logs are slower, setting this 0 is ideal for performance.
- **Log default path**
  - **Description:** Controls the location of the queue manager log files. Use this parameter when high volumes of messages are sent through a server.
  - **How to view or set:** In the registry, navigate to **HKEY_LOCAL_MACHINE** > **SOFTWARE** > **IBM** > **MQSeries** > **CurrentVision** > **Configuration** > **QueueManager** > *QM_name* > **Log** > **LogDefaultPath**. For embedded JMS, edit the `createmq.properties` setting LogPath in the *install_root*\properties directory to the desired value. Run the **deletemq** and **createmq** commands to delete or recreate the queue manager. For external JMS, set LogDefaultPath to the desired value. Do this before creating the queue manager in the registry. Run the **amqmdain regsec** command to secure the registry value and create the queue manager.
  - **Default value:** *WebSphereMQHome*\log
  - **Recommended value:** It is ideal to have a disk dedicated to this task because WebSphere MQ tries to keep the head of the disk positioned at the place in the file where it needs to write next. A fast RAID volume is best.
- **Default queue buffer size**
  - **Description:** Controls the size in bytes of an in-memory buffer for nonpersistent queues. Use this parameter when large message sizes are used, or large bursts of messages cause the queue to back up. If the queue backs up past this buffer, messages are flushed out to the disk.
  - **How to view or set:** In the registry, navigate to **HKEY_LOCAL_MACHINE** > **SOFTWARE** > **IBM** > **MQSeries** > **CurrentVersion** > **Configuration** > **QueueManager** > *QM_name* > **TuningParameters** > **DefaultQBufferSize**. Embedded JMS is not currently supported; you need MQ5.3 CSD2 and an accompanying Version 5.0 fix. For external JMS, set DefaultQBufferSize to the desired value. Do this before creating the queue manager. Run the **amqmdain regsec** command to secure the registry value and create the queue manager.
  - **Default value:** 64K (registry key does not exist)
  - **Recommended value:** Set this parameter to accommodate the typical number of messages sitting on the queue at any given time. This should be numberOfMessages*(500+messageSizeInBytes). The maximum value is 100MB, but typically 1MB is enough.
- **Default persistent queue buffer size**

- **Description:** Controls the size in bytes of an in-memory buffer for nonpersistent queues. Use this parameter whenever memory is available.
- **How to view or set:** In the registry, navigate to **HKEY_LOCAL_MACHINE** > **SOFTWARE** > **IBM** > **MQSeries** > **CurrentVersion** > **Configuration** > **QueueManager** > *QM_name* > **TuningParameters** > **DefaultPQBufferSize**. Embedded JMS is not currently supported; you need MQ5.3 CSD2 and an accompanying Version 5.0 fix. For external JMS, set DefaultPQBufferSize to the desired value. Create the queue manager. This is a permanent queue setting. To make a change, delete and recreate the queue.
- **Default value:** 0 (registry key does not exist)
- **Recommended value:** Set this parameter to accommodate the number of typical concurrently processed messages, plus a little more for read-ahead capabilities. Do this by calculating numOfCocurrentMessages*(msgSizeInBytes+500)*2. Typically, 1MB is enough.

- **Maximum channels**
  - **Description:** Controls the allowable number of concurrent CLIENT transport clients. Use this parameter when large numbers of clients are being used.
  - **How to view or set:** In the registry, navigate to **HKEY_LOCAL_MACHINE** > **SOFTWARE** > **IBM** > **MQSeries** > **CurrentVersion** > **Configuration** > **QueueManager** > *QM_name* > **Channels** > **MaxChannels**. Embedded JMS is not currently supported; you need MQ5.3 CSD2 and an accompanying Version 5.0 fix. For external JMS, set MaxChannels to the desired value. Restart the queue manager.
  - **Default value:** External: 100 (registry key does not exist), embedded: 1000
  - **Recommended value:** Set this parameter high enough to contain the maximum number of concurrent JMS clients.
- **Channel application bind type**
  - **Description:** Controls if the channel application is an MQ FASTPATH application. Use this parameter at all times.
  - **How to view or set:** In the registry, navigate to **HKEY_LOCAL_MACHINE** > **SOFTWARE** > **IBM** > **MQSeries** > **CurrentVersion** > **Configuration** > **QueueManager** > *QM_name* > **Channels** > **MQIBindType**. Embedded JMS is not currently supported; you need MQ5.3 CSD2 and an accompanying Version 5.0 fix. For external JMS, set MQIBindType to the desired value. Restart the queue manager.
  - **Default value:** Not FASTPATH (registry key does not exist)
  - **Recommended value:** FASTPATH

- **Configure resources for the embedded WebSphere JMS provider.**
  - **Number of threads**
    - **Description:** With the embedded JMS publications and subscriptions broker, this value is the number of threads used to match publications to subscribers. Use this parameter when concurrent publications and subscriptions exist that would exceed the capacity of the default value.
    - **How to view or set:**
      1. Open the administrative console.
      2. Click **Servers** > **Application Servers** > *server_name*.
      3. Click the *Server Component* > **JMS servers**.
      4. Click **Apply** or **OK**.
      5. Click **Save**.
      6. Stop and restart the application server.
    - **Default value:** 1

- **Recommended value:** Set this value a little higher than the number of concurrent message publishers. If large numbers of subscribers exist, increasing this value can also provide some benefit.

Troubleshooting WebSphere Messaging

Use this overview task to help resolve a problem that you think is related to the WebSphere Messaging.

To identify and resolve problems that you think are related to WebSphere Messaging, you can use the standard WebSphere Application Server troubleshooting facilities. If you encounter a problem that you think might be related to WebSphere Messaging, complete the following stages. Some problems and their troubleshooting are specific to whether you are using the embedded WebSphere Messaging or WebSphere MQ as the JMS provider.

1. Check for common problems related to WebSphere Messaging. For example, check that the JMS server has been started, that you have added queue names to the list on the JMS server page of the Administrative Console, and that you have successfully installed the WebSphere Messaging function.

   For tips about solving problems related to the WebSphere Messaging, see Tips for troubleshooting WebSphere Messaging. If those tips do not help you fix the problem, complete the following general stages.
2. Check the Release Notes for specific problems and workarounds The section *Possible Problems and Suggested Fixes* of the Release Notes, available from the WebSphere Application Server library web site, is updated regularly to contain information about known defects and their workarounds. Check the latest version of the Release Notes for any information about your problem. If the Release Notes do not contain any information about your problem, you can also search the Technotes database on the WebSphere Application Server web site.
3. Check for WebSphere Messaging error messages.

   Check in the SYSPRINT or SYSOUT log for error messages with the prefixes MSGS and WMSG.

   The associated message reference information provides an explanation and any user actions to resolve the problem.
4. Check for more informational and error messages that might provide a clue to a related problem. If the JMS server is running, but you have problems accessing JMS resources, check for more error messages and extra details about the problem.
5. Check your JMS resource configurations If the WebSphere Messaging functions seem to be running properly (the JMS server is running without problems), check that the JMS resources have been configured correctly. For example, check that queue destinations and their connection factories have corresponding JNDI names, that the JNDI names match those configured for the messaging applications, and that the connection factories are configured onto nodes that can provide the JMS resources.
6. Get a detailed exception dump for WebSphere Messaging. If the information obtained in the preceding steps is still inconclusive, you can enable the application server debug trace for the “Messaging” group to provide a detailed exception dump.

Tips for troubleshooting WebSphere Messaging

This topic provides a set of tips to help you troubleshoot problems with WebSphere Messaging.

- The JMS server does not start by default when WebSphere Messaging is installed
- The JMS server tries to start, but fails
- An MDB listener fails to start
- Failure to create a queue connection
- Embedded WebSphere Messaging failed to install on top of an existing WebSphere MQ product
- Problems running JMS applications with security enabled
- Queue manager fails to stop on Redhat Linux
- Application server fails to start in zh_TW.EUC locale on Solaris

The JMS server does not start when starting the WebSphere administrative server

During installation, the system PATH setting is updated to include the WebSphere Messaging directories. If you try to start the WebSphere administrative server from a session that does not use the updated system PATH, the JMS Server fails to start properly. To resolve this after installing WebSphere Application Server, stop then restart your host or open a new session that uses the updated system PATH.

Check that the JMS server has started before trying to use WebSphere Messaging. For messages that indicate the JMS server has started successfully, see the following tip The JMS server tries to start, but fails.

For more information about managing JMS servers, see Administering WebSphere JMS support.

The JMS server tries to start, but fails

To see if the JMS Server has started, check for error messages in the SYSPRINT or SYSOUT log. If the JMS server started successfully, you should see the following messages:

```
MSGS0050I: Starting the Queue Manager
MSGS0051I: Queue Manager open for business
MSGS0052I: Starting the Broker
MSGS0053I: Broker open for business
```

If the JMS server tries to start, but fails, you should see messages indicating that the queue manager could not start, like the following:

```
MSGS0153E: The Queue Manager process could not be started - error:
com.ibm.ws.process.exception.InvalidExecutableException:
The system cannot find the file specified.
002: No such file or directory
```

If you see a message like the following, check to see that required WebSphere MQ messages are not being suppressed by the message processing facility (MPF):

```
BBOO0222I WSVR0002I: Server CONTROL PROCESS xyzabc open for e-business,
problems occurred during startup
```

Before starting a JMS server, ensure that the following WebSphere MQ messages are not being suppressed by the message processing facility:

```
CSQV086E CSQY022I CSQY003I CSQX022I CSQM132I CSQ9022I CSQX017I CSQ3104I CSQ3106E
```

An MDB listener fails to start

If an MDB listener fails to start, you should see the following message:

```
WMSG0019E: Unable to start MDB Listener {0}, JMSDestination {1} : {2}
```

To troubleshoot the cause of an MDB listener not starting, check the following factors:

- Check that the administrative resources have been configured correctly; for example, use the administrative console to check the listener port properties: Destination JNDI name and Connection factory JNDI name. Check that other properties of the listener port, destination, and connection factory are correct.
- Check that the queue exists and has been added to the JMS server.
- Check that the queue manager and JMS server have started.
- Check that the Remote Queue Manager Listener has started.
- If security is enabled, check that a component-managed authetication alias has been specified on the queue connection factory or topic connection factory used by the message-driven bean. This is not required if security is not enabled.
- Check that the user ID used to start the MDB listener has 12 characters or less.

Failure to create a queue connection

If WebSphere Application Server fails to create a queue connection, the SYSPRINT and SYSOUT log contains error messages like the following:

```
J2CA0046E: Method addNewConnection caught javax.resource.spi.ResourceAdapterInternalException:
createQueueConnection failed
```

Check that the JMS server is running (including that the Internal WebSphere Messaging or WebSphere MQ JMS provider was installed correctly) as described in preceding tips.

Note: In a Network Deployment or Enterprise multi-node cell, the JMS server used by a messaging application can be on a different node to the application. Either check that all JMS servers in the cell have started, or use the Administrative console to determine which JMS server the application is trying to connect to (look at the Node property of the appropriate connection factory), then check that the JMS server has started.

Embedded WebSphere Messaging failed to install on top of an existing WebSphere MQ product

When preparing to install WebSphere Application Server on a host that already has WebSphere MQ installed, you should ensure that WebSphere MQ is at a prerequisite level, as described in the Release Notes and Supported hardware, software, and APIs page of the WebSphere Application Server library web site. You can also check the WebSphere MQ Support service summary Web page at http://www-3.ibm.com/software/ts/mqseries/support/summary/.

If you have a problem installing the embedded WebSphere Messaging function, first check the mq_install.log. Failures during the WebSphere Messaging prereq stage usually indicate a shortage of space. Failures after this stage are usually due to a conflict between messaging components already installed on the machine and the levels required to support the J2EE 1.3 specification.

If the embedded WebSphere Messaging function installed successfully, you should see messages like the following in mq_install.log:

```
...
date time MsiInstallProduct() returning ERROR_SUCCESS (0)
date time ====================== Exiting Publish And Subscribe Install ======================
date time <<Function Successful>> return code WASM_ERROR_SUCCESS (0)
date time ====================== End of WebSphere Messaging Install Log ======================
```

You can also check the createMQ.log for any messages that indicate a configuration problem with the installation of WebSphere Messaging.

Problems running JMS applications with security enabled

When trying to run a JMS application with security enabled, you can encounter problems indicated by one of the error messages:

MSGS0508E: The JMS Server security service was unable to authenticate userid:
This indicates that the JMS connection has not provided the WebSphere JMS server with any security credentials.

WMSG0019E: Unable to start MDB Listener PSSampleMDB, JMSDestination Sample/JMS/listen : javax.jms.JMSSecurityException:
This indicates that the security credentials supplied are not valid.

In both cases the problem can be removed by doing one of the following:
- If the authentication mechanism is set to `Application`, then the application needs to supply valid credentials.
- If the authentication mechanism is set to `Container`, then you need to configure the JMS ConnectionFactory with a container-managed Authentication Alias and ensure that the associated username and password are valid.

MQJMS2013 invalid security authentication supplied for MQQueueManager
If using a WebSphere MQ JMS Provider JMS connection when using Bindings transport mode, and the user specified is not the current logged on user for the WebSphere Application Server process, then the WebSphere MQ JMS Bindings authentication throws the error `MQJMS2013 invalid security authentication supplied for MQQueueManager`.

If you want to use a WebSphere MQ JMS Provider JMS connection when using Bindings transport mode, you set the property **Transport type**=BINDINGS on the WebSphere MQ Queue Connection Factory. You must also choose one of the following options:
- To use security credentials, ensure that the user specified is the currently logged on user for the WebSphere Application Server process.
- Do not specify security credentials. On the WebSphere MQ Connection Factory, ensure that both the **Component-managed Authentication Alias** and the **Container-managed Authentication Alias** properties are not set.

For more information about messaging security, see Asynchronous messaging - security considerations.

Queue manager fails to stop on Redhat Linux

When trying to stop an application server on Redhat Linux, the queue manager can hang with a Java core dump, and the last message in the SystemOut.log file is `Stopping Queue manager...`.

This is caused by a known RedHat problem (https://bugzilla.linux.ibm.com/show_bug.cgi?id=2336), that was introduced in libstdc++-2.96-116.7.2 and beyond.

The workaround is to go back to the libstdc++-2.96-108.1 level.

Application server fails to start in zh_TW.EUC locale on Solaris

If you have set the locale to zh_TW.EUC on Solaris, and are using the WebSphere embedded JMS provider or WebSphere MQ as the JMS provider, you can encounter problems that stop application servers starting up.

If you intend using the WebSphere embedded JMS provider or WebSphere MQ as the JMS provider on Solaris, do not set the LANG and LC_ALL variables to zh_TW.EUC (Traditional Chinese locale) to avoid problems when starting application servers. Set the LANG and LC_ALL variables to zh_TW instead of zh_TW.EUC.

Messaging: Resources for learning

The WebSphere MQ for z/OS library includes books specific to the z/OS platform, and family books that apply to all WebSphere MQ products, including WebSphere MQ for z/OS .

WebSphere MQ for z/OS publications

Publications specific to the z/OS platform are as follows:

- WebSphere MQ for z/OS Concepts and Planning Guide

  GC34-6051, describes the concepts of WebSphere MQ for z/OS and tells you how to plan your WebSphere MQ for z/OS systems.
- WebSphere MQ for z/OS System Setup Guide

  SC34-6052, tells you how to customize WebSphere MQ for z/OS after you have installed it. It also tells you how to monitor system use and performance, and how to set up security.
- WebSphere MQ for z/OS System Administration Guide

  SC34-6053, tells you how to operate WebSphere MQ for z/OS using commands, panels, and utilities, and how to write applications to administer WebSphere MQ for z/OS . The latter part of the book deals with termination, recovery, and restart.
- WebSphere MQ for z/OS Problem Determination Guide

  GC34-6054, will help you to determine the causes of WebSphere MQ for z/OS problems, resolve those problems, deal with the IBM support center, and handle APARs.
- WebSphere MQ for z/OS Messages and Codes

  GC34-6056, lists all the user messages and abend reason codes returned by WebSphere MQ for z/OS , with explanations and suggested responses. It is designed for use as a quick reference, and is linked with the WebSphere MQ for z/OS Problem Determination Guide, which you should also consult if a message indicates that there is a WebSphere MQ for z/OS problem.

WebSphere MQ family publications

WebSphere MQ family books that include information for WebSphere MQ for z/OS:

- WebSphere MQ Script (MQSC) Command Reference

  SC34-6055, describes the MQSC commands, used by system operators and administrators to manage queue managers. It introduces the commands and tells you how to use them, before describing the commands in detail, in alphabetic order.
- WebSphere MQ Intercommunication

SC34-6059, describes intercommunication between WebSphere MQ products. It introduces the concepts of intercommunication (transmission queues, message channel agent programs, and communication links) that are brought together to form message channels. It describes how geographically-separated queue managers are linked together by message channels to form a network of queue managers. It discusses the distributed-queuing management (DQM) facility of WebSphere MQ, which provides the services that enable applications to communicate using queue managers.

- WebSphere MQ Application Programming Reference

  SC34-6062, introduces the concepts of messages and queues, and gives a full description of the WebSphere MQ programming interface, including data types, function calls, attributes, return codes, and constants.
- WebSphere MQ Application Programming Guide

  SC34-6064, introduces the concepts of messages and queues, and shows you how to design, write, and build applications that use the services that WebSphere MQ provides.
- WebSphere MQ Using Java

  SC34-6066, tells you how to install, and write programs with, the WebSphere MQ classes for Java to access WebSphere MQ systems, and the WebSphere MQ classes for Java Message Service to access both Java Message Service (JMS) and WebSphere MQ applications.
- WebSphere MQ Using C++

  SC34-6067, describes the C++ programming-language binding to the WebSphere MQ Message Queue Interface (MQI). It introduces the binding, describes considerations associated with using C++ with WebSphere MQ, and describes the WebSphere MQ C++ classes.

Using message-driven beans in applications

WebSphere Application Server supports asynchronous messaging as a method of communication based on the Java Message Service (JMS) programming interface.

Message-driven beans (a type of enterprise bean defined in the EJB 2.0 specification) extend the base JMS support and the Enterprise JavaBean component model to provide automatic asynchronous messaging. When a message arrives on a destination, a listener passes the message to a new instance of a user-developed message-driven bean for processing.

You can use WebSphere Studio Application Developer to develop applications that use message-driven beans. You can use the WebSphere Application Server runtime tools, like the administrative console, to deploy and administer applications that use message-driven beans.

For more information about implementing WebSphere enterprise applications that use message-drive beans, see the following topics:

- An overview of message-driven beans
- Designing an enterprise application to use a message-driven bean
- Developing an enterprise application to use a message-driven bean
- Deploying an enterprise application to use a message-driven bean
- Configuring message listener resources for message-driven beans
- Troubleshooting problems with message-driven beans

Message-driven beans - an overview

WebSphere Application Server supports automatic asynchronous messaging with message-driven beans (a type of enterprise bean defined in the EJB 2.0 specification). Messaging with message-driven beans is shown in the figure "Message-driven beans - an overview."

The support for message-driven beans is based on the message listener service, which comprises a listener manager that controls and monitors one or more listeners. Each listener monitors a JMS destination for incoming messages. When a message arrives on the destination, the listener passes the message to a new instance of a user-developed message-driven bean (an enterprise bean) for processing. The listener then looks for the next message without waiting for the bean to return.

Messages arriving at a destination being processed by a listener have no client credentials associated with them; the messages are anonymous. Security depends on the role specified by the RunAs Identity for the message-driven bean as an EJB component. For more information about EJB security, see the Information Center topic ″EJB component security.″

You are recommended to develop a message-driven bean to delegate the business processing of incoming messages to another enterprise bean, to provide clear separation of message handling and business processing. This also enables the business processing to be invoked by either the arrival of incoming messages or, for example, from a WebSphere J2EE client.

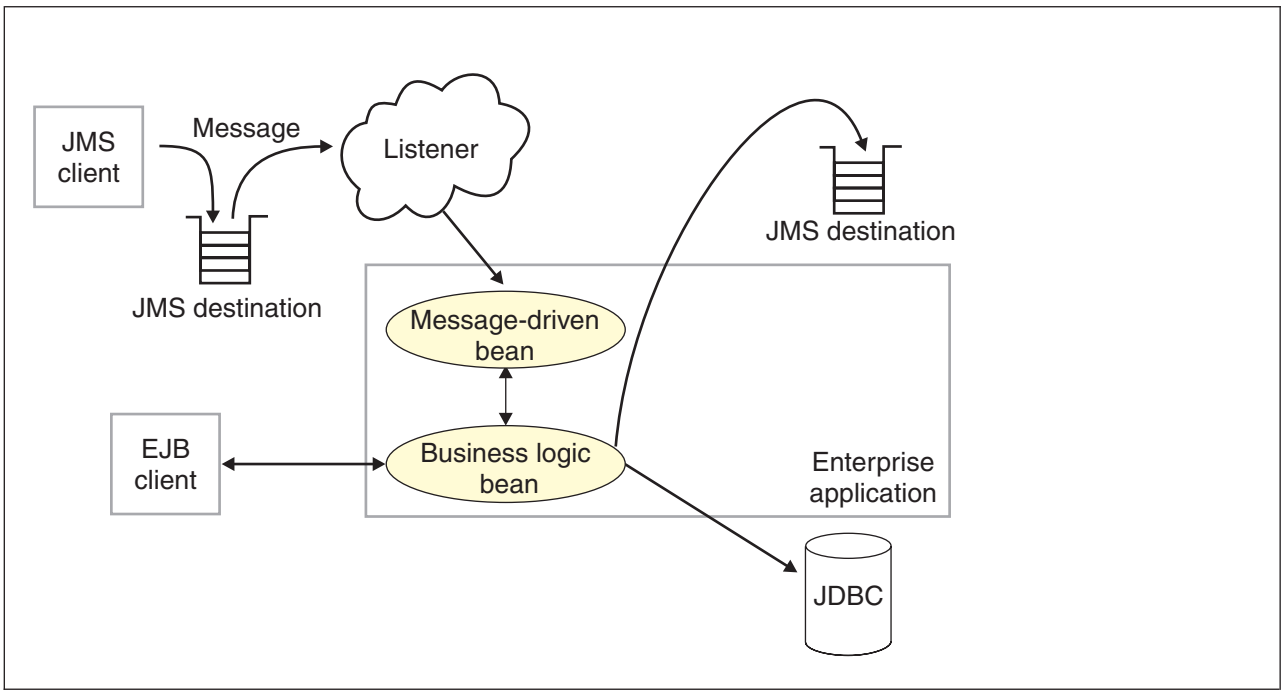

Figure 4. Message-driven beans and the message listener service. This figure shows an incoming message being passed by a JMS listener to a message-driven bean, which passes the message on to a business logic bean for business processing. This messaging is controlled by the listener manager. For more information, see the text that accompanies this figure.

Message-driven beans - components

The WebSphere Application Server support for message-driven beans is based on JMS message listeners and the message listener service, and builds on the base

support for JMS. The main components of WebSphere Application Server support for message-driven beans are shown in the following figure and described after the figure:

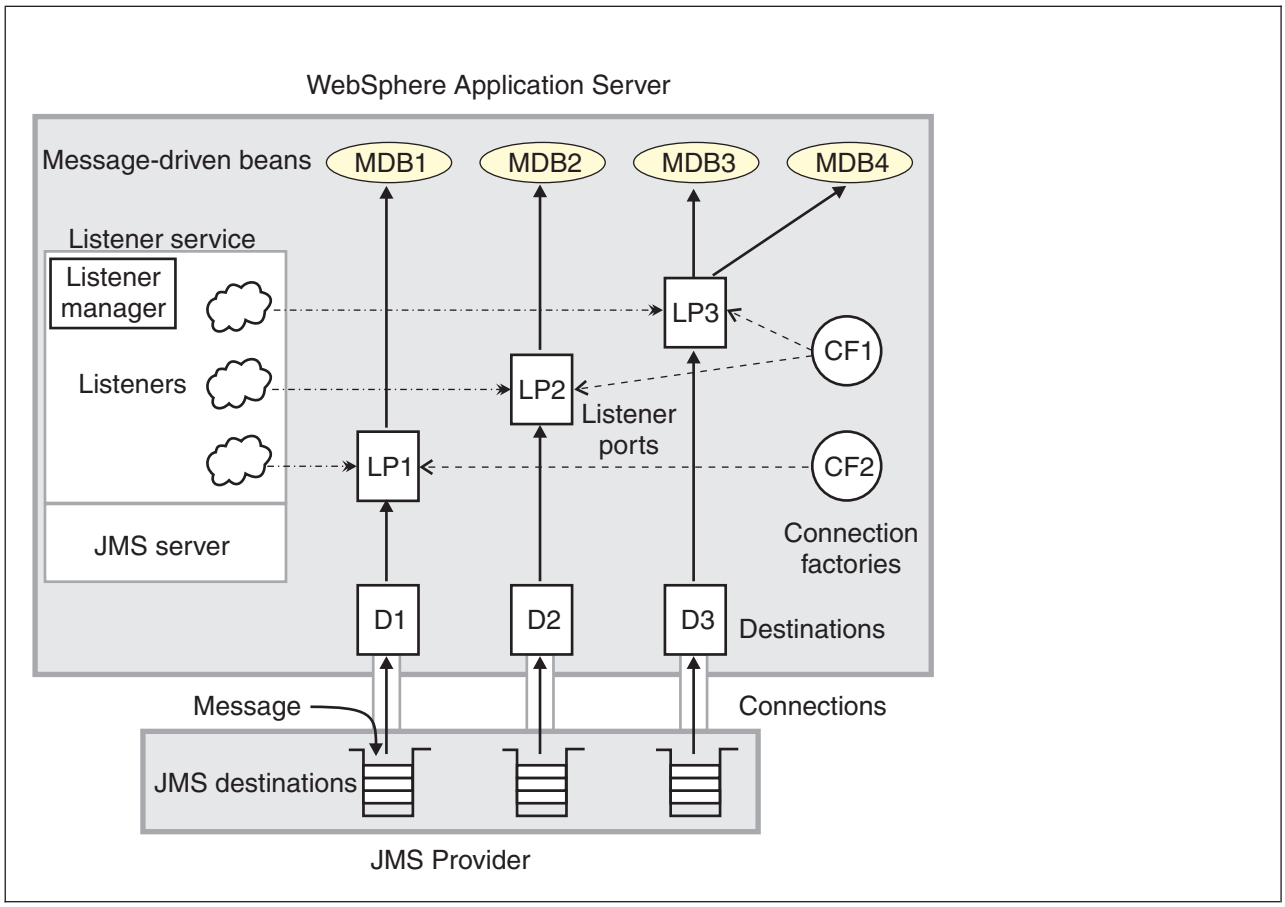

Figure 5. The main components for message-driven beans. This figure shows the main components of WebSphere support for message-driven beans, from JMS provider through a connection to a destination, listener port, then deployed message-driven bean that processes the message retrieved from the destination. Each listener port defines the association between a connection factory, destination, and a deployed message-driven bean. The other main components are the message listener service, which comprises a listener for each listener port, all controlled by the same listener manager. For more information, see the text that accompanies this figure.

The message listener service is an extension to the JMS functions of the JMS provider and provides a listener manager, which controls and monitors one or more JMS listeners.

Each listener monitors either a JMS queue destination (for point-to-point messaging) or a JMS topic destination (for publish/subscribe messaging).

A connection factory is used to create connections with the JMS provider for a specific JMS queue or topic destination. Each connection factory encapsulates the configuration parameters needed to create a connection to a JMS destination.

A listener port defines the association between a connection factory, a destination, and a deployed message-driven bean. Listener ports are used to simplify the administration of the associations between these resources.

When a deployed message-driven bean is installed, it is associated with a listener port and the listener for a destination. When a message arrives on the destination, the listener passes the message to a new instance of a message-driven bean for processing.

When an application server is started, it initializes the listener manager based on the configuration data. The listener manager creates a dynamic session thread pool for use by listeners, creates and starts listeners, and during server termination controls the cleanup of listener message service resources. Each listener completes several steps for the JMS destination that it is to monitor, including:

- Creating a JMS server session pool, and allocating JMS server sessions and session threads for incoming messages.
- Interfacing with JMS ASF to create JMS connection consumers to listen for incoming messages.
- If specified, starting a transaction and requesting that it is committed (or rolled back) when the EJB method has completed.
- Processing incoming messages by invoking the onMessage() method of the specified enterprise bean.

Message-driven beans - transaction support

Message-driven beans can handle messages read from JMS destinations within the scope of a transaction. If transaction handling is specified for a JMS destination, the JMS listener starts a global transaction *before* it reads any incoming message from that destination. When the message-driven bean processing has finished, the JMS listener commits or rolls back the transaction (using JTA transaction control).

Note:

- All messages retrieved from a specific destination have the same transactional behavior.

If messages are queued to be sent within a global transaction they are sent when the transaction is committed. If the processing of a message causes the transaction to be rolled back, then the message that caused the bean instance to be invoked is left on the JMS destination.

You can configure the **Maximum retries** property of the listener port to define the maximum number of times the listener attempts to read a message from a destination. When the Max retries limit is reached, the listener for that destination is stopped. When you have resolved the problem, you must then restart the listener.

Designing an enterprise application to use message-driven beans

This topic describes things to consider when designing an enterprise application to use message-driven beans.

The considerations in this topic are based on a generic enterprise application that uses one message-driven bean to retrieve messages from a JMS queue destination and passes the messages on to another enterprise bean that implements the business logic.

To design an enterprise application to use message-driven beans, complete the following steps:

1. Identify the JMS resources that the application is to use. This helps to identify the properties of resources that need to be used within the application and

configured as application deployment descriptors or within WebSphere Application Server.

JMS resource type	Properties
Queue connection factory	**Name:** SamplePtoPQueueConnectionFactory **JNDI Name:** Sample/JMS/QCF
Queue destination	**Name:** Q1 **JNDI Name:** Sample/JMS/Q1
Listener port (for the destination)	**Name:** SamplePtoPListenerPort **Connection Factory JNDI Name:** Sample/JMS/QCF **Destination JNDI Name:** Sample/JMS/Q1 **Maximum Sessions:** 5 **Maximum Retries:** 10 **Maximum Messages:** 1
Message-driven bean (deployment properties)	**Name:** JMSppSampleMDBBean **Transaction type:** Container **Destination type:** Queue **Listener port name:** SamplePtoPListenerPort
Business logic bean	**Name:** MyLogicBean

Ensure that you use consistent values where needed; for example, the JNDI names for the connection factory and destination must be the same for both those resources and the equivalent properties of the listener port.

2. Separation of business logic. You are recommended to develop a message-driven bean to delegate the business processing of incoming messages to another enterprise bean. This provides clear separation of message handling and business processing. This also enables the business processing to be invoked by either the arrival of incoming messages or, for example, from a WebSphere J2EE client.
3. Security considerations.

   Messages arriving at a destination being processed by a listener have no client credentials associated with them; the messages are anonymous. Security depends on the role specified by the RunAs Identity for the message-driven bean as an EJB component. For more information about EJB security, see the Information Center topic ″EJB component security.″
4. General JMS considerations For Publish/Subscribe messaging, choose the JMS server port to be used depending on your needs for transactions or performance:

   Queued port
   The TCP/IP port number of the listener port used for all point-to-point and Publish/Subscribe support.

   Direct port
   The TCP/IP port number of the listener port used for direct TCP/IP connection (non-transactional, non-persistent, and non-durable subscriptions only) for Publish/Subscribe support.

   Note: Message-driven beans cannot use the direct listener port for Publish/Subscribe support. Therefore, any topic connection factory configured with **Port**set to `Direct` cannot be used with message-driven beans.

   A non-durable subscriber can only be used in the same transactional context (for example, a global transaction or an unspecified transaction context) that existed when the subscriber was created. For more information about this context restriction, see The effect of transaction context on non-durable subscribers.

Developing an enterprise application to use message-driven beans

Use this task to develop an enterprise application to use a message-driven bean. The message-driven bean is invoked by a JMS listener when a message arrives on the input queue that the listener is monitoring.

You are recommended to develop the message-driven bean to delegate the business processing of incoming messages to another enterprise bean, to provide clear separation of message handling and business processing. This also enables the business processing to be invoked by either the arrival of incoming messages or, for example, from a WebSphere J2EE client. Responses can be handled by another enterprise bean acting as a sender bean, or handled in the message-driven bean.

You develop an enterprise application to use a message-driven bean like any other enterprise bean, except that a message-driven bean does not have a home interface or a remote interface.

This topic describes how to develop a completely new message-driven bean class. If you have a WAS 4.0 enterprise application that uses the JMS listener, you can migrate that application to use message-driven beans.

For more information about writing the message-driven bean class, see *Creating a message-driven bean* in the WebSphere Studio help bookshelf.

To develop an enterprise application to use a message-driven bean, complete the following steps:

1. Creating the Enterprise Application project, as described in the WebSphere Studio article .
2. Creating the message-driven bean class.

   You can use the New Enterprise Bean wizard of WebSphere Studio Application Developer to create an enterprise bean with a bean type of Message-driven bean. The wizard creates appropriate methods for the type of bean.

   By convention, the message bean class is named *name*Bean, where *name* is the name you assign to the message bean; for example:

   ```
   public class MyJMSppMDBBean implements MessageDrivenBean, MessageListener
   ```

   The message-driven bean class must define and implement the following methods:

   - onMessage(message), which must meet the following requirements:
     - The method must have a single argument of type `javax.jms.Message`.
     - The throws clause must *not* define any application exceptions.
     - If the message-driven bean is configured to use bean-managed transactions, it must call the javax.transaction.UserTransaction interface to scope the transactions. Because these calls occur inside the onMessage() method, the transaction scope does not include the initial message receipt. This means the application server is given one attempt to process the message.

     To handle the message within the onMessage() method (for example, to pass the message on to another enterprise bean), you use standard JMS. (This is known as bean-managed messaging.)
   - ejbCreate()

     You must define and implement an ejbCreate method for each way in which you want a new instance of an enterprise bean to be created.

- ejbRemove().

  This method is invoked by the container when a client invokes the remove method inherited by the enterprise bean's home interface from the javax.ejb.EJBHome interface. This method must contain any code that you want to execute before an enterprise bean instance is removed from the container (and the associated data is removed from the data source).

For example, the following code extract shows how to access the text and the JMS MessageID, from a JMS message of type TextMessage:
The result of this step is a message-driven bean that can be assembled into an

```
public void onMessage(javax.jms.Message msg)
{
        String text      = null;
        String messageID = null;

        try
        {
                text = ((TextMessage)msg).getText();

                System.out.println("senderBean.onMessage(), msg text2: "+text);

                //
                // store the message id to use as the Correlator value
                //
                messageID = msg.getJMSMessageID();

                // Call a private method to put the message onto another queue
                putMessage(messageID, text);
        }
        catch  (Exception err)
        {
                err.printStackTrace();
        }
        return;
}
```

Figure 6. Code example: The onMessage() method of a message bean. This figure shows a code extract for a basic onMessage() method of a sample message-driven bean. The method unpacks the incoming text message to extract the text and message identifier and calls a private putMessage method (defined within the same message bean class) to put the message onto another queue.

.EAR file for deployment.

3. Assembling and packaging the application for deployment.

   You can use WebSphere Studio to assemble and package the application for deployment.

The result of this task is an .EAR file, containing an application message-driven bean, that can be deployed in WebSphere Application Server.

After you have developed an enterprise application to use message-driven beans, configure and deploy the application; for example, define the listener ports for the message-driven beans and, optionally, change the deployment descriptor attributes for the application. For more information about configuring and deploying an application that uses message-driven beans, see Deploying an enterprise application to use message-driven beans

Deploying an enterprise application to use message-driven beans

Use this task to deploy an enterprise application to use message-driven beans.

This task description assumes that you have an .EAR file, which contains an application enterprise bean with code for message-driven beans, that can be deployed in WebSphere Application Server.

To deploy an enterprise application to use message-driven beans, complete the following steps:

1. Use the WebSphere administrative console to define the listener ports for the application, as described in Adding a new listener port.
2. 5.1 + For each message-driven bean in the application, configure the deployment attributes to match the listener port definitions, as described in the Information Center topic ″Configuring deployment attributes using the Assembly Toolkit.″
3. Use the WebSphere administrative console to install the application.

   This stage is a standard WebSphere Application Server task, as described in the Information Center topic ″Installing a new application″.

   When you install the application, you are prompted to specify the name of the listener port that the application is to use for late responses. Select the listener port, then click **OK**.

Configuring deployment attributes using the Assembly Toolkit

Use this task to configure the message-driven beans deployment attributes for an enterprise bean, to override the deployment attributes defined within the application EAR file.

You can configure the deployment attributes of an application by using the Deployment Descriptor Editor of WebSphere Studio Application Developer or the Assembly Toolkit.

This topic describes the use of the Assembly Toolkit to configure the deployment attributes of an application. This task description assumes that you have an EAR file, which contains an application enterprise bean developed as a message-driven bean, that can be deployed in WebSphere Application Server. For more details about using the Assembly Toolkit, see the Information Center topic ″Assembling applications with the Assembly Toolkit.″

To configure the message-driven beans deployment attributes for an enterprise bean, use the Assembly Toolkit to configure the deployment attributes of the application to match the listener port definitions:

1. Start the Assembly Toolkit.
2. Create or edit the application EAR file. For example, to change attributes of an existing application, use the import wizard to import the EAR file into the Assembly Toolkit. To start the import wizard:
   a. Click **File-> Import-> EAR file**
   b. Click **Next**, then select the EAR file.
   c. Click **Finish**
3. In the J2EE Hierarchy view, right-click the EJB module for the message-driven bean , then click **Open With > Deployment Descriptor Editor**. A property dialog notebook for the message-driven bean is displayed in the property pane.
4. Specify general deployment properties.
   a. In the property pane, select the Beans tab.
   b. Specify the following properties:

Transaction type

Whether the message bean manages its own transactions or the container manages transactions on behalf of the bean. All messages retrieved from a specific destination have the same transactional behavior. To enable the transactional behavior that you want, you must configure the JMS destination with the same transactional behavior as you configure for the message bean.

Bean The message bean manages its own transactions

Container

The container manages transactions on behalf of the bean

5. Specify advanced deployment properties.
   a. Specify the following properties:

Message selector

The JMS message selector to be used to determine which messages the message bean receives; for example:

```
JMSType='car' AND color='blue' AND weight>2500
```

The selector string can refer to fields in the JMS message header and fields in the message properties. Message selectors cannot reference message body values.

Acknowledge mode

How the session acknowledges any messages it receives.

This property applies only to message-driven beans that uses bean-managed transaction demarcation (**Transaction type** is set to `Bean`).

Auto Acknowledge

The session automatically acknowledges a message when it has either successfully returned from a call to receive, or the message listener it has called to process the message successfully returns.

Dups OK Acknowledge

The session lazily acknowledges the delivery of messages. This is likely to result in the delivery of some duplicate messages if JMS fails, so it should be used only by consumers that are tolerant of duplicate messages.

As defined in the EJB specification, clients cannot use using Message.acknowledge() to acknowledge messages. If a value of CLIENT_ACKNOWLEDGE is passed on the create*xxx*Session call, then messages are automatically acknowledged by the application server and Message.acknowledge() is not used.

Destination type

Whether the message bean uses a queue or topic destination.

Queue

The message bean uses a queue destination.

Topic The message bean uses a topic destination.

Subscription durability

Whether a JMS topic subscription is durable or non-durable.

Durable

A subscriber registers a durable subscription with a unique identity that is retained by JMS. Subsequent subscriber objects with the same identity resume the subscription in the state it was left in by the earlier subscriber. If there is no active subscriber for a durable subscription, JMS retains

the subscription's messages until they are received by the subscription or until they expire.

Nondurable

Non-durable subscriptions last for the lifetime of their subscriber object. This means that a client sees the messages published on a topic only while its subscriber is active. If the subscriber is not active, the client is missing messages published on its topic.

A non-durable subscriber can only be used in the same transactional context (for example, a global transaction or an unspecified transaction context) that existed when the subscriber was created. For more information about this context restriction, see The effect of transaction context on non-durable subscribers.

6. Specify bindings deployment properties.
   a. Specify the following property:
      Listener port name
      Type the name of the listener port for this message-driven bean.
7. Save your changes to the deployment descriptor.
   a. Close the deployment descriptor editor.
   b. When prompted, click **Yes** to indicate that you want to save changes to the deployment descriptor.
8. Verify the archive files.
9. Generate code for deployment for EJB modules or for enterprise applications that use EJB modules.
10. Optional: Test your completed module on a WebSphere Application Server installation. Right-click a module, click **Run on Server**, and follow the instructions in the displayed wizard. Note that **Run on Server** works on the Windows, Linux/Intel, and AIX operating systems only; you cannot deploy remotely from the Assembly Toolkit to a WebSphere Application Server installation on a UNIX operating system such as Solaris.

    Important

    Important: Use **Run On Server** for unit testing only. Assembly Server Toolkit controls the WebSphere Application Server installation and, when an application is published remotely, the Toolkit overwrites the server configuration file for that server. Do not use on production servers.

    For instructions on remote testing, see the article "Setting Up a Remote WebSphere Application Server in WebSphere Studio V5" at http://www7b.boulder.ibm.com/wsdd/techjournal/0303_yuen/yuen.html.

After assembling your application, use a systems management tool to deploy the EAR file onto the application server that is to run the application; for example, using the administrative console as described in the Information Center topic "Deploying and managing applications".

Configuring message listener resources for message-driven beans

Use the following tasks to configure resources needed by the message listener service to support message-driven beans.

- Configuring the message listener service

- Adding a new listener port
- Configuring a listener port
- Configuring security for message-driven beans

Configuring the message listener service

Use this task to configure the properties of the message listener service for an application server.

To configure the properties of the message listener service for an application server, use the administrative console to complete the following steps:

1. In the navigation pane, select **Servers-> Application Servers** This displays a table of the application servers in the administrative domain.
2. In the content pane, click the name of the application server. This displays the properties of the application server in the content pane.
3. In the Additional Properties table, select **Message Listener Service** This displays the Message Listener Service properties in the content pane.
4. Specify appropriate properties of the message listener service.
5. Optional: Specify any of the following optional properties that you need, as **Custom properties** of the message listener service: NON.ASF.RECEIVE.TIMEOUT, MQJMS.POOLING.TIMEOUT, MQJMS.POOLING.THRESHOLD, MAX.RECOVERY.RETRIES, and RECOVERY.RETRY.INTERVAL.

   For more information about these custom properties, see Custom Properties.
6. Click **OK**.
7. Save your configuration.
8. To have the changed configuration take effect, stop then restart the Application Server.

Message listener service:

The message listener service is an extension to the JMS functions of the JMS provider. It provides a listener manager that controls and monitors one or more JMS listeners, which each monitor a JMS destination on behalf of a deployed message-driven bean.

This panel displays links to the Additional Properties pages for Listener Ports and Custom Properties for the message listener service.

To view this administrative console page, click **Servers-> Application Servers-> *application_server*-> Message Listener Service**

Thread pool:

Controls the maximum number of threads the Message Listener Service is allowed to run. Select this link to display the service thread pool properties.

Adjust this parameter when multiple message-driven beans are deployed in the same application server and the sum of their maximum session values exceeds the default value of 10.

The thread pool is shared out based on demand, so if the thread pool size is equal to the number of listener ports, then there is no guarantee that one thread is available for each listener port. However, a listener port should not get blocked by busy threads because scheduling should be based on FIFO requests.

Data type	Integer
Units	Not applicable
Default	Minimum: 10, maximum 50
Range	Not applicable
Recommended	Set the minimum thread pool size to the expected low-load total message-driven beans sessions. Set the maximum thread pool size to the expected high-load total message-driven beans sessions. If the total number of sessions for all listener ports exceeds this maximum thread pool size, adjust the minimum and maximum to the appropriate size for JVM resource reuse, parallel processing of messages, and speed of delivery; for example: • Scenario 1. If JVM resource reuse=none, parallel processing=always, speed of delivery=quick, then minThreadPool=maxTotalSession and maxThreadPool does not matter (that is, maxThreadPool=maxTotalSession). • Scenario 2. If JVM resource resuse=whenPossible, parallel processing=whenPossible, speed of delivery=quick, then minThreadPool=(a tested number) < maxTotalSession and maxThreadPool=maxTotalSession.

Custom Properties:

An optional set of name and value pairs for custom properties of the message listener service.

You can use the Custom properties page to define the following properties for use by the message listener service.

- NON.ASF.RECEIVE.TIMEOUT
- MQJMS.POOLING.TIMEOUT
- MQJMS.POOLING.THRESHOLD
- MAX.RECOVERY.RETRIES
- RECOVERY.RETRY.INTERVAL

Message listener service custom properties:

Use this panel to view or change an optional set of name and value pairs for custom properties of the message listener service.

To view this administrative console page, click **Servers-> *application_server*-> Message Listener Service-> (In content pane, under Additional Properties) Custom Properties**

You can use the Custom properties page to define the following properties for use by the message listener service.

- NON.ASF.RECEIVE.TIMEOUT
- MQJMS.POOLING.TIMEOUT

- MQJMS.POOLING.THRESHOLD
- MAX.RECOVERY.RETRIES
- RECOVERY.RETRY.INTERVAL

NON.ASF.RECEIVE.TIMEOUT:

The timeout in milliseconds for synchronous message receives performed by message-driven bean listener sessions in the non-ASF mode of operation.

You should set this property to a non-zero value only if you want to enable the non-ASF mode of operation for all message-driven bean listeners on the application server.

The message listener service has two modes of operation, Application Server Facilities (ASF) and non-Application Server Facilities (non-ASF).

- The ASF mode is meant to provide concurrency and transactional support for applications. For publish/subscribe message-drive beans, the ASF mode provides better throughput and concurrency, because in the non-ASF mode the listener is single-threaded.
- The non-ASF mode is mainly for use with generic JMS providers that do not support JMS ASF, which is an optional extension to the JMS specification. The non-ASF mode is also transactional but, because the path length is shorter than the ASF mode, usually provides improved performance.

  Use non-ASF if:
  - Your generic JMS provider does not provide JMS ASF support
  - You are using message-driven beans with WebSphere topic connections with the DIRECT port, because the embedded publish/subscribe broker using that port does not support XA transactions or JMS ASF.
  - Message order is a strict requirement

Data type Integer

Units Milliseconds

Default ASF mode (custom property not created)

Range 0 or greater milliseconds

0 non-ASF mode is disabled

1 or more
The timeout in milliseconds for non-ASF message-driven bean listener synchronous session receives

Recommended If a transaction timeout occurs, the message must recycle causing extra work. If you want to use the non-ASF mode, set this property to lower than the transaction timeout, but leave spare at least the maximum duration of your message-driven bean's onMessage() method. For example, if your message-driven bean's onMessage() method typically takes a maximum of 10 seconds, and the transaction timeout is set to 120 seconds, you might set the NON.ASF.RECEIVE.TIMEOUT property to no more than 110000 (110000 milliseconds, that is 110 seconds).

MQJMS.POOLING.TIMEOUT:

The number of milliseconds after which a connection in the pool is destroyed if it has not been used.

An MQSimpleConnectionManager allocates connections on a most-recently-used basis, and destroys connections on a least-recently-used basis. By default, a connection is destroyed if it has not been used for five minutes.

Data type	Integer
Units	Milliseconds
Default	5 minutes
Range	

MQJMS.POOLING.THRESHOLD:

The maximum number of unused connections in the pool.

An MQSimpleConnectionManager allocates connections on a most-recently-used basis, and destroys connections on a least-recently-used basis. By default, a connection is destroyed if there are more than ten unused connections in the pool.

Data type	Integer
Units	Number of connections
Default	10
Range	

MAX.RECOVERY.RETRIES:

The maximum number of times that the listener service tries to get a message from a listener port before the associated listener is stopped, in the range 0 through 2147483647.

Data type	Integer
Units	Retry attempts
Default	0 (no retries)
Range	0 (no retries) through 2147483647

RECOVERY.RETRY.INTERVAL:

The time in seconds between retry attempts by the listener service to get a message from a listener port.

Data type	Integer
Units	Seconds
Default	10
Range	1 through 2147483647

Message listener port collection:

The message listener ports configured in the administrative domain

This panel displays a list of the message listener ports configured in the administrative domain. Each listener port is used with a message-driven bean to automatically receive messages from an associated JMS destination. You can use

this panel to add new listener ports or to change the properties of existing listener ports. For more information about the property fields for listener ports, see Listener port properties.

To view this administrative console page, click **Servers-> *application_server*-> Message Listener Service-> Listener Ports**

Listener port settings:

A listener port is used to simplify administration of the association between a connection factory, destination, and deployed message-driven bean.

Use this panel to view or change the configuration properties of the selected listener port.

To view this administrative console page, click **Servers-> Application Servers-> *application_server*-> Message Listener Service-> Listener Ports-> *listener_port***

Name:

The name by which the listener port is known for administrative purposes.

Data type String
Default Null

Initial state:

The state that you want the listener port to have when the application server is next restarted

Data type Enum
Units Not applicable
Default Started
Range
Started When the application server is next started, the listener port is started automatically.
Stopped When the application server is next started, the listener port is not started automatically. If message-driven beans are to use this listener port on the application server, the system administrator must start the port manually or select the Started value of this property then restart the application server.

Description:

A description of the listener port, for administrative purposes within IBM WebSphere Application Server.

Data type String
Default Null

Connection factory JNDI name:

The JNDI name for the JMS connection factory to be used by the listener port; for example, `jms/connFactory1`.

Data type	String
Default	Null

Destination JNDI name:

The JNDI name for the destination to be used by the listener port; for example, `jms/destn1`.

If the extended messaging service is to use this listener port to handle late responses, the value of this property must match the JMS response destination on the output port used by the sender bean.

You cannot use a temporary destination for late responses.

Data type	String
Default	Null

Maximum sessions:

Specifies the maximum number of concurrent sessions that a listener can have with the JMS server to process messages.

Each session corresponds to a separate listener thread and therefore controls the number of concurrently processed messages. Adjust this parameter when the JMS server does not fully use the available capacity of the machine and if you do not need to process messages in a specific message order.

Data type	Integer
Units	Sessions
Default	1
Range	1 through 2147483647
Recommended	• If you want to process messages in a strict message order, set the value to 1, so only one thread is ever processing messages. • If you want to process multiple messages simultaneously (known as "message concurrency"), set this property to a value greater than 1. Keep this value as low as possible to prevent overloading client applications. A good starting point for a 100% JMS workload with short transaction times is 2 to 4 sessions per processor. If longer running transactions exist, you may need more sessions, which should be determined by experimentation. • If you are using XA transactions, this property should always be set to 1. If it this property is set to a higher value, multiple messages are delivered in the same transaction, which is usually not the desired behavior.

Maximum retries:

The maximum number of times that the listener tries to deliver a message before the listener is stopped, in the range 0 through 2147483647.

The maximum number of times that the listener tries to deliver a message to a message-driven bean instance before the listener is stopped.

Data type	Integer
Units	Retry attempts
Default	0 (no retries)
Range	0 (no retries) through 2147483647

Maximum messages:

The maximum number of messages that the listener can process in one session with the JMS server.

For WebSphere embedded messaging or WebSphere MQ as the JMS provider, the listener processes all messages in the session as one batch within the same transaction. For a generic JMS provider, the listener processes each message in the session within a separate transaction.

Data type	Integer
Units	Number of messages
Default	1
Range	1 through 2147483647
Recommended	*For WebSphere embedded messaging or WebSphere MQ as the JMS provider*, if you want to process multiple messages in a single transaction, then set this value to more than 1. This enables multiple messages to be batch-processed into a single transaction, and eliminates much of the overhead of transactions on JMS messages. **CAUTION:** • If one message in the batch fails processing with an exception, the entire batch of messages is put back on the queue for processing. • Any resource lock held by any of the interactions for the individual messages are held for the duration of the entire batch. • Depending on the amount of processing that messages need, and if XA transactions are being used, setting a value greater than 1 can cause the transaction to time out. If an XA transaction does time out routinely because processing multiple messages exceeds the transaction timeout, reduce this property to 1 (to limit processing to one message per transaction) or increase your transaction timeout.

Adding a new listener port

Use this task to add a new listener port to the message listener service, so that message-driven beans can be associated with the port to retrieve messages.

To add a new listener port, use the administrative console to complete the following steps:

1. In the navigation pane, select **Servers-> Application Servers** This displays a table of the application servers in the administrative domain.
2. In the content pane, click the name of the application server. This displays the properties of the application server in the content pane.
3. In the Additional Properties table, select **Message Listener Service** This displays the Message Listener Service properties in the content pane.
4. In the content pane, select **Listener Ports**. This displays a list of the listener ports.
5. In the content pane, click **New**.
6. Specify appropriate properties for the listener port.
7. Click **OK**.
8. To save your configuration, click **Save** on the task bar of the Administrative console window.
9. To have the changed configuration take effect, stop then restart the application server.

If enabled, the listener port is started automatically when a message-driven bean associated with that port is installed.

Configuring a listener port

Use this task to change the properties of an existing listener port, used by message-driven beans associated with the port to retrieve messages.

To configure the properties of a listener port, use the administrative console to complete the following steps:

1. In the navigation pane, select **Servers-> Application Servers** This displays a table of the application servers in the administrative domain.
2. In the content pane, click the name of the application server. This displays the properties of the application server in the content pane.
3. In the Additional Properties table, select **Message Listener Service** This displays the Message Listener Service properties in the content pane.
4. In the content pane, click **Listener Ports**. This displays a list of the listener ports.
5. Click the listener port that you want to modify. This displays the properties of the listener port in the content pane.
6. Specify appropriate properties for the listener port.
7. Click **OK**.
8. To save your configuration, click **Save** on the task bar of the Administrative console window.
9. To have the changed configuration take effect, stop then restart the application server.

Deleting a listener port

Use this task to delete a listener port from the message listener service, to prevent message-driven beans associated with the port from retrieving messages.

To delete a listener port, use the administrative console to complete the following steps:

1. In the navigation pane, select **Servers-> Application Servers** This displays a table of the application servers in the administrative domain.
2. In the content pane, click the name of the application server. This displays the properties of the application server in the content pane.
3. In the Additional Properties table, select **Message Listener Service** This displays the Message Listener Service properties in the content pane.
4. In the content pane, select **Listener Ports**. This displays a list of the listener ports.
5. In the content pane, select the checkbox for the listener port that you want to delete.
6. Click **Delete**. This action stops the port (needed to allow the port to be deleted) then deletes the port.
7. To save your configuration, click **Save** on the task bar of the Administrative console window.
8. To have the changed configuration take effect, stop then restart the application server.

Configuring security for message-driven beans

Use this task to configure resource security and security permissions for message-driven beans.

Messages arriving at a listener port have no client credentials associated with them. The messages are anonymous.

To call secure enterprise beans from a message-driven bean, the message-driven bean needs to be configured with a RunAs Identity deployment descriptor. Security depends on the role specified by the RunAs Identity for the message-driven bean as an EJB component.

For more information about EJB security, see the Information Center topic ″EJB component security.″ For more information about configuring security for your application, see the Information Center topic ″Assembling secured applications.″

JMS connections used by message-driven beans can benefit from the added security of using J2C container-managed authentication. To enable the use of J2C container authentication aliases and mapping, define a J2C container-managed alias on the JMS connection factory definition that the MDB is using to listen upon (defined by the **Connection factory JNDI name** property of the listener port). If defined, the listener uses the container-managed authentication alias for its JMSConnection security credentials instead of any application-managed alias. To set the container-managed alias, use the administrative console to complete the following steps:

1. To display the listener port settings, click **Servers->** ***application_server*****-> Message Listener Service-> Listener Ports->** ***listener_port***
2. To get the name of the JMS connection factory, look at the **Connection factory JNDI name** property.
3. Display the JMS connection factory properties. For example, to display the properties of a queue connection factory provided by the embedded WebSphere JMS provider, click **Resources-> WebSphere JMS Provider-> (In content pane, under Additional Properties) WebSphere Queue Connection Factories->** ***connection_factory***

4. Set the **Container-managed Authentication Alias** property.
5. Click **OK**

Administering listener ports

Use the following tasks to administer listener ports, which each define the association between a connection factory, a destination, and a message-driven bean.

You can use the WebSphere administrative console to administer listener ports, as described in the following tasks.

- Adding a new listener port

  Use this task to create a new listener port, to specify a new association between a connection factory, a destination, and a message-driven bean. This enables deployed message-driven beans associated with the port to retrieve messages from the destination.
- Configuring a listener port

  Use this task to view or change the configuration properties of a listener port.
- Starting a listener port

  Use this task to start a listener port manually.
- Stopping a listener port

  Use this task to stop a listener port manually.

Note: If configured as enabled, a listener port is started automatically when a message-driven bean associated with that port is installed. You do not normally need to start or stop a listener port manually.

Starting a listener port:

Use this task to start a listener port on an application server, to enable the listeners for message-driven beans associated with the port to retrieve messages.

A listener is active, that is able to receive messages from a destination, if the deployed message-driven bean, listener port, and message listener service are all started. Although you can start these components in any order, they must all be in a started state before the listener can retrieve messages.

If configured as enabled, a listener port is started automatically when a message-driven bean associated with that port is installed. However, you can start a listener port manually, as described in this topic.

When a listener port is started, the listener manager tries to start the listeners for each message-driven bean associated with the port. If a message-driven bean is stopped, the port is started but the listener is not started, and remains stopped. If you start a message-driven bean, the related listener is started.

To start a listener port on an application server, use the administrative console to complete the following steps:

1. If you want the listener for a deployed message-driven bean to be able to receive messages at the port, check that the message-driven bean has been started.
2. In the navigation pane, select **Servers-> Application Servers** This displays a table of the application servers in the administrative domain.
3. In the content pane, click the name of the application server. This displays the properties of the application server in the content pane.

4. In the Additional Properties table, select **Message Listener Service** This displays the Message Listener Service properties in the content pane.
5. In the content pane, select **Listener Ports**. This displays a list of the listener ports.
6. Select the checkbox for the listener port that you want to start.
7. Click **Start**.
8. To save your configuration, click **Save** on the task bar of the Administrative console window.

Stopping a listener port:

Use this task to stop a listener port on an application server, to prevent the listeners for message-driven beans associated with the port from retrieving messages.

When you stop a listener port as described in this topic, the listener manager stops the listeners for all message-driven beans associated with the port.

To stop a listener port on an application server, use the administrative console to complete the following steps:

1. In the navigation pane, select **Servers-> Application Servers** This displays a table of the application servers in the administrative domain.
2. In the content pane, click the name of the application server. This displays the properties of the application server in the content pane.
3. In the Additional Properties table, select **Message Listener Service** This displays the Message Listener Service properties in the content pane.
4. In the content pane, select **Listener Ports**. This displays a list of the listener ports.
5. In the content pane, select the listener port that you want to stop.
6. Click **Stop**.
7. To save your configuration, click **Save** on the task bar of the Administrative console window.
8. To have the changed configuration take effect, stop then restart the application server.

Important files for message-driven beans and extended messaging

The following files in the WAS_HOME/temp directory are important for the operation of the WebSphere Application Server messaging service, so should not be deleted. If you do need to delete the WAS_HOME/temp directory or other files in it, ensure that you preserve the following files.

server_name-**durableSubscriptions.ser**
: You should not delete this file, because the messaging service uses it to keep track of durable subscriptions for message-driven beans. If you uninstall an application that contains a message-driven bean, this file is used to unsubscribe the durable subscription.

server_name-**AsyncMessageRequestLog.ser**
: You should not delete this file, because the messaging service uses it to keep track of late responses that need to be delivered to the late response message handler for the extended messaging provider.

Troubleshooting message-driven beans

Use this overview task to help resolve a problem that you think is related to message-driven beans.

Message-driven beans support uses the standard WebSphere Application Server troubleshooting facilities. If you encounter a problem that you think might be related to the message-driven beans, complete the following stages:

1. Check for more informational and error messages that might provide a clue to a related problem. If the JMS server is running, but you have problems accessing JMS resources, check for more error messages and extra details about the problem.

   For messages related to WebSphere Messaging, look for the prefixes: MSGS and WMSG.
2. Check the Release Notes for specific problems and workarounds The section *Possible Problems and Suggested Fixes* of the Release Notes, available from the WebSphere Application Server library web site, is updated regularly to contain information about known defects and their workarounds. Check the latest version of the Release Notes for any information about your problem. If the Release Notes does not contain any information about your problem, you can also search the Technotes database on the WebSphere Application Server web site.
3. Check that message listener service has started. The message listener service is an extension to the JMS functions of the JMS provider. It provides a listener manager that controls and monitors one or more JMS listeners, which each monitor a JMS destination on behalf of a deployed message-driven bean.
4. Check your JMS resource configurations If the WebSphere Messaging functions seem to be running properly (the JMS server is running without problems), check that the JMS resources have been configured correctly. For example, check that the listener ports have been configured correctly and have been started.
5. Check for problems with the WebSphere Messaging functions For more information about troubleshooting WebSphere Messaging, see the related topics.
6. Get a detailed exception dump for messaging. If the information obtained in the preceding steps is still inconclusive, you can enable the application server debug trace for the ″Messaging″ group to provide a detailed exception dump.

Message-driven beans samples

The following examples are provided, as part of the WebSphere Samples Gallery, to illustrate use of the message-driven beans support. When the Samples are installed on your local machine, they are available to try out. Locate them at http://localhost:9080/WSsamples/. (The default port is 9080.) For more information about where to find the Samples Gallery, see the Information Center topic ″Samples Gallery.″

- Point-to-point samples:
  - ″Tutorial: Creating JMS message sample″

    This tutorial is designed to help you develop and deploy a JMS message sample application that tests the WebSphere Application Server message-driven beans support in a point-to-point scenario. This sample illustrates how to develop and deploy an application that comprises the following components:
    - A Java/JMS program that writes a message to a queue.
    - A message-driven bean that is invoked by a JMS listener when a message arrives on a defined queue.

For more information about this sample, see the samples article ″Tutorial: Creating JMS message sample″ that is installed with the Samples option.

  - ″Sample: Message Listener (point-to-point)″

    This sample is designed to demonstrate the use and behavior of message-driven beans for a simple point-to-point scenario. This sample uses the JMS message sample deployed in the sample above.

    For more information about this sample, see the samples article ″Sample: Message Listener (Point-to-Point)″ that is installed with the Samples option.

- Publish/subscribe samples
  - ″Tutorial: Creating JMS message publish/subscribe sample″

    This tutorial is designed to help you develop and deploy a JMS message sample application that tests the WebSphere Application Server message-driven beans support in a publish/subscribe scenario. This sample illustrates how to develop and deploy an application that comprises the following components:
    - A client program that starts the message sequence by publishing a message to a selected topic.
    - A message-driven bean that is invoked by a JMS listener when the broker passes a message to the listener from a topic to which it has subscribed.

    For more information about this sample, see the samples article ″Tutorial: Creating JMS message publish/subscribe sample″ that is installed with the Samples option.
  - ″Sample: Message Listener (publish/subscribe)″

    This sample is designed to demonstrate the use and behavior of message-driven beans for a simple publish/subscribe scenario. This sample uses the JMS message sample deployed in the publish/subscribe sample above.

    For more information about this sample, see the samples article ″Sample: Message Listener (publish/subscribe)″ that is installed with the Samples option.

Chapter 4. Using mail

Using JavaMail API, a code segment can be embedded in any Java 2 Enterprise Edition (J2EE) application component, such as an EJB or a servlet, allowing the application to send a message and save a copy of the mail to the Sent folder.

The following is a code sample that you would embed in a J2EE application:

```
javax.naming.InitialContext ctx = new javax.naming.InitialContext();

   javax.mail.Session mail_session = (javax.mail.Session)
  ctx.lookup("java:comp/env/mail/MailSession3");
```

Note: The previous line of code should be entered on one line; it is split here for formatting purposes.

```
 MimeMessage msg = new MimeMessage(mail_session);

   msg.setRecipients(Message.RecipientType.TO, InternetAddress.parse("bob@coldmail.net"));

   msg.setFrom(new InternetAddress("alice@mail.eedge.com"));

   msg.setSubject("Important message from eEdge.com");

   msg.setText(msg_text);

   Transport.send(msg);


   Store store = mail_session.getStore();

   store.connect();

   Folder f = store.getFolder("Sent");

   if (!f.exists()) f.create(Folder.HOLDS_MESSAGES);

   f.appendMessages(new Message[] {msg});
```

J2EE applications can use JavaMail APIs by looking up references to logically named mail connection factories through the `java:comp/env/mail` subcontext declared in the application deployment descriptor and mapped to installation specific mail session resources. As in the case of other J2EE resources, this can be done in order to eliminate the need for the application to hard code references to external resources.

1. Locate a resource through Java Naming and Directory Interface (JNDI). The J2EE specification considers a mail session instance as a resource, or a factory from which mail transport and store connections can be obtained. You should never *hardcode* mail sessions, namely fill up a Properties object, then use it to new up a javax.mai.Session object. Instead, you must follow the J2EE programming model of configuring resources through the system facilities and then locating them through JNDI lookups.

   In the sample code above, the line `javax.mail.Session mail_session = (javax.mail.Session) ctx.lookup("java:comp/env/mail/MailSession3");` is an example of not hard coding a mail session and using a resource name located through JNDI. You can consider the lookup name `mail/MailSession3`, as a *soft link* to the real resource.

2. Define resource references while assembling your application. You must define a resource reference for the mail resource in the deployment descriptor of the component, because a mail session is referenced in the JNDI lookup. Typically, you can use the Assembly Toolkit (ATK) shipped with WebSphere Application Server. To learn more about using the ATK see the article *Starting the Assembly Toolkit*.

   When you create this reference, be sure that the name of the reference matches the name used in the code. For example, the code above uses `java:comp/env/mail/MailSession3` in its lookup, therefore the name of this reference must be `mail/Session3` and the type of the resource must be`javax.mail.Session`. After being defined, the deployment descriptor contains the following entry for the mail resource reference:

   ```
   <resource-reference>
   <description>description</description>
   <res-ref-name>mail/MailSession3</res-ref-name>
   <res-type>javax.mail.Session</res-type>
   <res-auth>Container</res-auth>
   ```

3. Configure mail providers and sessions. The sample code references a mail resource, the deployment descriptor declares the reference, but the resource itself does not exist yet. Now you need to configure the mail resource that is referenced by your application component. Notice that the mail session you configure must have both its transport and mail access portions defined; the former required because the code is sending a message, the latter because it also saves a copy to the local mail store. When you configure the mail session, you will be asked to specify a JNDI name. This is an important name which you will require when you install your application and link up the resource references in your application with the real resources that you have configured.
4. Install your application. You can install your application using either the administrative console or the scripting tool. One important step in the install process is that the system will go through all resource references, among other things, and expect you to supply a JNDI name for each of them. This is not an arbitrary JNDI name but the JNDI name given to a particular, configured resource which is the target of the reference.
5. Manage existing mail providers and sessions. You can update and remove mail providers and sessions.

   To update mail providers and sessions:

   a. Open the administrative console.
   b. Click **Resources** > **Mail Providers** in the console navigation tree. Then, click **Mail Provider** > *mail_provider* > **Mail Session**.
   c. Click the *mail_provider* or *mail_session* that you want to modify. To remove a mail provider or mail session, click **Remove** after making your selection.
   d. Click **Apply** or **OK**.
   e. Save the configuration.

If your application has a client, you can update mail providers and mail sessions using the Application Client Resource Configuration Tool (ACRCT).

Configuring mail providers and sessions

WebSphere Application Server includes a default mail provider called *built-in* provider. If you use the default mail provider you only have to configure the mail session, which is the last step in this task. To use the customized mail provider you must first create the mail provider and session:

1. Open the administrative console.
2. Click **Resources** > **Mail Providers**.
3. Create the mail provider.
   a. Click **New**.
   b. Type the name of the mail provider in the **Name** field.
   c. Click **Apply** or **OK**.
4. Define the protocol provider for the mail provider.
   a. Click *mail_provider*.
   b. Click **Protocol Providers**.
   c. Click **New**.
   d. Type the protocol name in the **Protocol** field.
   e. Type the classname in the **Classname** field.
   f. Click **Apply** or **OK**.

   Ensure that every mail session is defined under a parent mail provider. Select a mail provider first and then create your new mail session.
5. Create the **mail session**.
   a. Click *mail_provider*.
   b. Click **Mail Sessions**.
   c. Click **New**.
   d. Type the mail session name in the **Name** field.
   e. Type the JNDI name in the **JNDI Name** field.
   f. Click **Apply** or **OK**.
6. Configure the mail session.
   a. Click *mail_provider*.
   b. Click **Mail Sessions**.
   c. Click *mail_session*.
   d. Make changes to appropriate fields.
   e. Click **Apply** or **OK**.

If your application has a client you can configure JavaMail providers and sessions using the Application Client Resource Configuration Tool (ACRCT).

Mail provider collection

Use this page to implement JavaMail and create mail sessions, which are a collection of protocol providers such as the WebSphere Application Server built-in mail provider that encompasses three protocol providers: Simple Mail Transfer Protocol (SMTP), Internet Message Access Protocol (IMAP) and Post Office Protocol (POP3).

To view this administrative console page, click **Resources** > **Mail Providers**.

Name

Specifies the name of the JavaMail resource provider.

Description

Specifies the resource provider description.

Mail provider settings

Use this page to implement JavaMail and create mail sessions, which are a collection of protocol providers such as the WebSphere Application Server built-in mail provider that encompasses three protocol providers: Simple Mail Transfer Protocol (SMTP), Internet Message Access Protocol (IMAP) and Post Office Protocol (POP3).

To view this administrative console page, click **Resources** > **Mail Providers** > *mail_provider*.

Scope
Specifies the scope of the configured resource. This value indicates the configuration location for the configuration file.

Name
Specifies the name of the JavaMail resource provider.

Description
Specifies the mail provider description.

Protocol providers collection

Use this page to select or add a new protocol provider to interact with JavaMail APIs and mail servers that run on pertaining protocols. An example is Simple Mail Transfer Protocol (SMTP), which is a popular transport protocol for sending mail. JavaMail applications can connect to an SMTP server and send mail through it by using the protocol provider.

To view this administrative console page, click **Resources** > **Mail Providers** > *mail_provider* > **Protocol Providers**.

Protocol
Specifies the configuration of the protocol provider for a given protocol.

Classname
Specifies the implementation class for the specific protocol provider (also known as JavaMail service provider).

Classpath
Specifies the path to the implementation class for the specific protocol provider (also known as JavaMail service provider).

Type
Specifies the type of protocol provider. Valid options are **STORE** or **TRANSPORT**.

Protocol providers settings

Use this page to configure protocol provider settings.

To view this administrative console page, click **Resources** > **Mail Providers** > *mail_provider* > **Protocol Providers** > *protocol_provider*.

Protocol
Specifies the configuration of the protocol provider for a given protocol.

Classname
Specifies the implementation class for the specific protocol provider (also known as JavaMail service provider).

Classpath
Specifies the path to the implementation class for the specific protocol provider (also known as JavaMail service provider).

Type
Specifies the type of protocol provider. Valid options are **STORE** or **TRANSPORT**.

Mail session collection
Use this page to configure mail session properties defined under the parent mail provider.

To view this administrative console page, click **Resources** > **Mail Providers** > *mail_provider* > **Mail Sessions**.

Name
Specifies the administrative name of the JavaMail session object.

JNDI Name
Specifies the Java Naming and Directory Interface (JNDI) name for the resource, including any naming subcontexts.

This name provides the link between the platform binding information for resources defined in the client application deployment descriptor and the actual resources bound into JNDI by the platform.

Description
Specifies an optional description for your administrative records.

Category
Specifies an optional collection for classifying or grouping sessions.

Mail session settings
Use this page to configure specific mail providers.

To view this administrative console page, click **Resources**> **Mail Providers** > *mail_provider* > **Mail Sessions** > *mail_session*.

Name
Specifies the administrative name of the JavaMail session object.

JNDI Name
Specifies the Java Naming and Directory Interface (JNDI) name for the resource, including any naming subcontexts.

This name provides the link between the platform binding information for resources defined in the client application deployment descriptor and the actual resources bound into JNDI by the platform.

Description
Specifies an optional description for your administrative records.

Category
Specifies an optional collection for classifying or grouping sessions.

Mail Transport Host
Specifies the server accessed when sending mail.

Mail Transport Protocol
Specifies the transport protocol used when sending mail.

Mail Transport User
Specifies the user ID when the mail transport host requires authentication.

This setting is not generally used for most mail servers. Leave this field blank unless you use a mail server that requires a user ID and password.

Mail Transport Password
Specifies the password when the mail transport host requires authentication.

This setting is not generally used for most mail servers. Leave this field blank unless you use a mail server that requires a user ID and password.

Mail from
Specifies the mail originator.

This value represents the Internet e-mail address that, by default, displays in the received message, as either the `From` or the `Reply-To` address. The recipient's reply comes to this address.

Mail Store Host
Specifies the server accessed when receiving the mail.

This setting, combined with the mail store user ID and password, represents a valid mail account. For example, if the mail account is *john_william@my.company.com* then the mail store host is *my.company.com*.

Mail Store Protocol
Specifies the protocol used when receiving mail; it could be IMAP, POP3 or any store protocol for which the user has installed a provider.

Mail Store User
Specifies the user ID for the given mail account.

For example, if the mail account is *john_william@my.company.com* then the user is *john_william*.

Mail Store Password
Specifies the password for the given mail account .

For example, if the mail account is *john_william@my.company.com* then enter the password for ID *john_william*.

Debug
Toggles debug mode on and off for this mail session.

JavaMail API

The JavaMail APIs provide a platform and protocol-independent framework for building Java-based mail client applications.

WebSphere Application Server supports JavaMail, Version 1.2 and the JavaBeans Activation Framework (JAF) Version 1.0. In WebSphere Application Server, the JavaMail API is supported in all Web application components, namely:
- Servlets
- Java Server Pages (JSP) files
- Enterprise beans
- Application clients

The JavaMail APIs are generic for sending and reading mail. They require service providers, known in WebSphere Application Server as protocol providers, to interact with mail servers that run on pertaining protocols.

For example, Simple Mail Transfer Protocol (SMTP) is a popular transport protocol for sending mail. JavaMail applications can connect to an SMTP server and send mail through it by using this SMTP protocol provider.

In addition to service providers, the JavaMail API requires the Java Application Framework (JAF) to handle mail content that is not plain text, including Multipurpose Internet Mail Extensions (MIME), URL pages, and file attachments.

The JavaMail APIs, the JAF, the service providers and the protocols are shipped as part of WebSphere Application Server using the following Sun licensed packages:
- `mail.jar` - Contains the JavaMail APIs, and the SMTP, IMAP, and POP3 service providers.
- `activation.jar` - Contains the JavaBeans Activation Framework.

Mail providers and mail sessions

A JavaMail service provider is a driver that supports JavaMail interaction with mail servers using a particular mail protocol. WebSphere Application Server includes service providers, also known as *protocol providers*, for mail protocols including Simple Mail Transfer Protocol (SMTP), Internet Message Access Protocol (IMAP), and Post Office Protocol 3 (POP3).

Mail provider encapsulates a collection of protocol providers. For example, WebSphere Application Server has a built-in mail provider that encompasses the three protocol providers: SMTP, IMAP and POP3. These protocol providers are installed as the default and suffice for most applications.

If you have a particular application that requires custom protocol providers, you must first follow the steps outlined in the ″JavaMail API Design Specification, V1.2, Chapter 5 - The Mail Session″ to install your own protocol providers. See the article, *Mail: Resources for learning*, for a link to this documentation.

Mail sessions are represented by the `javax.mail.Session` class. A mail `Session` object authenticates users, and controls users' access to messaging systems.

To create platform-independent applications, use a resource factory reference to create a JavaMail session. A resource factory is an object that provides access to resources in the deployed environment of a program using the naming conventions defined by the Java Naming and Directory Interface (JNDI).

Ensure that every mail session is defined under a parent mail provider. Select a mail provider first and then create your new mail session.

Mail migration tip

Specifications for Java Server Page(JSP) 1.2 changes the way the EmailBean class works with Email.jsp.

The specifications state that the JSP container creates a JSP page implementation class for each JSP page. The name of the JSP page implementation class is implementation dependent. The JSP page implementation object belongs to an implementation-dependent named package which can vary between one JSP and another, therefore minimal assumptions should be made. The unnamed package should not be used without explicit import of the class.

Following these specifications, you should place EmailBean.class in a package referred to it by the fully qualified packageName in Email.jsp. Otherwise, Email.jsp is unable to find EmailBean.class.

JavaMail security permissions best practices

In many of its activities, the JavaMail API needs to access certain configuration files. The JavaMail and JavaBeans Activation Framework binary packages themselves already contain the necessary configuration files. However, JavaMail allows the user to define user-specific and installation-specific configuration files to meet special requirements.

The two locations where such configuration files can exist are `<user.home>`and `<java.home>/lib`. For example, if the JavaMail API needs to access a file named `mailcap` when sending a message, it first tries to access `<user.home>/.mailcap`. If that attempt fails, either due to lack of security permission or a nonexistent file, the API continues to try`<java.home> /lib/mailcap`. If that attempts also fails, it will continue and try META-INF/mailcap in the classpath, which actually leads to the configuration files contained in the `mail.jar`and `activation.jar` files. WebSphere Application Server uses the general-purpose JavaMail configuration files contained in the `mail.jar` and `activation.jar` files and does not put any mail configuration files in `<user.home>`and `<java.home>/lib`. File read permission for both the `mail.jar` and `activation.jar` files is granted to all applications to ensure proper functioning of the JavaMail API, as shown in the following segment of the `app.policy` file:

```
grant codeBase "file:${application}" {
 // The following are required by Java mail
permission java.io.FilePermission
 "${was.install.root}${/}java${/}jre${/}lib${/}ext${/}mail.jar", "read";
permission java.io.FilePermission
 "${was.install.root}${/}java${/}jre${/}lib${/}ext${/}activation.jar", "read";
```

Note: The previous two lines of code (both beginning `permission java.io.FilePermission`) must be entered on one line each; they are split here for formatting purposes.

```
};
```

JavaMail code attempts to access configuration files at `<user.home>`and `<java.home>/lib` causing `AccessControlExceptions` to be thrown, since there is no file read permission granted for those two locations by default. This activity does not affect the proper functioning of the JavaMail API, but you might see a large

amount of JavaMail-related security exceptions reported in the system log, which might swamp harmful errors that you are looking for. If this situation is a problem, consider adding two more permission lines to the permission block above. This should eliminate most, if not all, JavaMail-related harmless security exceptions from the log file. The application permission block in the `app.policy` file now looks like:

```
 grant codeBase "file:${application}" {
// The following are required by Java mail
permission java.io.FilePermission
 "${was.install.root}${/}java${/}jre${/}lib${/}ext${/}mail.jar", "read";
permission java.io.FilePermission
 "${was.install.root}${/}java${/}jre${/}lib${/}ext${/}activation.jar", "read";
```

Note: The previous two lines of code (both beginning `permission java.io.FilePermission`) must be entered on one line each; they are split here for formatting purposes.

```
permission java.io.FilePermission "${user.home}${/}.mailcap", "read";
permission java.io.FilePermission "${java.home}${/}lib${/}mailcap", "read";
 };
```

Mail: Resources for learning

Use the following links to find relevant supplemental information about the JavaMail API. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

These links are provided for convenience. Often, the information is not specific to the IBM WebSphere Application Server product, but is useful all or in part for understanding the product. When possible, links are provided to technical papers and Redbooks that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas.

Programming model and decisions

- JavaMail documentation

Programming specifications

- JavaMail 1.2 API documentation (Sun's javadoc)

Chapter 5. Using URL resources within an application

Java 2 Enterprise Edition (J2EE) applications can use Uniform Resource Locators (URLs) by looking up references to logically named URL connection factories through the `java:comp/env/url` subcontext declared in the application deployment descriptor and mapped to installation specific URL resources. As in the case of other J2EE resources, this can be done in order to eliminate the need for the application to hard code references to external resources. The process is the same used with other J2EE resources, such as JDBC and JavaMail.

1. Develop an application that relies on naming features.
2. Define resource references while assembling your application. A URL resource that uses a built-in protocol, such as http, ftp or file, can use the default URL provider. URL resources that use other protocols need to use a custom URL provider.
3. Configure your URL resources within an application.
   a. Open the administrative console.
   b. Click **Resources**>**URL Providers** in the console navigation tree.
   c. Click *URL_provider*>**URLs**.
4. Optional: Configure URL providers and URLs within an application client using the Application Client Resource Configuration Tool (ACRCT).
5. Manage URL providers and URLs used by the deployed application. To update or remove existing URL configurations:
   a. Open the administrative console.
   b. Click **Resources** > **URL Providers** in the console navigation tree.
   c. Click **URL Provider** > **URLs**.
   d. Select the URL to modify.
   e. Modify the URL properties.
   f. Click **Apply** or **OK**.

   To remove URL providers and URLs, after step 2, Click *URL_provider* > URLs. Select the URL you want to remove and click **Delete**. Then, click **Apply** or **OK**.

URLs

A Uniform Resource Locator (URL) is an identifier that points to an electronically accessible resource, such as a directory file on a machine in a network, or a document stored in a database.

URLs appear in the format *scheme*:*scheme_information*.

You can represent a *scheme* as `http`, `ftp`, `file`, or another term that identifies the type of resource and the mechanism by which you can access the resource.

In a World Wide Web browser location or address box, a URL for a file available using HyperText Transfer Protocol (HTTP) starts with `http:`. An example is `http://www.ibm.com`. Files available using File Transfer Protocol (FTP) start with `ftp:`. Files available locally start with `file:`.

The *scheme_information* commonly identifies the Internet machine making a resource available, the path to that resource, and the resource name. The

scheme_information for HTTP, FTP and File generally starts with two slashes (//), then provides the Internet address separated from the resource path name with one slash (/). For example,

`http://www-4.ibm.com/software/webservers/appserv/library.html`.

For HTTP and FTP, the path name ends in a slash when the URL points to a directory. In such cases, the server generally returns the default index for the directory.

URL provider collection

Use this page to create new URL providers to handle URL protocols that are not handled by the Java Developer Kit (JDK), for example protocols other than http, ftp and file. A URL provider implements the functionality for a particular URL protocol. This provider is comprised of two classes that extend `java.net.URLStreamHandler` and `java.net.URLConnection`. A default URL provider is included in the initial product configuration. This provider utilizes the URL support provided by the JDK. Any URL resource with protocols based on Java 2 Standard Edition 1.3.1, such as HTTP or FTP, can use the default URL provider.

To view this administrative console page, click **Resources** > **URL Providers**.

Name

Specifies the administrative name for the URL provider.

Description

Describes the URL provider for your administrative records.

URL provider settings

Use this page create new URL providers.

To view this administrative console page, click **Resources** > **URL Providers** > *URL_provider*.

Name

Specifies the administrative name for the URL provider.

Description

Describes the URL provider, for your administrative records.

Classpath

Specifies paths or JAR file names which together form the location for the resource provider classes.

Stream Handler Class Name

Specifies fully qualified name of a user-defined Java class that extends `java.net.URLStreamHandler` for a particular URL protocol, such as FTP.

Protocol

Specifies the protocol supported by this stream handler. For example, ″nntp″, ″smtp″, ″ftp″.

URL configuration collection

Use this page to configure Uniform Resource Locators (URLs) that point to electronically accessible resources, such as directory files on a machine in a network, or a document stored in a database.

To view this administrative console page, click **Resources** > **URL Providers** > *URL_provider* > **URLs**.

Name

Specifies the display name for the resource.

JNDI Name

Specifies the JNDI name.

Description

Specifies the description of the resource.

Category

Specifies the category string, which you can use to classify or group the resource.

URL configuration settings

Use this page to configure Uniform Resource Locators (URLs) that point to electronically accessible resources, such as a directory file on a machine in a network, or a document stored in a database.

To view this administrative console page, click **Resources** > **URL Providers** > *URL_provider* > **URLs** > *URL*.

Name

Specifies the display name for the resource.

JNDI Name

Specifies the JNDI name.

Description

Specifies the description of the resource.

Category

Specifies the category string, which you can use to classify or group the resource.

Spec

Specifies the string from which to form a URL.

URLs: Resources for learning

Use the following links to find relevant supplemental information about URLs. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

These links are provided for convenience. Often, the information is not specific to the IBM WebSphere Application Server product, but is useful all or in part for understanding the product. When possible, links are provided to technical papers and Redbooks that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas.

Programming specifications

- W3C Architecture - Naming and Addressing: URIs, URLs
- URL API documentation

Chapter 6. Resource environment entries

1. Resource Environment Provider collection
2. Resource Env Entries collection
3. Referenceables collection

Resource environment providers and resource environment entries

A resource environment reference maps a logical name used by the client application to the physical name of an object.

Not all objects bound into the server JNDI namespace are intended for use by an application client. For example, the WebSphere Application Server client run time does not support the use of Java 2 Connector (J2C) objects on the client. The object needs to be remotable, and the client-side implementations must be made available on the application client run-time classpath.

Resource environment references are different than resource references. Resource environment references allow your application client to use a logical name to look up a resource bound into the server JNDI namespace. A resource reference allows your application to use a logical name to look up a local J2EE resource. The J2EE specification does not specify a particular implementation of a resource.

Resource Environment Provider collection

Use this page to view the resource environment providers.

To view this administrative console page, click **Resources** >**Resource Environment Providers**

Name

Specifies a text identifier for the resource environment provider.

Data type String

Description

Specifies a text string describing the resource environment provider.

Data type String

Resource environment provider settings

Use this page to create settings for a resource environment provider.

To view this administrative console page, click **Resources** >**Resource Environment Providers** > *resource environment provider*

Scope

Specifies the level to which this resource definition is visible -- the cell, node, or server level.

Resources such as JDBC Providers, Namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

Note that no matter what the scope of a defined resource, the resource's properties only apply at an individual server level. For example, if you define the scope of a data source at the Cell level, all users in that Cell can look up and use that data source, which is unique within that Cell. However, resource property settings are local to each server in the Cell. For example, if you define *max connections* to 10, then each server in that Cell can have 10 connections.

Cell The most general scope. Resources defined at the Cell scope are visible from all Nodes and servers, unless they are overridden. To view resources defined in the cell scope, do not specify a server or a node name in the scope selection form.

Node The default scope for most resource types. Resources defined at the Node scope override any duplicates defined at the Cell scope and are visible to all servers on the same node, unless they are overridden at a server scope on that node. To view resources defined in a node scope, do not specify a server, but select a node name in the scope selection form.

Server The most specific scope for defining resources. Resources defined at the Server scope override any duplicate resource definitions defined at the Cell scope or parent Node scope and are visible only to a specific server. To view resources defined in a server scope, specify a server name as well as a node name in the scope selection form.

When resources are created, they are always created into the current scope selected in the panel. To view resources in other scopes, specify a different node or server in the scope selection form.

Data type String

Name
Specifies the name of the resource provider.

Data type String

Description
Specifies a text description for the resource provider.

Data type String

New Resource Environment Provider
Use this page to define the configuration for a library that provides the implementation for some environment resource factory.

To view this administrative console page, click **Resources** >**Resource Environment Providers** > **New**.

Scope
Specifies the level to which this resource definition is visible -- the cell, node, or server level.

Resources such as JDBC Providers, Namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

Note that no matter what the scope of a defined resource, the resource's properties only apply at an individual server level. For example, if you define the scope of a data source at the Cell level, all users in that Cell can look up and use that data source, which is unique within that Cell. However, resource property settings are local to each server in the Cell. For example, if you define *max connections* to 10, then each server in that Cell can have 10 connections.

Cell The most general scope. Resources defined at the Cell scope are visible from all Nodes and servers, unless they are overridden. To view resources defined in the cell scope, do not specify a server or a node name in the scope selection form.

Node The default scope for most resource types. Resources defined at the Node scope override any duplicates defined at the Cell scope and are visible to all servers on the same node, unless they are overridden at a server scope on that node. To view resources defined in a node scope, do not specify a server, but select a node name in the scope selection form.

Server The most specific scope for defining resources. Resources defined at the Server scope override any duplicate resource definitions defined at the Cell scope or parent Node scope and are visible only to a specific server. To view resources defined in a server scope, specify a server name as well as a node name in the scope selection form.

When resources are created, they are always created into the current scope selected in the panel. To view resources in other scopes, specify a different node or server in the scope selection form.

Data type String

Name

Specifies a text identifier for the resource environment provider.

Data type String

Description

Specifies a text string describing the resource environment provider.

Data type String

Resource Env Entries collection

Use this page to view Resource Environment Entries.

An environment resource can be of any arbitrary type. See the EJB 2.0 specification for more information about resource-env-refs and environment resources.

To view this administrative console page, click **Resources** >**Resource Environment Providers** > *resource_environment_provider* > **Resource Env Entries**.

Name

Specifies a text identifier that helps distinguish this resource-env entry from others.

For example, you can use *My Resource* for the name.

Data type String

JNDI Name

Specifies the string to be used when looking up this environment resource using JNDI.

This is the string to which you bind resource-env-ref deployment descriptors.

Data type String

Description

Specifies text for information to help further identify and distinguish this resource

Data type String

Category

Specifies a category you can use to group environment resources according to some common feature.

It is strictly an organizational property and has no effect on the function of the environment resource.

Data type String

Resource env entry settings

Use this page to set resource environment entries, which define configuration for an environment resource that is the binding target for a resource-environment-reference in some application's deployment descriptor.

To view this administrative console page, click **Resources** >**Resource Environment Providers** > *resource_environment_provider* > **Resource Env Entries** > *resource_environment_entry*.

Scope

Specifies the level to which this resource definition is visible -- the cell, node, or server level.

Resources such as JDBC Providers, Namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

Note that no matter what the scope of a defined resource, the resource's properties only apply at an individual server level. For example, if you define the scope of a data source at the Cell level, all users in that Cell can look up and use that data source, which is unique within that Cell. However, resource property settings are local to each server in the Cell. For example, if you define *max connections* to 10, then each server in that Cell can have 10 connections.

Cell The most general scope. Resources defined at the Cell scope are visible

from all Nodes and servers, unless they are overridden. To view resources defined in the cell scope, do not specify a server or a node name in the scope selection form.

Node The default scope for most resource types. Resources defined at the Node scope override any duplicates defined at the Cell scope and are visible to all servers on the same node, unless they are overridden at a server scope on that node. To view resources defined in a node scope, do not specify a server, but select a node name in the scope selection form.

Server The most specific scope for defining resources. Resources defined at the Server scope override any duplicate resource definitions defined at the Cell scope or parent Node scope and are visible only to a specific server. To view resources defined in a server scope, specify a server name as well as a node name in the scope selection form.

When resources are created, they are always created into the current scope selected in the panel. To view resources in other scopes, specify a different node or server in the scope selection form.

Data type String

Name

Specifies a display name for the resource.

Data type String

JNDI name

Specifies the JNDI name for the resource, including any naming subcontexts.

This name is used as the linkage between the platform's binding information for resources defined by a module's deployment descriptor and actual resources bound into JNDI by the platform.

Data type String

Description

Specifies a text description for the resource.

Data type String

Category

Specifies a category string that you can use to classify or group the resource.

Data type String

Referenceables

Specifies the referenceable that holds the factoryClassname of the factory that converts information in the name space into a class instance for the type of resource desired, and for the classname of the type to be returned.

Data type Dropdown menu

Referenceables collection

Use this page to specify the factoryClassname of the factory that will convert information in the name space into a class instance for the type of resource desired.

To view this administrative console page, click **Resources** >**Resource Environment Providers** > *resource_environment_provider* > **Referenceables**.

Factory Classname

Specifies a javax.naming.ObjectFactory implementation class name

Data type String

Classname

Specifies the Java type that a Referenceable provides access to, for binding validation and to create the reference

Data type String

Referenceables settings

Use this page to set the factoryClassname of the factory that converts information in the name space into a class instance for the type of resource desired

To view this administrative console page, click **Resources** >**Resource Environment Providers** > *resource_environment_provider* > **Referenceables** > *referenceable*.

Scope

Specifies the level to which this resource definition is visible -- the cell, node, or server level.

Resources such as JDBC Providers, Namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

Note that no matter what the scope of a defined resource, the resource's properties only apply at an individual server level. For example, if you define the scope of a data source at the Cell level, all users in that Cell can look up and use that data source, which is unique within that Cell. However, resource property settings are local to each server in the Cell. For example, if you define *max connections* to 10, then each server in that Cell can have 10 connections.

Cell The most general scope. Resources defined at the Cell scope are visible from all Nodes and servers, unless they are overridden. To view resources defined in the cell scope, do not specify a server or a node name in the scope selection form.

Node The default scope for most resource types. Resources defined at the Node scope override any duplicates defined at the Cell scope and are visible to all servers on the same node, unless they are overridden at a server scope on that node. To view resources defined in a node scope, do not specify a server, but select a node name in the scope selection form.

Server The most specific scope for defining resources. Resources defined at the Server scope override any duplicate resource definitions defined at the Cell scope or parent Node scope and are visible only to a specific server. To

view resources defined in a server scope, specify a server name as well as a node name in the scope selection form.

When resources are created, they are always created into the current scope selected in the panel. To view resources in other scopes, specify a different node or server in the scope selection form.

Data type String

Factory Classname

Specifies a javax.naming.ObjectFactory implementation class name

Data type String

Classname

Specifies the Java type that a Referenceable provides access to, for binding validation and to create the reference

Data type String

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, New York 10594 USA

Trademarks and service marks

The following terms are trademarks of IBM Corporation in the United States, other countries, or both:

- AIX
- AS/400
- CICS
- Cloudscape
- DB2
- DFSMS
- Domino
- Everyplace
- iSeries
- IBM
- IMS
- Informix
- iSeries
- Language Environment
- Lotus
- MQSeries
- MVS
- OS/390
- RACF
- Redbooks
- RMF
- SecureWay
- SupportPac
- Tivoli
- ViaVoice
- VisualAge
- VTAM
- WebSphere
- z/OS
- zSeries

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.