IBM WebSphere Application Server Enterprise,
Version 5.0.2

J2EE Resources

IBM WebSphere Application Server Enterprise,
Version 5.0.2

J2EE Resources

Compilation date: August 15, 2003

Contents

Chapter 4. Resource environment entries 313

Chapter 5. Using mail 321

Chapter 6. Using URL resources within an application 333

Notices 337

Chapter 1. Welcome to Resources

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

In addition to the standard J2EE resources described in this topic, the Enterprise product adds the following resources in support of enterprise extensions:

- Extended messaging providers
- Schedulers
- Work managers
- Object pools
- Staff plug-in providers

Data access (JDBC and J2C)
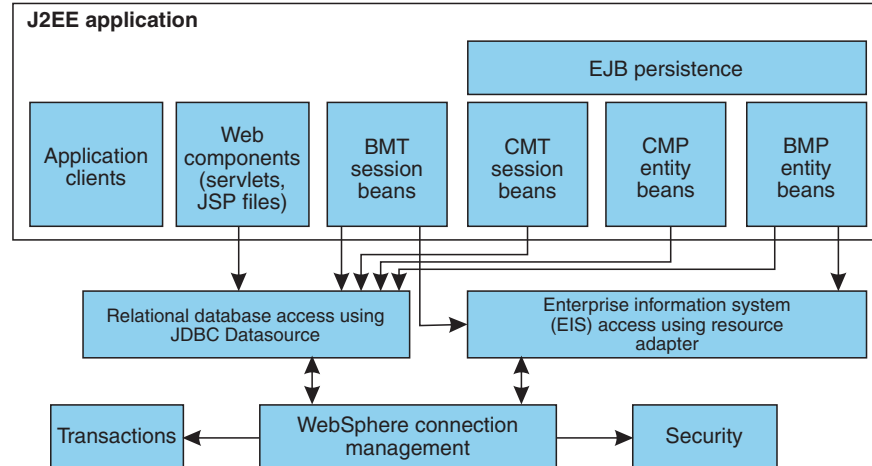
The J2EE Connector architecture defines a standard architecture that enables the integration of various enterprise information systems (EIS) with application servers and enterprise applications. It defines a standard resource adapter used by a Java application to connect to an EIS. This resource adapter 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.

The Enterprise JavaBeans (EJB) 2.0 architecture for container-managed persistence (CMP) provides a separation between the client view of a bean (as presented by its home and remote interfaces) and the entity bean instance (which provides the implementation of the client view). This separation enables you to change an entity bean independently from its clients. It also means that you can redeploy an entity bean across different persistence managers and different persistent data stores, without requiring the redefinition or recompilation of the entity bean class.

The *Bean Provider* concentrates on the business logic of the object and defines the relationship through the abstract persistence schema; whereas the *Persistence Manager* is responsible for providing the implementation of the persistent fields and relationships, as well as all data access to the underlying persistent store. To achieve this portability, the EJB 2.0 data access model is based on the J2EE

Connector Architecture (JCA) specification. This specification is different from the EJB 1.x data access model that used the JDBC Connection Manager (CM) model from Version 4.0.

The EJB 2.0 Persistence Resource Adapter model utilizes the JCA defined *Resource Adapter* to connect to various backend data stores without changing the persistence manager. JDBC applications that access relational databases using the JDBC API indirectly use a WebSphere Application Server Resource Adapter specifically designed to work with JDBC data sources. The product also enables any JCA compliant connector to plug in, enabling the application to access an EIS system.

Users of data sources do not see any differences in the programming model from previous releases because of the underlying use of the JCA architecture. JDBC users still configure and use data sources according to the JDBC programming model. Note that some applications migrating from previous versions of the product can experience behavioral differences because of J2EE Version 1.3 requirements.

WebSphere Application Server 5.0 provides support for the JDBC Connection Manager model from Version 4.0, enabling J2EE 1.2 applications to run unaltered. However, the EJB 2.0 module within a J2EE 1.3 application cannot use the 4.0 JDBC Connection Manager.

For more information, you can read the data access concept articles listed below:
- Resource adapter
- Connection factory
- JDBC providers
- Data sources
- Data access beans
- Connection management

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). An application can explicitly poll for messages on a destination. The product also provides a message listener service that 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.

The Enterprise edition provides *extended messaging*, which uses the EJB container to manage the messaging infrastructure, and provides more types of messaging beans. This enables application developers to concentrate on the business logic for enterprise beans and to leave the messaging usage to messaging objects and configuration of the EJB container.

Tools for working with messaging include the WebSphere Studio for developing and packaging J2EE applications that use JMS, message-driven beans, or extended messaging; the Application Assembly Tool, as described in "Assembling applications" (not in this document); and the product systems administration tools, as described in "Welcome to System Administration" (not in this document).

For more information about implementing enterprise applications that use asynchronous messaging, see the following topics:

- An overview of WebSphere asynchronous messaging, as described in ″Asynchronous messaging with WebSphere - an overview″ (not in this document).
- Implementing WebSphere J2EE applications that use JMS, as described in (″Using JMS and messaging in applications″).
- Implementing WebSphere J2EE applications that use extended messaging, as described in [ ].

Mail

New terminology in Mail technology includes:

- Protocol Provider, a concept equivalent to Service Provider in the JavaMail specification
- Mail Provider, a holder of Protocol Providers

New features in Mail technology include:

- JavaMail specification level changed from 1.1 to 1.2, while JavaBeans Activation Framework (JAF) specification level remains at 1.0
- JavaMail implementation level changed from 1.1.3 to 1.2, and JAF implementation level moved up from 1.0.1 to 1.0.2.
- It is now easier to install and configure custom protocol providers
- You now can define your own custom mail providers, in addition to the built-in mail provider, to easily mix and match various custom and default protocol providers
- There is equal support for mail store protocols as for transport protocols, and equal support for POP3 as for IMAP
- Store-related APIs have been more thoroughly tested. You can define store-only mail sessions if you would like.

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. That is, they can look up references to logically named URL connection factories through the `java:comp/env/url` subcontext that have been declared in the application deployment descriptor and mapped to installation-specific URL resources.

Resource environment entries

A resource environment reference maps a logical name used by the client application to the physical name of an object. For more information, see (″Resource environment entries″).

Chapter 2. 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 JMS support
- Developing a J2EE application to use JMS
- Developing a JMS client
- Deploying a J2EE application to use JMS
- Resolving problems with WebSphere JMS

For more information about JMS, see the JMS documentation at http://java.sun.com/products/jms/docs.html.

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 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). An J2EE application can use JMS queue destinations for point-to-point messaging and JMS topic destinations for publish/subscribe messaging. An J2EE application can

explicitly poll for messages on a destination then retrieve messages for processing by business logic beans (enterprise beans).

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.
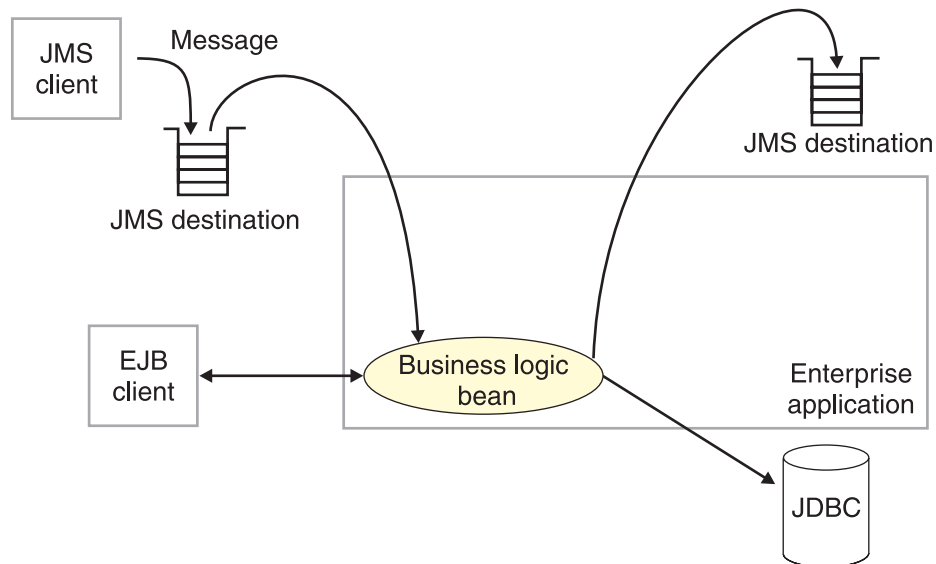
With the base JMS/XA support, the J2EE application uses standard JMS calls to process messages, including any responses or outbound messaging. Responses can be handled by an enterprise bean acting as a sender bean, or handled in the enterprise bean that receives the incoming messages. Optionally, this 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 an 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

With WAS Enterprise, 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:

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.
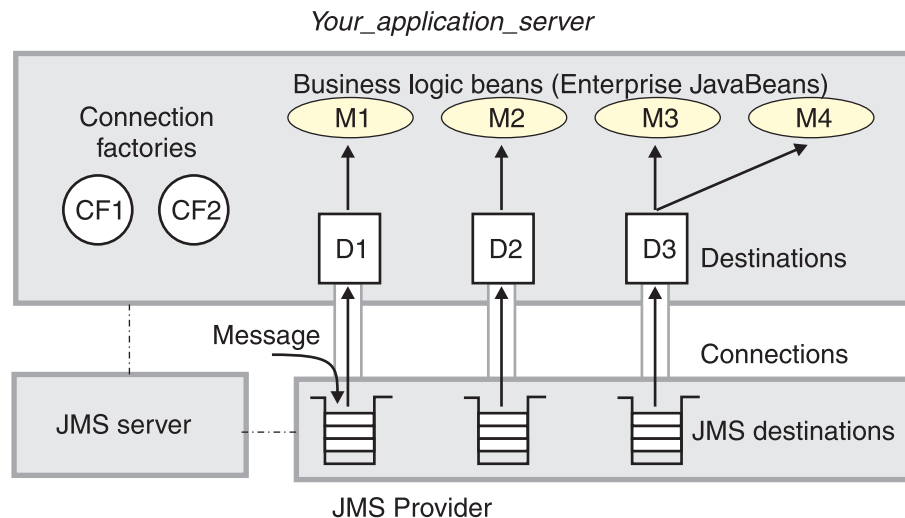
WebSphere Application Server supports asynchronous messaging based on the Java Messaging Service (JMS) of a *JMS provider* that conforms to the JMS specification version 1.0.2 and supports the Application Server Facility (ASF) function defined within that specification. WebSphere Application Server provides 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 "Configuring 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
- "Securing messaging directories and log files"
- Enabling security for the embedded WebSphere JMS provider
- Displaying administrative lists of JMS resources
- Managing WebSphere internal JMS servers
- 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.

- 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 MQ features.
  – You can install IBM WebSphere Application Server with embedded messaging then later install WebSphere MQ for use as a JMS provider.

- 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 Server** option. You are recommended to install and use the WebSphere Application Server **Embedded Messaging Client**.
  – You can install WebSphere MQ for use as a JMS provider on top of WebSphere Application Server embedded messaging; this results in a single JMS installation.
    - 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.
  – 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.

- 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.

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/

Usage scenario

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 IBM 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 IBM 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 be 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.
   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 IBM WebSphere Application Server with the **Embedded Messaging Server** and **Embedded Messaging Client** options.

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 IBM WebSphere Application Server with the **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 IBM 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 IBM WebSphere Application Server with 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.

Before you begin

(UNIX platforms only) Before you install WebSphere MQ, create and mount a journalized file system called /var/mqm for your messaging working data. Use a partition strategy with a separate volume for the WebSphere MQ data. This means that other system activity is not affected if a large amount of messaging work builds up in /var/mqm. You can also create separate file systems for your log data (var/mqm/log) and error files (var/mqm/errors). You should store log files on a different physical volume from the messaging queues (var/mqm). This ensures data integrity in the case of a hardware failure. If you are creating separate file systems, allow a minimum of 30 MB of storage for /var/mqm, 20 MB of storage for /var/mqm/log, and 4 MB of storage for /var/mqm/errors.

- The /var file system is used to store all the security logging information for the system, and is used to store the temporary files for email and printing. Therefore, it is critical that you maintain free space in /var for these operations. If you do not create a separate file system for messaging data, and /var fills up, all security logging will be stopped on the system until some free space is available in /var. Also, email and printing will no longer be possible until some free space is available in /var.
- For more information the security logging issue, see "Integrating IBM WebSphere Application Server security with existing security systems" (not in this document).
- For more information about creating file systems for WebSphere MQ, and WebSphere MQ space requirements for /var file systems, see the section "Preparing for Installation: Creating WebSphere MQ file systems" in the appropriate WebSphere MQ *Quick Beginnings* book.
- For other installation prerequisites, see the appropriate WebSphere MQ *Quick Beginnings* book, as follows:
  - *WebSphere MQ for Windows, V5.3 Quick Beginnings*, GC34-6073
  - *WebSphere MQ for AIX, V5.3 Quick Beginnings*, GC34-6076
  - *WebSphere MQ for Solaris, V5.3 Quick Beginnings*, GC34-6075
  - **5.0.1** *WebSphere MQ for HP-UX, V5.3 Quick Beginnings*, GC34-6077
  - *WebSphere MQ for Linux for Intel and Linux for zSeries, V5.3 Quick Beginnings*, GC34-6078

  You can get these books from the WebSphere MQ messaging platform-specific books Web page at http://www-3.ibm.com/software/ts/mqseries/library/manualsa/manuals/platspecific.html

To install and configure WebSphere MQ (MQSeries) for use as a JMS provider to IBM WebSphere Application Server, complete the following steps:

Steps for this task

1. Install WebSphere MQ 5.3, with the required MQ features, as described in the installation instructions provided with WebSphere MQ.

   These notes apply both if you want to use an existing WebSphere MQ 5.3 installation or if you want to install, or upgrade to, a new WebSphere MQ 5.3 installation.

Note: The WebSphere Application Server Enterprise package includes copies of the WebSphere MQ 5.3 and Event Broker installation packages, with restricted licensing for use with WebSphere Application Server Enterprise. (WebSphere MQ Event Broker is not available for Linux.) For more information about the usage restrictions for the WebSphere MQ 5.3 and Event Broker installation packages, see the WebSphere Application Server Enterprise licensing information.

If you want to use the original WebSphere MQ 5.3 release, ensure that you install the CSD03 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 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".

You can use the WebSphere MQ 5.3 installation package provided with WebSphere Application Server Enterprise to install the required MQ features into an existing WebSphere MQ 5.3 or to install a new WebSphere MQ 5.3 with the required MQ features for use with WebSphere Application Server Enterprise subject to the licensing conditions.

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. **(Optional)** If you want to install IBM WebSphere Application Server on the same host as WebSphere MQ, and have not yet done so, install IBM 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 Server** option. You are recommended to install and use the WebSphere Application Server **Embedded Messaging Client**.

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.

Results

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.

Before you begin

(UNIX platforms only) Restrict access to the messaging errors directories and logging files, by using the following commands:

1. For the /var/mqm/errors directory:

   ```
   chmod 3777 /var/mqm/errors
   chown mqm:mqm /var/mqm/errors

   touch /var/mqm/errors/AMQERR01.LOG
   chown mqm:mqm /var/mqm/errors/AMQERR01.LOG
   chmod 666 /var/mqm/errors/AMQERR01.LOG

   touch /var/mqm/errors/AMQERR02.LOG
   chown mqm:mqm /var/mqm/errors/AMQERR02.LOG
   chmod 666 /var/mqm/errors/AMQERR02.LOG

   touch /var/mqm/errors/AMQERR03.LOG
   chown mqm:mqm /var/mqm/errors/AMQERR03.LOG
   chmod 666 /var/mqm/errors/AMQERR03.LOG
   ```

2. For the /var/mqm/qmgrs/@SYSTEM/errors directory:

   ```
   chmod 3777 /var/mqm/qmgrs/@SYSTEM/errors
   chown mqm:mqm /var/mqm/qmgrs/@SYSTEM/errors

   touch /var/mqm/qmgrs/@SYSTEM/errors/AMQERR01.LOG
   chown mqm:mqm /var/mqm/qmgrs/@SYSTEM/errors/AMQERR01.LOG
   chmod 666 /var/mqm/qmgrs/@SYSTEM/errors/AMQERR01.LOG

   touch /var/mqm/qmgrs/@SYSTEM/errors/AMQERR02.LOG
   chown mqm:mqm /var/mqm/qmgrs/@SYSTEM/errors/AMQERR02.LOG
   chmod 666 /var/mqm/qmgrs/@SYSTEM/errors/AMQERR02.LOG

   touch /var/mqm/qmgrs/@SYSTEM/errors/AMQERR03.LOG
   chown mqm:mqm /var/mqm/qmgrs/@SYSTEM/errors/AMQERR03.LOG
   chmod 666 /var/mqm/qmgrs/@SYSTEM/errors/AMQERR03.LOG
   ```

This is part of the procedure to secure the directories and log files needed for WebSphere MQ, as described in "Securing messaging directories and log files".

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 you begin

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:

Steps for this task

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. **(Optional)** 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.

> **Abstract:**
>
> If you want to use Embedded Messaging, you can install the following options of IBM WebSphere Application Server:
>
> **Embedded Messaging Server**
> > This option installs the messaging server functions of the WebSphere JMS provider. The WebSphere JMS provider supports both queues (for point-to-point messaging) and topics (for publish/subscribe messaging).
>
> **Embedded Messaging Client**
> > This option installs the messaging client functions that enable applications running in WebSphere Application Server to communicate with the WebSphere JMS provider.

Before you begin

Before you install the embedded messaging options of IBM WebSphere Application Server, you must complete the following steps:

1. If you want to install embedded messaging on a machine where you already have WebSphere MQ installed, you must ensure that you have upgraded to WebSphere MQ 5.3 with the required MQ features:

   a. If you have the original WebSphere MQ 5.3 release installed, ensure that you have applied the CSD03 update.

   To determine if your WebSphere MQ 5.3 installation is at the required level, run the **mqver** utility provided by WebSphere MQ. The required level as indicated by mqver is shown below:

   ```
   Name:        WebSphere MQ
      Version:     530.3  CSD03
      CMVC level:  p530-CSD03J
      BuildType:   IKAP - (Production)
   ```
   In comparison, earlier levels of the WebSphere MQ 5.3 release have indicated levels such as p000-L020617 and p000-L021011.
   For more information about CSD03 and other WebSphere MQ product support, see the WebSphere MQ support page for your platform.

   b. Ensure that you have installed 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".

If you have not installed WebSphere MQ 5.3 with the required MQ features, then installation of IBM WebSphere Application Server Embedded Messaging options fails with prerequisite check errors.

The WebSphere Application Server Enterprise package includes copies of the WebSphere MQ 5.3 and Event Broker installation packages, with restricted licensing. (WebSphere MQ Event Broker is not available for Linux.) You can use the provided packages to install the required MQ features or WebSphere MQ 5.3 for use with WebSphere Application Server Enterprise.

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 follows:
- *WebSphere MQ for Windows, V5.3 Quick Beginnings*, GC34-6073
- *WebSphere MQ for AIX, V5.3 Quick Beginnings*, GC34-6076
- *WebSphere MQ for Solaris, V5.3 Quick Beginnings*, GC34-6075
- **5.0.1** *WebSphere MQ for HP-UX, V5.3 Quick Beginnings*, GC34-6077
- *WebSphere MQ for Linux for Intel and Linux for zSeries, V5.3 Quick Beginnings*, GC34-6078

You can get these books from the WebSphere MQ messaging platform-specific books Web page at http://www-3.ibm.com/software/ts/mqseries/library/manualsa/manuals/platspecific.html

2. Ensure that there is enough space in the file systems where you want to install the Embedded Messaging options and store associated messaging data. On UNIX platforms, you also need to create the required file systems before installing the embedded messaging feature.

- **(Windows only)** You can specify the file system into which the Embedded Messaging options are installed. The following table lists the default locations for the base messaging functions and the messaging broker functions (for publish/subscribe messaging). The table also provides figures for the file system sizes on which you can base your own calculations.

Installation directory and space needed for embedded messaging - Windows platform

	Base messaging	Messaging broker
Installation directory	C:\Program Files\IBM\WebSphere MQ	C:\Program Files\IBM\WebSphere MQ\WEMPS
Typical space needed	70 MB (server) or 15 MB (client)	45 MB (server)

If you are using the Installation wizard to install IBM WebSphere Application Server, you can specify an install location for the Embedded Messaging options during either a Full or Custom install. If you want to use a silent install for IBM WebSphere Application Server, you can specify an install location for the Embedded Messaging options when you tailor the options response file before issuing the command to install silently. For more

information about performing a silent installation (including editing the options response file), see "Installing silently" (not in this document).

- **(UNIX platforms only)** The file system into which the Embedded Messaging options are installed is fixed. The following table lists the default locations for the base messaging functions and the messaging broker functions (for publish/subscribe messaging). The table also provides figures for the file system sizes on which you can base your own calculations.

Installation directory and space needed for embedded messaging - UNIX platforms

	Base code	Broker code	Base data	Broker data
	/usr/mqm	/usr/opt/wemps	/var/mqm	/var/wemps
AIX	40MB (server) or 15MB (client)	80MB (server) or 15MB (client)	8MB (server) or 5MB (client)	5MB (server)
	/opt/mqm	/opt/wemps	/var/mqm	/var/wemps
Linux/Intel	40MB (server) or 15MB (client)	100MB (server) or 15MB (client)	8MB (server) or 5MB (client)	5MB (server)
HP-UX	40MB (server) or 15MB (client)	105MB (server) or 15MB (client)	8MB (server) or 5MB (client)	5MB (server)
Solaris	40MB (server) or 15MB (client)	70MB (server) or 15MB (client)	20MB (server) or 15MB (client)	5MB (server)

Before you install WebSphere embedded messaging, create and mount a journalized file system called /var/mqm for your messaging working data. Use a partition strategy with a separate volume for the WebSphere MQ data. This means that other system activity is not affected if a large amount of messaging work builds up in /var/mqm. You can also create separate file systems for your log data (var/mqm/log) and error files (var/mqm/errors). Store log files on a different physical volume from the Embedded Messaging queues (var/mqm). This ensures data integrity in the case of a hardware failure. If you are creating separate file systems, allow a minimum of 30 MB of storage for /var/mqm, 20 MB of storage for /var/mqm/log, and 4 MB of storage for /var/mqm/errors.

Note: The /var file system is used to store all the security logging information for the system, and is used to store the temporary files for email and printing. Therefore, it is critical that you maintain free space in /var for these operations. If you do not create a separate file system for messaging data, and /var fills up, all security logging will be stopped on the system until some free space is available in /var. Also, email and printing will no longer be possible until some free space is available in /var.

You have the same options for creating file systems for Embedded Messaging as you do for WebSphere MQ. For example:

- If you cannot install the Embedded Messaging options in the required file system (for example, if it is too small), you can do one of the following *before* installing the Embedded Messaging options:
  - Create and mount a new file system for the installation directory.
  - Create a new directory anywhere on your machine, and create a symbolic link from the required installation directory to the new directory. For example, on AIX:

```
mkdir /bigdisk/mqm
ln -s /bigdisk/mqm /usr/mqm
```

3. Define the operating system groups and users needed for embedded messaging:

   - **(UNIX platforms only)**
     a. If you have not already done so, create the groups **mqm** and **mqbrkrs**.
     b. Add the users **mqm** and **root** to the **mqm** group.
     c. Add the user **root** to the **mqbrkrs** group.

        Notes:
        1) You are recommended to run the JMS server process under the root user ID. If you run the JMS server process under another user ID, add that user ID to the **mqm** and **mqbrkrs** groups

           For more information about running servers under a non-root user ID, see "Running an Application Server with a non-root user ID and the nodeagent as root" (not in this document).
        2) User IDs longer than 12 characters cannot be used for authentication with the embedded WebSphere JMS provider.

   - **(Windows only)** Define the process user ID with these authorizations:
     – Assign the user ID to the Administrator group.
     – Give the user ID the advanced user right, Act as part of the operating system.
     – Give the user ID the advanced user right, Log on as a service.

     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.

     The IBM WebSphere Application Server installation wizard GUI grants your Windows user ID the advanced user rights, if the user ID belongs to the Administrator group. The silent installation does not. If you create a new user ID on a Windows platform to perform the silent installation, you must restart the system to activate the proper authorizations for the user ID, and to perform a successful silent installation.

   This user ID, the WebSphere Application Server process user ID, is used to start the JMS server (for general JMS support) and the WebSphere Embedded Broker (for WebSphere topic connections).

   Ensure that these operating system security settings are used when you next start IBM WebSphere Application Server; for example, either log off and then on again with the process user ID, or open a new shell in which to start IBM WebSphere Application Server.

   You have the same options for creating user IDs and groups for Embedded Messaging as you do for WebSphere MQ. For more information about creating user IDs and groups for WebSphere MQ, see the section "Preparing for Installation: Setting up the user ID and group" in the appropriate WebSphere MQ *Quick Beginnings* book, as listed above.

4. **(Solaris only)** Several Solaris kernel values are typically too small for the Embedded Messaging options. Starting the internal JMS server or client with insufficient kernel resources produces a First Failure Support Technology (FFST) file in the `/var/mqm/errors` directory.

Before installing Embedded Messaging, review the machine's configuration. To do this type the following command:

```
sysdef -i
```

The kernel values are set in the `/etc/system` file, as shown in the following example.

```
set shmsys:shminfo_shmmax = 4294967295
set shmsys:shminfo_shmseg = 1024
set shmsys:shminfo_shmmni = 1024
set semsys:seminfo_semaem = 16384
set semsys:seminfo_semmni = 1024
set semsys:seminfo_semmap = 1026
set semsys:seminfo_semmns = 16384
set semsys:seminfo_semmsl = 100
set semsys:seminfo_semopm = 100
set semsys:seminfo_semmnu = 2048
set semsys:seminfo_semume = 256
set msgsys:msginfo_msgmap = 1026
set msgsys:msginfo_msgmax = 65535
set rlim_fd_cur=1024
```

You can change kernel values by editing the `/etc/system` file then rebooting the operating system. For more information about setting up the Solaris system, see the Solaris System Administration documentation; for example, the Solaris Tunable Parameters Reference Manual.

Note: Queue managers are generally independent of each other. Therefore system kernel parameters, for example shmmni, semmni, semmns, and semmnu need to allow for the number of queue managers in the system.

5. **5.0.1** **(HP-UX only)** Several HP-UX kernel values are typically too small for the Embedded Messaging options. Starting the internal JMS server or client with insufficient kernel resources produces a First Failure Support Technology (FFST) file in the `/var/mqm/errors` directory.

    Before installing Embedded Messaging, review the machine's configuration and, if needed, set appropriate HP-UX kernel settings.

    You should set new values for the following messaging-related kernel parameters (along with the base set of kernel parameters required by IBM WebSphere Application Server):

Parameter	Value
sema	1
semaem	16384
semmns	16384
semvmx	32767
shmem	1

    This table lists only those messaging-related kernel parameters that either are extra to the base set or need values that are greater than given in the base set.

    For information about how to review and set the base set of kernel parameters for IBM WebSphere Application Server, see the item "Configure HP-UX kernel settings before installing" in Platform-specific tips for installing and migrating (HP-UX platforms).

    Note: Queue managers are generally independent of each other. Therefore system kernel parameters, for example shmmni, semmni, semmns, and semmnu need to allow for the number of queue managers in the system.

To install the Embedded Messaging options of WebSphere Application Server for use as the WebSphere JMS provider, complete the following steps:

<u>Steps for this task</u>

1. Login with the WebSphere Application Server process user ID (defined as part of the prerequisites)

   Note: (UNIX platforms only) Login as root.

2. On a machine where you want to host queues or topics, install IBM WebSphere Application Server with the **Embedded Messaging Server** option.

   If you also want application servers on the host to run messaging applications, install the **Embedded Messaging Client** option.

   Both options are selected by default.

3. On a machine where you want application servers to run messaging applications that use a JMS provider on another host, install IBM WebSphere Application Server with the **Embedded Messaging Client** option.

<u>Results</u>

This task has installed WebSphere Application Server with its embedded messaging as the JMS provider.

You can configure JMS resources to be provided by embedded messaging, by using the WebSphere administrative console to define WebSphere JMS resources.

<u>What to do next</u>

(UNIX platforms only) Restrict access to the messaging errors directories and logging files; for example, by using the following commands:

1. For the /var/mqm/errors directory:

   ```
   chmod 3777 /var/mqm/errors
   chown mqm:mqm /var/mqm/errors

   touch /var/mqm/errors/AMQERR01.LOG
   chown mqm:mqm /var/mqm/errors/AMQERR01.LOG
   chmod 666 /var/mqm/errors/AMQERR01.LOG

   touch /var/mqm/errors/AMQERR02.LOG
   chown mqm:mqm /var/mqm/errors/AMQERR02.LOG
   chmod 666 /var/mqm/errors/AMQERR02.LOG

   touch /var/mqm/errors/AMQERR03.LOG
   chown mqm:mqm /var/mqm/errors/AMQERR03.LOG
   chmod 666 /var/mqm/errors/AMQERR03.LOG
   ```

2. For the /var/mqm/qmgrs/@SYSTEM/errors directory:

   ```
   chmod 3777 /var/mqm/qmgrs/@SYSTEM/errors
   chown mqm:mqm /var/mqm/qmgrs/@SYSTEM/errors

   touch /var/mqm/qmgrs/@SYSTEM/errors/AMQERR01.LOG
   chown mqm:mqm /var/mqm/qmgrs/@SYSTEM/errors/AMQERR01.LOG
   chmod 666 /var/mqm/qmgrs/@SYSTEM/errors/AMQERR01.LOG

   touch /var/mqm/qmgrs/@SYSTEM/errors/AMQERR02.LOG
   chown mqm:mqm /var/mqm/qmgrs/@SYSTEM/errors/AMQERR02.LOG
   chmod 666 /var/mqm/qmgrs/@SYSTEM/errors/AMQERR02.LOG

   touch /var/mqm/qmgrs/@SYSTEM/errors/AMQERR03.LOG
   chown mqm:mqm /var/mqm/qmgrs/@SYSTEM/errors/AMQERR03.LOG
   chmod 666 /var/mqm/qmgrs/@SYSTEM/errors/AMQERR03.LOG
   ```

This is part of the procedure to secure the directories and log files needed for WebSphere embedded messaging, as described in "Securing messaging directories and log files".

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.

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.

Steps for this task

1. For WebSphere MQ point-to-point messaging, install the base WebSphere MQ product.

   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 follows:

   - *WebSphere MQ for Windows, V5.3 Quick Beginnings*, GC34-6073
   - *WebSphere MQ for AIX, V5.3 Quick Beginnings*, GC34-6076
   - *WebSphere MQ for Solaris, V5.3 Quick Beginnings*, GC34-6075
   - *WebSphere MQ for Linux for Intel and Linux for zSeries, V5.3 Quick Beginnings*, GC34-6078

   You can get WebSphere MQ messaging platform-specific books at this Web page: http://www-3.ibm.com/software/ts/mqseries/ library/manualsa/manuals/platspecific.html

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 WebSphere internal JMS servers

Use this task to manage WebSphere internal JMS servers on WebSphere Application Server Network Deployment and WebSphere Application Server Enterprise.

Each JMS server provides the functions of the JMS provider for a node in your administrative domain. 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.

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:

Steps for this task

1. In the navigation pane, select **Servers-> JMS Servers**

   This displays a table of the JMS servers, showing their runtime status.

2. **(Optional)** 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. **(Optional)** 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 the JMS server to be started automatically, 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. **(Optional)** 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:

Steps for this task
1. Display the embedded WebSphere JMS provider.
   a. In the navigation pane, click **Resources-> WebSphere JMS Provider**
   b. In the content pane, click the name of the WebSphere JMS provider that you want to work with.

   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 Connection Factories**

   This displays any existing queue connection factories for the WebSphere JMS provider in the content pane.
3. 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.
4. Specify appropriate properties for the queue connection factory.
5. Click **OK**.
6. To save your configuration, click **Save** on the task bar of the Administrative console window.
7. **(Optional)** 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:

Steps for this task
1. Display the embedded WebSphere JMS provider.
   a. In the navigation pane, click **Resources-> WebSphere JMS Provider**
   b. In the content pane, click the name of the WebSphere JMS provider that you want to work with.

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 Topic Connection Factories**

   This displays any existing topic connection factories for the WebSphere JMS provider in the content pane.

3. 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.

4. Specify appropriate properties for the topic connection factory.

5. Click **OK**.

6. To save your configuration, click **Save** on the task bar of the Administrative console window.

7. **(Optional)** 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 configure the properties of a queue destination for use with the embedded WebSphere JMS provider, use the administrative console to complete the following steps:

Steps for this task

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 WebSphere internal JMS servers.

8. **(Optional)** To have the changed configuration take effect, stop then restart the application server.

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 factory.

To configure the properties of a topic destination factory for use with the embedded WebSphere JMS provider, use the administrative console to complete the following steps:

Steps for this task

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 Topic Destinations**

   This displays any existing topic destinations for the WebSphere JMS provider in the content pane.

3. 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.

4. Specify appropriate properties for the topic destination.

5. Click **OK**.

6. To save your configuration, click **Save** on the task bar of the Administrative console window.

7. **(Optional)** To have the changed configuration take effect, stop then restart the application server.

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:

Steps for this task

1. Display the WebSphere MQ JMS provider.
   a. In the navigation pane, click **Resources-> WebSphere MQ JMS Provider**
   b. In the content pane, click the name of the WebSphere MQ JMS provider that you want to work with.

   This displays in the content pane a table of properties for the WebSphere MQ JMS provider, including links to the types of JMS resources supported by the JMS provider.
2. 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.
3. 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.
4. Specify appropriate properties for the queue connection factory.
5. Click **OK**.
6. To save your configuration, click **Save** on the task bar of the Administrative console window.
7. **(Optional)** 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:

Steps for this task

1. Display the WebSphere MQ JMS provider.
   a. In the navigation pane, click **Resources-> WebSphere MQ JMS Provider**
   b. In the content pane, click the name of the WebSphere MQ JMS provider that you want to work with.

   This displays in the content pane a table of properties for the WebSphere MQ JMS provider, including links to the types of JMS resources supported by the JMS provider.
2. 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.

3. 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.

4. Specify appropriate properties for the topic connection factory.

5. Click **OK**.

6. To save your configuration, click **Save** on the task bar of the Administrative console window.

7. **(Optional)** 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 factory.

To configure the properties of a queue destination factory for use with the WebSphere MQ JMS provider, use the administrative console to complete the following steps:

Steps for this task

1. In the navigation pane, click **Resources-> WebSphere MQ JMS Provider**

   This displays in the content pane a table of properties for the WebSphere MQ JMS provider, including links to the types of JMS resources supported by the JMS provider.

2. 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.

3. 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.

4. Specify appropriate properties for the queue destination.

5. If you have already defined the properties for the queue manager that is to host the queue, the queue connection properties defined to WebSphere MQ are automatically retrieved and displayed. Otherwise, define the queue connection properties.

   Note: To be able to view or change these properties, the WebSphere MQ Queue Manager on which the queue resides must be configured for remote administration and be running.

   More details about these properties are provided in WebSphere MQ config properties for the queue destination.

6. **(Optional)** If needed, change the 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. **(Optional)** To have the changed configuration take effect, stop then restart the application server.

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 factory.

To configure the properties of a topic destination factory for use with the WebSphere MQ JMS provider, use the administrative console to complete the following steps:

Steps for this task
1. In the navigation pane, click **Resources-> WebSphere MQ JMS Provider**

   This displays in the content pane a table of properties for the WebSphere MQ 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 MQ Topic Destinations**

   This displays a table listing any existing topic destinations, with a summary of their properties.
3. 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.
4. Specify appropriate properties for the topic destination.
5. Click **OK**.
6. To save your configuration, click **Save** on the task bar of the Administrative console window.
7. **(Optional)** To have the changed configuration take effect, stop then restart the application server.

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:

Steps for this task
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. **(Optional)** 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:

Steps for this task

1. Display the JMS provider.
   a. In the navigation pane, click **Resources-> Generic JMS Providers**
   b. In the content pane, click the name of the JMS provider that you want to work with.

   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.
2. 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.
3. 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.
4. Specify appropriate properties for the JMS connection factory.
5. Click **OK**.
6. To save your configuration, click **Save** on the task bar of the Administrative console window.
7. **(Optional)** 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:

Steps for this task

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. 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.

3. 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.

4. 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.

5. Specify appropriate properties for the JMS destination.

6. Click **OK**.

7. To save your configuration, click **Save** on the task bar of the Administrative console window.

8. **(Optional)** To have the changed configuration take effect, stop then restart the application server.

Securing messaging directories and log files

Use this task to restrict access to the /var/mqm directories and log files needed for WebSphere embedded messaging or WebSphere MQ as the JMS provider.

You need to set the permissions described in this topic, to reduce the risk of severe security exposures.

Note: The /var file system is used to store all the security logging information for the system, and is used to store the temporary files for email and printing. Therefore, it is critical that you maintain free space in /var for these operations and prevent unauthorized access to the file system. If you do not create a separate file system for messaging data, and /var fills up, all security logging will be stopped on the system until some free space is available in /var. Also, email and printing will no longer be possible until some free space is available in /var.

This procedure involves steps that you complete at different stages of installing and using IBM WebSphere Application Server, as described below. The steps are also described at appropriate points in other tasks, but are collected here for completeness.

This procedure applies only to the ordinary UNIX file system. If your site uses access-control lists, secure the files by using that mechanism. Any site-specific requirements can affect the desired owner, group and corresponding privileges. For example, on AIX, complete the following steps:

Steps for this task

1. Before installing WebSphere embedded messaging or WebSphere MQ, create and mount a journalized file system called /var/mqm.

   Use a partition strategy with a separate volume for the messaging data. This means that other system activity is not affected if a large amount of messaging work builds up in /var/mqm.

2. Install WebSphere embedded messaging or WebSphere MQ as the JMS provider.

  This stage is described in the following topics:

  - Installing WebSphere embedded messaging as the JMS provider
  - Installing WebSphere MQ as the JMS provider

  As part of this stage, the installation program creates the /var/mqm/errors and /var/mqm/qmgrs/@SYSTEM/errors directories used to hold messaging logging files.

3. Restrict access to the /var/mqm/errors directory and the logging files, by using the following commands:

```
chmod 3777 /var/mqm/errors
chown mqm:mqm /var/mqm/errors

touch /var/mqm/errors/AMQERR01.LOG
chown mqm:mqm /var/mqm/errors/AMQERR01.LOG
chmod 666 /var/mqm/errors/AMQERR01.LOG

touch /var/mqm/errors/AMQERR02.LOG
chown mqm:mqm /var/mqm/errors/AMQERR02.LOG
chmod 666 /var/mqm/errors/AMQERR02.LOG

touch /var/mqm/errors/AMQERR03.LOG
chown mqm:mqm /var/mqm/errors/AMQERR03.LOG
chmod 666 /var/mqm/errors/AMQERR03.LOG
```

4. Restrict access to the /var/mqm/qmgrs/@SYSTEM/errors directory and the logging files, by using the following commands:

```
chmod 3777 /var/mqm/qmgrs/@SYSTEM/errors
chown mqm:mqm /var/mqm/qmgrs/@SYSTEM/errors

touch /var/mqm/qmgrs/@SYSTEM/errors/AMQERR01.LOG
chown mqm:mqm /var/mqm/qmgrs/@SYSTEM/errors/AMQERR01.LOG
chmod 666 /var/mqm/qmgrs/@SYSTEM/errors/AMQERR01.LOG

touch /var/mqm/qmgrs/@SYSTEM/errors/AMQERR02.LOG
chown mqm:mqm /var/mqm/qmgrs/@SYSTEM/errors/AMQERR02.LOG
chmod 666 /var/mqm/qmgrs/@SYSTEM/errors/AMQERR02.LOG

touch /var/mqm/qmgrs/@SYSTEM/errors/AMQERR03.LOG
chown mqm:mqm /var/mqm/qmgrs/@SYSTEM/errors/AMQERR03.LOG
chmod 666 /var/mqm/qmgrs/@SYSTEM/errors/AMQERR03.LOG
```

5. For each application server that uses WebSphere embedded messaging or WebSphere MQ, restrict access to the server's /var/mqm/qmgrs/*long_server_name*/errors directory and its messaging logging files.

  You should restrict access to the server's directory and logging files, as soon after creating the application server as possible.

  To restrict access to the server's directory and logging files, use the following commands:

```
chmod 3775 /var/mqm/qmgrs/long_server_name/errors
chown mqm:mqm /var/mqm/qmgrs/long_server_name/errors

touch /var/mqm/qmgrs/long_server_name/errors/AMQERR01.LOG
chown mqm:mqm /var/mqm/qmgrs/long_server_name/errors/AMQERR01.LOG
chmod 666 /var/mqm/qmgrs/long_server_name/errors/AMQERR01.LOG

touch /var/mqm/qmgrs/long_server_name/errors/AMQERR02.LOG
chown mqm:mqm /var/mqm/qmgrs/long_server_name/errors/AMQERR02.LOG
chmod 666 /var/mqm/qmgrs/long_server_name/errors/AMQERR02.LOG
```

```
touch /var/mqm/qmgrs/long_server_name/errors/AMQERR03.LOG
chown mqm:mqm /var/mqm/qmgrs/long_server_name/errors/AMQERR03.LOG
chmod 666 /var/mqm/qmgrs/long_server_name/errors/AMQERR03.LOG
```
Where *long_server_name* is the long name assigned to the server, in the
following form: WAS_nodename_server_name. For example, if you created an
application server called server1 to run on the node called appnode1, the long
server name would be: WAS_appnode1_server1.

Results

This task has restricted access to the /var/mqm directories and log files needed for
WebSphere embedded messaging or WebSphere MQ as the JMS provider, such that
only the user ID mqm or members of the mqm user group have write access.

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 "Configuring global security" (not in this
document).

Steps for this task

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_install_root\config\cells\your_cell_name\integral-jms-
   authorizations.xml file.

   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.
   * Pub, sub, 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. Please note the file is
   in Unicode, which requires a binary FTP to the host from a workstation.

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.

Results

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.

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 `was_install\config\cells\your_cell_name\integral-jms-authorisations.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.

Network Deployment and Enterprise - 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
Resources-> WebSphere JMS Providers-> (In content pane) *WebSphereJMSProvider*	WebSphere JMS provider	List properties and resources of the selected WebSphere JMS provider
Resources-> WebSphere JMS Providers-> (In content pane) *WebSphereJMSProvider*-> (Under Additional Properties) WebSphere JMS Queue Connection Factories	WebSphere queue connection factories	List all queue connection factories of the selected WebSphere JMS provider
Resources-> WebSphere JMS Providers-> (In content pane) *WebSphereJMSProvider*-> (Under Additional Properties) WebSphere Topic Connection Factories	WebSphere topic connection factories	List all topic connection factories of the selected WebSphere JMS provider
Resources-> WebSphere JMS Providers-> (In content pane) *WebSphereJMSProvider*-> (Under Additional Properties) WebSphere Queue Destinations	WebSphere queue destinations	List all queue destinations of the selected WebSphere JMS provider
Resources-> WebSphere JMS Providers-> (In content pane) *WebSphereJMSProvider*-> (Under Additional Properties) WebSphere Topic Destinations	WebSphere topic destinations	List all topic destinations of the selected WebSphere JMS provider
WebSphere MQ JMS provider		
Resources-> WebSphere MQ JMS Providers	WebSphere MQ JMS providers	List properties and resource types for the WebSphere MQ JMS provider
Resources-> WebSphere MQ JMS Providers-> (In content pane) *MQJMSProvider*	WebSphere MQ JMS provider	List properties and resources of the selected WebSphere MQ JMS provider
Resources-> WebSphere MQ JMS Providers-> (In content pane) *MQJMSProvider*-> (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

Path	Panel	Description
Resources-> WebSphere MQ JMS providers-> (In content pane) *MQJMSProvider*-> (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
Resources-> WebSphere MQ JMS Providers-> (In content pane) *MQJMSProvider*-> (Under Additional Properties) WebSphere MQ Queue destinations	WebSphere MQ queue destination	List all queue destinations of the selected WebSphere MQ JMS provider
Resources-> WebSphere MQ JMS providers-> (In content pane) *MQJMSProvider*-> WebSphere MQ Topic destinations	WebSphere MQ topic destination	List all topic destinations of the selected WebSphere MQ JMS provider
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*
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*
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 "Administrative console buttons" (not in this document) provided.

WebSphere internal JMS server settings

The JMS functions on a node within the WebSphere Application Server administration domain are served by the JMS server on that node. Use this panel to view or change the configuration properties of the selected JMS server.

You can use this panel 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.

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, 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.

Name: The name by which the JMS server is known for administrative purposes within IBM WebSphere Application Server.

This name should not be changed.

Data type	String
Units	Not applicable
Default	WebSphere Internal JMS Server
Range	Not applicable

Description: A description of the JMS server, for administrative purposes within IBM WebSphere Application Server.

This string should not be changed.

Data type	String
Default	WebSphere Internal JMS Server

Number of threads: The number of concurrent threads to be used by the publish/subscribe matching engine

The number of concurrent threads should only be set to a small number.

Data type	Integer
Units	Threads
Default	1
Range	Greater than or equal to 1.

Queue Names: The names of the queues hosted by this JMS server. Each queue name must be added on a separate line.

Each queue listed in this field must have a separate queue administrative object with the same administrative name. To make a queue available to applications,

define a WebSphere queue and add its name to this field on the JMS Server panel for the host on which you want the queue to be hosted.

Data type	ASCII
Units	Queue name
Default	Not applicable
Range	Each entry in this field is a queue name of up to 45 characters, which must match exactly (including use of upper- and lowercase characters) the WebSphere queue administrative object defined for the queue.

Initial state: The state that you want the JMS server to have when it is next restarted.

Data type	Enum
Units	Not applicable
Default	Started
Range	
	Started The JMS server is started automatically.
	Stopped The JMS server is not started automatically. If any deployed enterprise applications are to use JMS server functions provided by the JMS server, the system administrator must start the JMS server manually or select the Started value of this property then restart the JMS server.
	To restart the JMS server, stop then restart that JMS server.

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 "Administrative console buttons" (not in this document) 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**

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.

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

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.

Data type	String
Default	$MQJMS_LIB_ROOT
Range	1 through 256 ASCII characters

Native Library Path: The native library path for the JMS provider.

Data type	String
Default	$MQJMS_LIB_ROOT
Range	1 through 256 ASCII characters

You must set the Native Library Path property to the library directory where the WebSphere MQ Java feature is installed; for example: `c:\WebSphereMQ\Java\lib`

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 WebSphere MQ JMS provider, for administrative purposes

Data type	String

Default	WebSphere MQ JMS provider

Classpath: The Java classpath for the JMS provider.

Data type	String
Default	$MQJMS_LIB_ROOT

Native Library Path: The native library path for the JMS provider.

Data type	String
Default	$MQJMS_LIB_ROOT

You must set the Native Library Path property to the library directory where the WebSphere MQ Java feature is installed; for example: `c:\WebSphereMQ\Java\lib`

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. In the content pane, select the JMS provider that you want to support the connection factory.
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 "Administrative console buttons" (not in this document) 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.

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 "Java Authentication and Authorization service configuration entry settings" (not in this document).

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. In WebSphere Application Server Enterprise 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. 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.

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. In the content pane, select the JMS provider that you want to support the connection factory.

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 "Administrative console buttons" (not in this document) 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 Java "Authentication and Authorization service configuration entry settings" (not in this document).

Data type	Enum

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 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. In WebSphere Application Server Enterprise 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. 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.

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. In the content pane, select the JMS provider that you want to support the queue destination.
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 "Administrative console buttons" (not in this document) 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.

Note: You must add the queue name to the list of queue names in the configuration of the JMS servers that are to host the queue.

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.

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

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. In the content pane, select the JMS provider that you want to support the topic destination.
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 "Administrative console buttons" (not in this document) 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	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/wsmqebv21.html

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

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.

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. In the content pane, select the JMS provider that you want to support the connection factory.
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 "Administrative console buttons" (not in this document) 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.

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*

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 a "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.

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 a "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.

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, nondurable, nontransactional 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

Data type	Enum
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 nontransactional, nondurable and nonpersistent Publish/Subscribe messasging. DIRECT is only works for clients and message-driven beans using the non-ASF protocol.

QUEUED

> QUEUED is a standard TCP protocol.

Recommended　**Queue connection factory transport type**

> BINDINGS is faster by 30% or more, but it lacks security. When you have security concerns, BINDINGS is more desirable than CLIENT.

Topic connection factory transport type

> DIRECT is the fastest 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.

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 tick 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 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. In WebSphere Application Server Enterprise 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. 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.

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. In the content pane, select the JMS provider that you want to support the connection factory.
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 "Administrative console buttons" (not in this document) 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.

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*

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, nondurable, nontransactional 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

Data type	Enum
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 nontransactional, nondurable and nonpersistent Publish/Subscribe messasging. DIRECT is only works for clients and message-driven beans using the non-ASF protocol.

QUEUED

    QUEUED is a standard TCP protocol.

Recommended

Queue connection factory transport type

    BINDINGS is faster by 30% or more, but it lacks security. When you have security concerns, BINDINGS is more desirable than CLIENT.

Topic connection factory transport type

    DIRECT is the fastest 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 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.

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 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. In WebSphere Application Server Enterprise 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. 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.

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. In the content pane, select the JMS provider that you want to support the queue destination.
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 "Administrative console buttons" (not in this document) 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*

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

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:
- Some properties displayed are read-only and cannot be changed.
- 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 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:
	• 999 999 999 if the queue is on OS/390
	• 640 000 if the queue is on any other 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 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, expand **Resources-> WebSphere MQ JMS provider**.
2. In the content pane, select the JMS provider that you want to support the topic destination.
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 "Administrative console buttons" (not in this document) 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/wsmqebv21.html

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 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. In the content pane, select the JMS provider that you want to support the JMS connection factory.
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 "Administrative console buttons" (not in this document) 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*

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.

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. In WebSphere Application Server Enterprise 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. 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.

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. In the content pane, select the JMS provider that you want to support the JMS destination.
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 "Administrative console buttons" (not in this document) 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.

Notes:

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, for authentication with the embedded WebSphere JMS provider, ensure that user IDs no longer than 12 characters are configured in J2C authentication data entries used or are specified by any applications that pass user IDs for authentication.

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 a "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 authorization security for the embedded WebSphere JMS provider".

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

   For more information about using these JMS interfaces, see the Java Message Service Documentation and the WebSphere MQ *Using Java* book, SC34-5456. 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. This 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 "Using message-driven beans in applications" (not in this document).

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, any method of a J2EE application completes the following general
steps:

Steps for this task

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);
            }
        }
    ```

What to do next

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 in the Application Assembly Tool.
- 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 "Developing J2EE application client code" (not in this document) and related tasks.

To use JMS, a typical JMS client program completes the following general steps:

Steps for this task

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:

Steps for this task

1. Configuring the application for deployment.

   Use the the application assembly tool to configure the deployment attributes for the application, as described in "Assembling applications" (not in this document).

2. Use the WebSphere administrative console to install the application.

   This stage is a standard WebSphere Application Server task, as described in "Installing a new application" (not in this document).

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.

Steps for this task

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 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 messages in the application server's SystemOut log at *was_home*\logs*server*\SystemOut.

Check the SystemOut log for error messages for messages related to WebSphere Messaging; look for the prefixes MSGS and WMSG. The associated "Message reference" (not in this document) information provides an explanation and any user actions to resolve the problem.

4. Check for more messages in the application server's SystemOut log.

   If the JMS server is running, but you have problems accessing JMS resources, check the SystemOut log file, which should contain more error messages and extra details about the problem.

   For messages related to WebSphere Messaging, look for the prefixes: MSGS and WMSG.

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. **(Optional)** 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 when starting the WebSphere administrative server
- 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

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.

Also, when installing WebSphere Application Server, the JMS server is not configured to start automatically by default, except on Windows if the messaging samples are installed. So, generally, if you want to use JMS resources on that host, use the administrative console to configure the JMS server to start automatically. For more information about managing JMS servers, see "Administering WebSphere JMS support".

Finally, check that the JMS server has started before trying to use WebSphere Messaging. For SystemOut.log messages that indicate the JMS server has started successfully, see the following tip The JMS server tries to start, but fails.

An MDB listener fails to start

To see if the JMS Server has started, check the *was_home*\logs*server*\SystemOut log. If the JMS server started successfully, you should see the following lines:

```
...
[date time] 58217585 JMSProvider
              A MSGS0050I: Starting the Queue Manager
[date time] 58217585 JMSProvider
              A MSGS0051I: Queue Manager open for business
[date time] 58217585 JMSProvider
              A MSGS0052I: Starting the Broker
[date time] 58217585 JMSProvider
              A 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:

```
...
[date time]
 58228abf QueueManagerM E 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 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 authentication 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. For example, the default Windows NT user ID, **administrator**, is not valid for use with WebSphere embedded messaging, because it contains 13 characters.

  (UNIX platforms only) Check that this user ID is a member of the mqm and mqbrkrs groups.

If WebSphere Application Server fails to create a queue connection, the SystemOut.log contains messages like the following: (split for publication)

```
[date time]
 577cb554 PoolManager  E  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 that 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. You can also check the WebSphere MQ Support service summary Web page.

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: (split for publication)

```
...
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 a "MQJMS2013 invalid security authentication supplied for MQQueueManager" error.

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 configuring the authentication mechanism, see (Data source settings).

Chapter 3. Accessing data from applications

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.
- **5.0.1** 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.

Steps for this task

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

   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.

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.
* (split for publication)
*/
public class ExampleDataStoreHelper
    extends com.ibm.websphere.rsadapter.GenericDataStoreHelper
{
    static final long serialVersionUID = 87889310901499088285L;


    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);
    }


    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;
                  }
              }
            }


        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.

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. 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:

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

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.

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. Use this name, along with its configuration information, 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 value set for the database connection pool must be at least one higher, one of the waiting threads to complete its second database connection and free up to allow database connections.

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 stderr.log file.

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.
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.

Condition	Description
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
  - 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.

Access to a resource marked as **Unshareable** means that there is a 1-to-1 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*, *res-auth*, and *res-isolation-level* help determine if it is possible to share a connection. The res-isolation-level is a WebSphere extension.
- 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.

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. This usage pattern is where an application, within a single method and without calling another method that might get a connection from the same data source or connection factory:

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. 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. 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.

WebSphere Application Server Enterprise provides *last participant support* that enables a single one phase commit resource to participate in a global transaction with one or more two phase commit resources.

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.

Cache instances

An application uses a cache instance to store, retrieve, and share data objects within WebSphere Application Server dynamic cache.

Each cache instance can be configured independently for Java Naming and Directory Interface (JNDI) name, cache size, priority, and disk offload. Objects stored in a particular cache instance are not affected by other cache instances. This means if you store an object named **object_1** with a value of `object_data` in cache instance x, you can also store an object with the same name, but different value in cache instance y.

In the WebSphere Application Server Enterprise environment, objects stored in a particular cache instance are available to applications on other servers by accessing a cache instance of the same name. The two servers must be within the same replication domain to share data.

The programming interface to access a cache instance is DistributedMap. You can locate DistributedMap in the `com.ibm.websphere.cache` Java package. See the DistributedMap Javadoc(../javadoc/ee/com/ibm/websphere/cache/DistributedMap.html) for more information.

Using cache instances

To use cache instances you must complete the following:

Steps for this task

1. Enable the cache instance service.
   a. Using the Administrative Console, select the application server you will use with cache instances and click **Cache Instance Service**.
   b. Click **Start**.

   The cache instance service will start when the server is started.

2. Configure one or more cache instances.
    a. Using the Administrative Console, click **Resources > Cache Instances**.
    b. Specify the scope for the cache instance.

       Cell scope will make the cache instance available to all servers within the cell. Node scope will make the cache instance available to all servers on the particular node. Server scope will make the cache instance available to only the selected server. You can mix scopes if necessary.
    c. Click **Apply** after changing the scope.
    d. Click **New**.
    e. Enter the Java Naming and Directory Interface (JNDI) name for this cache instance.

       This is name you will pass to the InitialContext lookup() method from within your application. For example, `services/cache/instance_one`.
    f. Enter or modify other properties as needed.
3. Update your application.

   To store and retrieve objects in a cache instance, you need a DistributedMap reference for the named cache instance. If you created a cache instance with a JNDI name of `services/cache/instance_one`, adding the following code to your application will return a DistributedMap reference for that cache instance:

   ```
   InitialContext ic = new InitialContext();
   DistributedMap dm1 = (DistributedMap)ic.lookup("services/cache/instance_one");
   ```

   See the DistributedMap Javadoc(../javadoc/ee/com/ibm/websphere/cache/DistributedMap.html) for more information.

Developing data access applications

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.

Steps for this task

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.

5.0.1 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 DB2 is your backend database.

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.

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 concurency");
  }

  // determine correct isolation level for
  // currently conigured 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 Application Assembly Tool (AAT) for more details.

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 Application Assembly Tool (AAT) 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) javax.rmi.PortableRemoteObject.narrow(
      (new InitialContext()).lookup("java:comp/env/jdbc/Section"),
                                 javax.sql.DataSource.class);
```

In the AAT, 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 Application Assembly Tool (AAT).

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.

Steps for this task
1. Start the Application Assembly Tool. Cancel the *Welcome to the Application Assembly Tool* window.
2. Click **File > Open**. Find the EAR file you want to change, click the file name, and click **Open**.
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. Click **Resource References**.
5. Right-click **Resource References** and click **New** to display the new Resource Reference window.
6. Specify settings.
7. Click **IBM Extensions** and select the *Isolation level*.
8. Click **Bindings** and fill in the *JNDI name*. You can choose to skip this step, as you are allowed to fill in or change this field during installation of the EAR file.
9. Click **OK**.
10. Save the EAR file. See "Saving applications after assembly" (not in this document)
11. Generate the code. See "Generating code for deployment" (not in this document)

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 Application Assembly Tool (AAT) 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"))
(javax.sql.DataSource)javax.rmi.PortableRemoteObject.narrow(
   (new IntialContext()).lookup("jdbc/Section"), javax.sql.DataSource.class);
```

In the AAT, 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 JNDI name of the DataSource during assembly using the AAT, this is the setting in the Bean Binding panel. In this example, the JNDI name *jdbc/Section* is used.

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.

5.0.2 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.

Predefined access intent policies

Seven predefined access intent policies are available. The policies are composed of different attributes. You can directly configure most of these attributes only with WebSphere Application Server Enterprise. The *access type* is of primary interest and controls the *isolation level*, lock type, and duration of locks obtained when bean data is read from the database.

A *pessimistic* access type indicates to hold locks for the duration of the transaction under which the data loads. An *optimistic* type indicates to drop locks immediately after the data is read from the backend. A *read* type indicates that the run time must not allow updates to the data; any attempt to do so on data read under a *read*

type results in an exception. *Update* types permit you to change data. The seven policies and their attribute definitions follow:

wsPessimisticUpdate

- Access type = Pessimistic update
- Collection scope = Transaction
- Collection increment = 1
- Resource manager prefetch increment = 0
- Read ahead hint = null

wsOptimisticUpdate

- Access type = Optimistic update
- Collection scope = Transaction
- Collection increment = 25
- Resource manager prefetch increment = 0
- Read ahead hint = null

wsOptimisticRead

- Access type = Optimistic read
- Collection scope = Transaction
- Collection increment = 25
- Resource manager prefetch increment = 0
- Read ahead hint = null

wsPessimisticRead

- Access type = Pessimistic read
- Collection scope = Transaction
- Collection increment = 25
- Resource manager prefetch increment = 0
- Read ahead hint = null

wsPessimisticUpdate-Exclusive

- Access type = Pessimistic update
- Exclusive = true
- Collection scope = Transaction
- Collection increment = 1
- Resource manager prefetch increment = 0
- Read ahead hint = null

wsPessimisticUpdate-NoCollision

- Access type = Pessimistic update
- No collision = true
- Collection scope = Transaction
- Collection increment = 25
- Resource manager prefetch increment = 0
- Read ahead hint = null

wsPessimisticUpdateWeakestLockAtLoad

- ***default policy**
- Access type = Pessimistic Update
- Promote = true

- Collection scope = transaction
- Collection increment = 25
- Resource manager prefetch increment = 0
- Read ahead hint = null

Note that to support connection sharing, you must ensure that all data loaded in the same transaction is under the same isolation level. Verify that all participating methods that drive loads are configured with either a pessimistic access type or an optimistic access type.

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
	DB2	Oracle*	SyBase	Informix	Cloudscape	SQL Server	
wsPessimisticUpdate-Weakest LockAtLoad (Default policy)	RR	RC	RR	RR	RR	RR	No (*Oracle, Yes)
wsPessimisticUpdate	RR	RC	RR	RR	RR	RR	Yes
wsPessimisticRead	RR	RC	RR	RR	RR	RR	No
wsOptimisticUpdate	RC	RC	RC	RC	RC	RC	No
wsOptimisticRead	RC	RC	RC	RC	RC	RC	No
wsPessimisticUpdateNo-Collisions	RC	RC	RC	RC	RC	RC	No
wsPessimisticUpdate-Exclusive	S	RC	S	S	S	S	Yes

- RC = JDBC Read committed
- RR = JDBC Repeatable read
- S = JDBC Serializable

- **5.0.2** * 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.

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:

<u>Steps for this task</u>

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.

<u>Usage scenario</u>

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.

5.0.2 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

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.

Steps for this task

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));
   }
   ```

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();
   ```

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) javax.rmi.PortableRemoteObject.narrow(
 ic.lookup(jndiString),
 javax.sql.DataSource.class);

 // 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 {
   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)
    javax.rmi.PortableRemoteObject.narrow(
      (new InitialContext()).lookup("java:comp/env/eis/Sample"),
javax.resource.cci.ConnectionFactory.class);
   }
   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.

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.

StaleConnectionException on Linux systems

The StaleConnectionException **SQL1224** is related to extension shared memory attachment. Linux systems have a semaphore problem causing the DB2 SQL1224 error.

To work around the problem, set the loopback for your database. For example, if your database is *WAS*, host name is *LHOST*, and database service port number is *50000*, issue the following commands from the DB2 command line window:

```
db2 catalog TCPIP node RHOST remote LHOST server 50000
db2 catalog WAS as WASAlias
db2 uncatalog db WAS
db2 catalog WASAlias as WAS at node RHOST
```

Verify this by issuing the following commands from the DB2 command line window:

```
db2 connect to WAS user xxx
passwd: xxx
```

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 {
 // Note the new Initial Context Factory interface
 // available in WebSphere 4.0
            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    (split for publication)   *
// ***********************************************************
 public void doGet(HttpServletRequest req,
        HttpServletResponse res)
        throws ServletException, IOException
    {
        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));
                }
//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());

//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 (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());
                  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());
                  }
                  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());

   //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());
                              }
                              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>");
                  }
                  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.
//* (split for publication)
//******************************************************************************
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.
//* (split for publication)
//****************************************************************************

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

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());
 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());
   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 {
    // Note the new Initial Context Factory interface available
    // in WebSphere 4.0
        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.
//    (split for publication)
//==================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;
}
//==================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.
//
// (split for publication)
//==================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;
}
//==================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.
//
// (split for publication)
//==================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.
//
//  (split for publication)
//===================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.
//* (split for publication)
//*****************************************************************************
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.
//* (split for publication)        *
//******************************************************************************

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) + ", " +
                            rs.getString(1) + " " + rs.getString(2));
                  }
//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());
            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());
          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 {
// Note the new Initial Context Factory interface
//  available in WebSphere 4.0
        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.
//
//  (split for publication)
//==================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;
}
//==================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.
//
//  (split for publication)
//==================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.
//
//    (split for publication)
//==================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 {

   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 (?,?,?,?,?)";

   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());
                    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
        {
         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 {}
/**
 * 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());
      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());
          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());
            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 {
// Note the new Initial Context Factory interface
// available in WebSphere 4.0
  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());
            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.
//
// (split for publication)
//==================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
 * (split for publication)
 */
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;
/**
 * 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;
}

//==================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.
//
//   (split for publication)
//==================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();
```

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);
    ...
  }
}
```

Embedded Structured Query language in Java (SQLJ) support

Structured Query Language in Java (SQLJ) is a set of programming extensions that allow a programmer, using the Java programming language, to embed statements that provide SQL (Structured Query Language) database requests.

Advantages of developing applications with SQLJ include improved performance and a shorter, more efficient development cycle:

- You improve performance by using static SQL statements.
- You reduce the development cycle because:
  - You write less code with the simpler SQLJ syntax, which reduces the number of lines of code required to execute statements, and to set and retrieve parameters.
  - You detect programming errors earlier in the development phase with the onlinecheck function, which performs data type and schema validation.

You might consider using SQLJ in situations where dynamic SQL is not needed, and where applications use DB2 as the backend database.

Using embedded Structured Query Language in Java (SQLJ) support

WebSphere Application Server Version 5.0.1 introduces support for Structured Query Language in Java (SQLJ) with the DB2 universal Java Database Connectivity (JDBC) driver provider and the DB2 Legacy CLI-based Type 2 JDBC driver for DB2 Version 8 FixPak1 (FP1).

Only the WebSphere Application Server Version 5.0.1 data source supports the DB2 universal JDBC driver provider. The WebSphere Application Server Version 4.0.x data source does not support this driver provider. In addition, the universal JDBC driver currently only supports a single phase data source.

Perform the following steps to convert existing JDBC applications to SQLJ in WebSphere Application Server:

Steps for this task

1. Acquire the required drivers to run SQLJ.
   - DB2 Legacy CLI-based JDBC driver

     You must have the db2java.zip and db2jcc.jar files in your classpath. Define these files in the classpath of the Websphere Application Server Version 5.0.1 DB2 Legacy CLI-based JDBC Driver Provider.
   - DB2 Universal JDBC driver

     You only need the db2jcc.jar file.
   - WebSphere Application Server driver

     WebSphere Application Server must be at the Version 5.0.1 level.
2. Review the appropriate task file for instructions on using SQLJ with either Enterprise Java Bean (EJB) Container Managed Persistence (CMP) beans, or with EJB Bean Managed Persistence (BMP) entity beans, session beans, and servlets.

Using Structured Query Language in Java (SQLJ) for Enterprise Java Bean (EJB) Container Managed Persistence (CMP) beans

Structured Query Language in Java (SQLJ) for EJB Persistence requires DB2 Version 8 FixPak1. You need DB2 UDB Application Development Client Version 8 FixPak1

on the machine where you generate the deployed code, and DB2 UDB Server (Version 8 FixPak1 for UNIX and Windows, and Version 6 or Version 7 for z/OS and OS/390) on the machine containing the database for running the scenario.

SQLJ support requires Version 2.0 CMP Enterprise Java Beans. If you mix EJB 1.1 and 2.0 beans in an EJB 2.0 module, then only the EJB 2.0 beans are deployed to use SQLJ. The EJB 1.1 beans continue using JDBC for data access.

The deployment command for Enterprise Java Beans (EJBDeploy) only provides SQLJ support for the Version 2.0 CMP beans. It is the user's responsibility to manually translate and customize the profile on the database for the Bean Managed Persistence (BMP) entity beans, session beans, and servlet SQLJ applications. See the Using SQLJ for Enterprise Java Bean (EJB) Bean Managed Persistence (BMP) entity beans, session beans, and Servlets article for more information.

WebSphere Application Server Version 5.0.1 provides tools for using SQLJ as the persistence mechanism for CMP Enterprise Java Beans. You can deploy the CMP beans to use SQLJ either by using the EJB deploy tool in WebSphere Studio Application Developer (WSAD), or by using the command utility **ejbdeploy** with the **-sqlj** option.

Note: You cannot deploy a CMP bean with the **sqlj** option from the administrative console or the Application Assembly Tool (AAT) in Version 5.01.

When you install an application comprised of CMP beans, you have the following choices:
- For data access with SQLJ, deploy the CMP beans to use SQLJ before installing the application in WebSphere Application Server. Perform deployment within the WSAD environment, or by using the command utility **ejbdeploy** with the **-sqlj** option.

  Do not deploy this bean again in the administrative console when you install the application. If you pre-deploy this bean with the **-sqlj** option and then re-deploy it from the administrative console when you install the application, this bean is then deployed with JDBC access.
- For JDBC access, use the **ejbdeploy** command or the administrative console to deploy the CMP bean.

  If the bean is not deployed, the installation of the application will fail.

Steps for this task
1. Deploy the EAR file.
   a. Ensure directory <was_home>/bin is in your PATH.
   b. Verify the ws_ant.bat file is available on your system.

      You need the ws_ant.bat file to run the rest of the commands.

      Note: On Windows, the Apache Ant tool is ws_ant.bat On UNIX systems, the Ant tool is ws_ant.
   c. Run the **ejbdeploy** command utility with the **-sqlj** option.

      You must supply the location of the SQLJ translator sqlj.zip file with the **-cp** (classpath) option: (split for publication)

      ```
      ejbdeploy d:\MyApplication.ear working d:\deployedMyApplication.ear
            -sqlj -dbvendor DB2UDB_V81
      -cp "C:\PROGRA~1\IBM\SQLLIB\java\sqlj.zip;
       C:\PROGRA~1\IBM\SQLLIB\java\sqlj.zip "
      ```

The **ejbdeploy** command will not access sqlj.zip from your system classpath. The **ejbdeploy** command will generate an EAR file with the name you specify as, for example, deployedMyApplication.ear and an Ant script with the name <InputEarName>ear.xml, or in this example, deployedMyApplication.ear.xml.

If you create the EAR file using WebSphere Studio Application Developer (WSAD), you can associate SQLJ with a particular database backend in the mapping editor. If you use the mapping editor, you do not have to specify the **-sqlj** option in the **ejbdeploy** command.

2. **(Optional)** Perform the SQLJ profile customization steps if your application is running in a clustered environment.

   If your application is not running in a clustered environment, go to the **was_ant** properties description.

   Perform customization once, on a single host.

   a. Run the DB2 SQLJ customizer, **db2sqljcustomize** , against the serialized profiles.

      One serialized profile exists for each EJB .jar file. You can find the serialized profile in the EJB .jar file. One example of a serialized profile file name is FS_TopDown1_SJProfile0.ser.

      When you run the DB2 SQLJ customizer with the "**-automaticbind yes** " default option against the serialized profiles, you create static SQL in the database, which is used at runtime. The customization phase creates four database packages that contain static SQL, one for each isolation level.

   b. Include the customized profiles in the EJB .jar files installed on WebSphere Application Server.

      The customization step also updates the generated customized profiles. To make these updated customized profiles available to the generated code at runtime, include the profiles in the EJB .jar file, which is installed in WebSphere Application Server.

      Note: If you omit the customization step, the EJB applications will run, but they will not use the static SQL stored in the database, and you lose all the benefits of SQLJ.

   c. Use the Ant script to make customization easier.

      When you run batch SQLJ ejbDeploy against an EAR file, it produces an Ant script. Use this script file to run the DB2 customizer against every serialized profile in every EJB .jar file in the associated EAR file. The script updates each EJB .jar file with a serialized profile, and replaces the .jar files in the EAR file with the modified versions. The Ant script is specific to the corresponding EAR file.

      Note: This script modifies the existing EAR file.

      The script also uses a set of default names for the packages created in the database. Change the names used by the script file to ensure the names for each customization profile do not conflict with existing package names in the database. Ant scripts, generated for different EAR files, by default use the same package names, and will overwrite existing packages unless the names are changed. Overwritten packages cause runtime errors.

   d. Change the values of the database URL, and the database userid and password properties in the generated Ant script.

      The package names, database URLs, userids, and passwords are created in the script using Ant properties.

      The Ant script defines the following global properties:
      - for the database URL - db.url

- for the user - `db.user`
- for the password - `db.password`

e. Change the values used by the Ant script and the serialized profile by specifying new values in a property file.

The Ant script uses the URL, userid, and password properties in the serialized profile to customize the profile. By default, the properties for the serialized profile are created from the global properties.

The script for a particular EAR file reads properties from two files:

- `ejbdeploy.sqlj.properties`

  The `ejbdeploy.sqlj.properties` file is common to all Ant scripts generated by the **EJBDeploy** command. Use the `ejbdeploy.sqlj.properties` file to specify global properties, like the database userid and password.

- `<ear-name>.properties`

  The `<ear-name>` value is the name of the EAR file. The `<ear-name>.properties` file is specific to the Ant script for the EAR file. If you want your Ant script to use a another file instead of `<ear-name>.properties`, specify the `script.property.file` property when you run the script.

f. Use the DB2 Control Center to identify the packages installed in the database.

The DB2 SQLJ customizer requires a type 4 database URL in the form of:

```
jdbc:db2://<host-name>:<port>/<database-name>
```

It also requires a userid and password. The value of the port is 50000, unless you changed it when you installed DB2.

3. Run the Ant script, specifying the **properties** target:

```
ws_ant -buildfile deployedMyApplication.ear.xml properties
```

This will create the properties file, `deployedMyApplication.ear.properties`. The `deployedMyApplication.ear.properties` file contains properties specifying default names for the packages corresponding to each serialized profile in the EAR file, as for example:

```
pkg.MyEJB1.jar.DB2UDBNT_V8_1=PKG1_
pkg.MyEJB2.jar.DB2UDBNT_V8_1=PKG2_
```

In this example, the EAR file contains two EJB `.jar` files: `MyEJB1.jar` and `MyEJB2.jar`.

4. Edit the generated properties file to change the package names.

You can also specify other properties in this file, such as the database URL, user, or password:

```
url.MyEJB1.jar.DB2UDBNT_V8_1
        =jdbc:db2://localhost:50000/MyDB1
user.MyEJB1.jar.DB2UDBNT_V8_1=dbuser
password.MyEJB1.jar.DB2UDBNT_V8_1
        =dbpassword
pkg.MyEJB1.jar.DB2UDBNT_V8_1=TEST
url.MyEJB2.jar.DB2UDBNT_V8_1
        =jdbc:db2://localhost:50000/MyDB2
user.MyEJB2.jar.DB2UDBNT_V8_1=dbuser
password.MyEJB2.jar.DB2UDBNT_V8_1
        =dbpassword
pkg.MyEJB2.jar.DB2UDBNT_V8_1=WORK
```

DB2 uses the first seven characters of the package name. The DB2 customizer uses this name to create four packages in the database. For example, if you specify the name TEST, the customizer will create packages called:

```
TEST1, TEST2, TEST3, TEST4
```

5. Use the following **db2sqljcustomize** options to temporarily circumvent profile customization problems.

   These options bypass errors during a profile customization and ensure a successful customization:

   ```
   -onlinecheck NO and -bindoptions "VALIDATE RUN"
   ```

   However, you must understand what the problems are and fix them at the appropriate time.

6. Run the Ant script.

   The DB2 db2jcc.jar file must be on the classpath. This file should have been added to the classpath environment variable when DB2 V8 FixPak1 was installed.

7. Specify a working directory for the Ant script.

   The script will create and delete files and sub-directories in this directory. If the working directory contains existing files and directories with the same name as the files and directories used by the script, the script will erase the files and directories.

   Use the following command to specify a working directory:

   ```
   ws_ant -Dwork.dir=tmp -buildfile MyApplication.ear.xml
   ```

   The script creates and uses a directory called tmp as its working directory. If you want the script to use a different property file, set the script.property.file property when you run the script:

   ```
   ws_ant -Dwork.dir=tmp -Dscript.property.file=other.properties
     -buildfile MyApplication.ear.xml
   ```

   The Ant script updates the original EAR file with the modified serialized profiles.

8. If you rerun the **EJBDeploy** command, rerun the Ant script.

   Generate a new properties file if any of the following conditions are true:
   - You changed the number of .jar files in the EAR file.
   - You changed the names of the .jar files in the EAR file.
   - You changed the database backend ids in any of the .jar files

9. Install the updated EAR file in WebSphere Application Server.

10. Create a jdbc provider and data source in WebSphere Application Server:

    To create a JDBC provider, in the administrative console:

    a. Click Resources > JDBC providers > NEW
    b. Choose the default value for DB2 universal JDBC Driver provider.
    c. Click Apply.

    To create a data source under this JDBC provider in the administrative console:

    a. Scroll down to the **Additional Properties** panel.
    b. Click DataSource > NEW
    c. Define the Alias from the **J2C Authentication Data Entries**
    d. Return to the data source panel .

e. Enter the alias in both the **Component** and **Container** managed authentication alias entries.

f. Specify the **databaseName** on the **Custom Properties** panel.

g. For a data source using the universal driver, specify the **portNumber** and **serverName** on the **Custom Properties** panel.

11. Install your application through the administrative console but do not re-deploy the EJB.

    If you check the **EJBDeploy** box, your application is redeployed to JDBC access.

12. Stop the server.

13. Start the server.

    You are now ready to run your application.

Using Structured Query Language in Java (SQLJ) for Enterprise Java Bean (EJB) Bean Managed Persistence (BMP) entity beans, session beans, and servlets

This article describes how JDBC applications, comprised of Bean Managed Persistence (BMP) beans or servlets, are converted to SQLJ applications. It also describes how the SQLJ applications are then deployed in WebSphere Application Server.

Follow these steps precisely and in the right order to ensure a correct conversion:

Steps for this task

1. Create a backup copy of your .java file.

   For example if your file is called MyServlet.java, copy MyServlet.java to MyServlet.java.bkup.

2. Rename your .java file to to a file name with a .sqlj extension.

   For example, if your application is a servlet named MyServlet.java, rename MyServlet.java to MyServlet.sqlj.

   Now when you run the **sqlj** tool in the next step, the .java file that it creates will have the same name as your old .java file, providing you with a seamless transition to the SQLJ technology.

3. Edit the .sqlj file to convert the JDBC syntax to SQLJ syntax.

   When using SQLJ in WebSphere Application Server, if you want WebSphere Application Server connection management to function properly, you must specify correct connection contexts.

   For example, convert the following JDBC operation:

```
        Connection con = dataSource.getConnection();
        Statement stmt = con.createStatement();
        stmt.execute("INSERT INTO users VALUES (1, 'user1')");
// At the top of the file and just below the import statements,
// define Connection_Context
        con.commit();
```

   to the following SQLJ:

```
        #sql context Connection_context;
         .
         .
        Connection con = dataSource.getConnection();
         .
         .
        Connection_context ctx1 = new Connection_context(con);
```

```
                                .
                                .
                                .
              #sql [ctx1] {INSERT INTO users VALUES (1, 'user1')};
                                .
                                .
                                .
              con.commit();
              ctx1.close();
```

4. Run the DB2 **sqlj** translator.

   This tool creates a `.java` version of your `.sqlj` file as well as a `.ser` profile
   that is used later in the processing. Refer to the DB2 documentation for more
   information on this tool.

5. Package your EJB jar and deploy it in the usual manner.

6. After the EJB files are deployed, run the **db2sqljcustomize** tool to customize
   your `.ser` file.

   One serialized profile exists for each EJB `.jar` file. You can find the profile in
   the EJB `.jar` file. One example of a serialized profile file name is
   MyBMPBeanProfile.ser.

   When you run the DB2 SQLJ customizer against the serialized profiles, you
   create static SQL in the database, which is used at runtime. The customization
   phase creates four database packages that contain static SQL, one for each
   isolation level.

7. Configure your database.

8. Update your EJB jar with the `.ser` file.

9. Package your EJB jar and servlets, along with `.ser` files, into an `.ear` file.

10. Install the application in the usual manner.

Assembling data access applications

Steps for this task

1. Define the resource reference attributes through the Application Assembly Tool.

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.

4. Map to EJB deployment tool schema

Example: Configuring isolation level on a resource reference during assembly

Specify the appropriate *isolation level* for each of your resource references. The
isolation level is used only by JDBC users. For information about setting isolation
levels on a resource reference, see Isolation level and resource reference.

You can define multiple resource references for the same bean-managed persistence
(BMP) bean or servlet. Each resource reference has its own Java Naming and
Directory Interface (JNDI) logical name. You can also bind multiple resource
references to the same data source. Each JNDI resource reference naming lookup
returns the connections from the data source that has the same isolation level.

For example, an account bean has two resource references defined:

```
jdbc/RRResRef:
resource_ref has RepeatableRead isolation level
jdbc/RCResRef:
resource_ref has ReadCommitted isolation level
```

If your application uses *RRResRef* to get the data source instance, all connections that are acquired from this data source instance have the RepeatableRead isolation level.

If your application uses *RCResRef* to get the data source instance, all connections that are acquired from this data source instance have the ReadCommitted isolation level.

Enterprise bean deployment tool schema

There are three options for mapping entity beans to a backend database schema: top-down, meet-in-the-middle, and bottom-up.

With the top-down option, the deployment tools generate a backend schema directly from the provided abstract persistence schema (abstract accessor methods plus the deployment descriptor elements) and then map the elements in the abstract persistence schema to the generated backend schema. See the section *Generating a top-down mapping* of the WebSphere Studio Application Developer (WSAD) documentation for details.

With meet-in-the-middle mapping, you employ a tool to manually map elements from the abstract persistence schema to elements in an existing backend schema. WSAD provides a meet-in-the-middle mapping tool. See the section *Generating a meet-in-the-middle mapping* of the WSAD documentation for details.

With the bottom-up approach, the tools take, as input, an existing backend database schema and generate equivalent entity beans and then map the elements of the two schemas. See the section *Generating a bottom-up mapping* of the WSAD documentation for details.

Migrating a version 4.0 data access application to version 5.0

In order to utilize the connection management infrastructure in WebSphere Application Server Version 5.0, you must package your application as a J2EE 1.3 application. This means repackaging your web modules to the 2.3 Specification and your EJB modules to the 2.0 Specification before installing them into WebSphere Application Server.

Note that with this repackaging comes the need to use Version 5.0 data sources. With the repackaging process, you cannot use a Version 4.0 data source. The 4.0 data source case is very simple, though, you just create the JDBC provider and data source, and install the 4.0 application as-is.

Converting a 2.2 web module to a 2.3 web module
Use the following steps to migrate each of your web modules.

Steps for this task
1. Open the Application Assembly Tool.
2. Create a new Web Module by selecting **File -> New -> Web Module**.
3. Add any required class files to the new module.
   a. Expand the **Files** portion of the tree.
   b. Right click **Class Files** and select **Add Files**.
   c. In the Add Files window click **Browse**.
   d. Navigate to your WebSphere 4.0 *EAR* file and click **Select**.

e. In the upper left pane of the Add Files window, navigate to your *WAR* file and expand the **WEB-INF** and **classes** directories.

f. Select each of the directories and files in the classes directory and click **Add**.

g. When you have added all of the required class files, click **OK**.

4. Add any required *JAR* files to the new module.

   a. Expand the **Files** portion of the tree.

   b. Right click **Jar Files** and select **Add Files**.

   c. Navigate to your WebSphere 4.0 *EAR* file and click **Select**.

   d. In the upper left pane of the Add Files window, navigate to your *WAR* file and expand the **WEB-INF** and **lib** directories.

   e. Select each *JAR* file and click **Add**.

   f. When you have added all of the required *JAR* files, click**OK**.

5. Add any required resource files, such as HTML files, JSP files, GIFs, and so on, to the new module.

   a. Expand the **Files** portion of the tree.

   b. Right click **Resource Files** and select **Add Files**.

   c. Navigate to your WebSphere 4.0 *EAR* file and click **Select**.

   d. In the upper left pane of the Add Files window, navigate to your *WAR* file.

   e. Select each of the directories and files in the *WAR*, *excluding* **META-INF** and **WEB-INF**, and click **Add**.

   f. When you have added all of the required resource files, click**OK**.

6. Import your Web Components.

   a. Right click on **Web Components** and select **Import**.

   b. In the Import Components window click **Browse**.

   c. Navigate to your WebSphere 4.0 *EAR* file and click **Open**.

   d. In the left top pane of the **Import Components** window highlight the *WAR* file you are migrating.

   e. Highlight each of the components displayed in the right top pane and click **Add**.

   f. When all of your web components show up in the Selected Components pane of the window, click **OK**.

   g. Verify that your web components are correctly imported under the Web Components section of your new web module.

7. Add servlet mappings for each of your web components.

   a. Right click **Servlet Mappings** and select **New**.

   b. Fill in a URL pattern for the web component.

   c. Select the web component from the Servlet drop down box.

   d. Click **OK**.

8. Add any necessary resource references by following the instructions in Creating a resource reference.

9. Add any other web module properties that are required. Click **Help** for a description of the settings.

10. **Save** the web module.

Converting a 1.1 EJB module to a 2.0 EJB module

Use the following steps to migrate each of your EJB modules.

Steps for this task

1. Open the Application Assembly Tool.
2. Create a new EJB Module by selecting **File -> New -> EJB Module**.
3. Add any required class files to the new module.
   a. Right click **Files object** and select **Add Files**.
   b. In the Add Files window click **Browse**.
   c. Navigate to your WebSphere 4.0 *EAR* file and click **Select**.
   d. In the upper left pane of the Add Files window, navigate to your EJB *JAR* file.
   e. Select each of the directories and class files and click **Add**.
   f. When you have added all of the required class files, click **OK**.
4. Create your session beans and entity beans. To find help on this subject, see Migrating enterprise bean code to the supported specification, the documentation for WebSphere Studio Application Developer, or the documentation for WebSphere Studio Application Developer Integration Edition.
5. Add any necessary resource references by following the instructions in Creating a resource reference.
6. Add any other EJB module properties that are required. Click **Help** for a description of the settings.
7. **Save** the EJB module.
8. Generate the deployed code for the EJB by selecting **File -> Generate Code for Deployment**.
9. Fill in the appropriate fields and click **Generate Now**.

Add the EJB Modules and Web Modules to an EAR file

Steps for this task

1. Open the Application Assembly Tool.
2. Create a new Application by selecting **File -> New -> Application**.
3. Add each of your EJB modules.
   a. Right click **EJB Modules** and select **Import**.
   b. Navigate to your converted EJB Module and click **Open**.
   c. Click **OK**.
4. Add each of your web modules.
   a. Right click **Web Modules** and select **Import**.
   b. Navigate to your converted web module and click **Open**.
   c. Fill in a **Context root** and click **OK**.
5. Fill in any other Application properties. Click **Help** for a description of the settings.
6. Save the *EAR* file.

Installing the Application in WebSphere

Steps for this task

1. Create a JDBC provider and a Version 5.0 data source object following the instructions in Creating a JDBC provider and data source.

2. Install the application, following the instructions in Installing a new application and binding the resource references to the data source you created.

Connection considerations when migrating servlets, JavaServer Pages, or enterprise session beans

If you are migrating from Websphere Application Server Version 3.5.x or Version 4.0.x to WebSphere Application Server Version 5.0.x, and you are running servlets, JavaServer Page (JSP) files, or enterprise session beans with bean managed transactions, a behavior change in the data access component can adversly affect the use of connections in your applications.

This change affects all applications that contain the following methods:
- `RequestDispatcher.include()`
- `RequestDispatcher.forward()`
- JSP includes (`<jsp:include>`)

Symptoms of the problem include:
- Session hang
- Session timeout
- Running out of connections

In Websphere Application Server Version 4.0.x and earlier releases, JSP files, servlets, and enterprise session beans with bean managed transactions are issued non-sharable connections, while all the other components are issued shareable connections.

Starting with Websphere Version 5.0.x, all connections that are accessed with a resource-reference, default to shareable unless specified unshareable in the resource-reference. This change is required for compliance with the J2EE 1.3 Specification. Using shareable connections in this context has the following effects:
- All connections received and used outside the scope of a user transaction are *not* returned to the free connection pool until the encapsulating method returns, even when the connection handle issues a **close()** .
- All connections received and used outside the scope of a user transaction are *not* shared with other component instances (that is, other servlets, JSP files, or enterprise java beans).

  For example, if session bean 1 gets a connection and then calls session bean 2 that also gets a connection, even if all properties are identical, each session bean receives its own connection.

If you did not anticipate this change in the connection behavior, the way you structure your application code could lead to excessive connection use, particularly in the cases of JSP includes, `RequestDispatcher.include()`routines, `RequestDispatcher.forward()` routines, session beans with bean managed transactions, or calls from these methods to other components.

Servlet A gets a connection, completes the work, commits the connection, and calls **close()** on the connection. Next, servlet A calls the `RequestDispatcher.include()` to include servlet B, which performs the same steps as servlet A. Since the servlet A connection did not return to the free pool until it returned from the current method, two connections are now busy. In this way, more connections might be in use than you intended in your application. If these connections are not accounted for in the **Max Connections** setting on the connection pool, this behavior might cause a lack of connections in the pool, which would result in

ConnectionWaitTimeOutExceptions. If the **connection wait timeout** is not enabled, or if the **connection wait timeout** is set to a large number, these threads might appear to hang because they are waiting for connections that are never returned to the pool. Threads waiting for new connections will not return the ones they are currently using if new connections are not available.

To resolve these problems:

1. Use unshared connections.

   If you use an unshared connection and are not in a user transaction, the connection is returned to the free pool when you issue a `close()`, (assuming you commited or rolled back the connection).

2. Increase the maximum number of connections.

   To calculate the number of required connections, multiply the number of configured threads by the deepest level of component call nesting (for those calls that use connections). See the Examples of the connection behavior change section for a description of call nesting.

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*.

Assembling Resource Adapter modules

The Resource Adapter Archive file contains code that implements a library for connecting with a backend Enterprise Information System (EIS). You may see the terms Resource Adapter *modules*, Resource Adapter *connectors* and Resource Adapter *archive files* used interchangeably.

Steps for this task

1. Click **File>New> Resource Adapter**.
   - If you are working with an existing file, click **File** > **Open**. Select the file you wish to modify.

   The navigation pane displays a hierarchical structure used to build the contents of the archive. The icons represent the security permissions, configuration properties, and files for the archive. A property dialog box containing general information about the archive is displayed in the property pane.

2. Enter values for other properties as needed.

   Note: The remaining information in this step only applies if you are creating *new* archive files.By default, the archive file name and the module display

name are the same. It is recommended that you change the display name in the property pane. Also by default, the temporary location of the Resource Adapter module is `installation_directory/bin`.

  a. Specify a new file name and location by clicking **File>Save**.

3. Enter the EIS type, vendor name, version, and specification (required).

  a. Click on the **Advanced** tab, and enter the implementation and interface class names for the EIS connection.

  b. Click **Browse** to locate the class files. By default, the root directory or archive is the current archive. If needed, browse the file system for the directory or archive where class files reside.

  c. Click **Select**.

    The archive's file structure is displayed in the window.

  d. Expand the structure and locate the files that you need. Select the file and click **OK**.

4. Define other properties for the resource adapter.

  a. Right-click the **Security Permissions** icon and click **New**. A property dialog box for Security Permissions is displayed.

  b. Enter values for each property and then click **OK**. Repeat to specify multiple security permissions. The top portion of the property pane lists each reference.

  c. Select the reference to view its corresponding property dialog box.

  a. Right-click the **Configuration Properties** icon and click **New**.

  b. A property dialog box for Configuration Properties is displayed.

  c. Enter values for each property and then click **OK**.

  d. Repeat to specify multiple configuration properties.

5. Add files for the Resource Adapter.

  a. In the navigation pane, right-click the **Files** icon and then choose **Add Files**. Use the file browser to locate files to add.

  b. First, browse for the root directory or archive where the files are located and click Select. If you are adding an entire archive, select the directory that contains the archive. The directory structure is displayed in the left pane.

  c. Browse the directory structure.

  d. From the right pane, select one or more files to be added and click **Add**. If you select a directory and click Add, all files in the directory, including the directory, are added. Relative path names are maintained.

  e. The selected files are displayed in the Selected Files window. Click **OK**. The file names, extensions, modification dates, sizes, and path names are displayed in the property pane.

6. Review the contents of the archive and make any desired changes.

What to do next

"Saving applications after assembly" (not in this document). "Verifying archive files" (not in this document). "Generating code for deployment" (not in this document).

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.

Steps for this task

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. Migrate Version 4.0 data access applications
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.

Relationship of assembly and administrative console data access settings

This article provides miscellaneous tips for using supported databases. See also the related links.

Always consult the product documentation for a list of the database brands and versions that are supported by your particular application server version, edition, and FixPak.

Notes about various databases

- When using local DB2 databases for data access by session clients on AIX Version 4.3.3 or later versions, in some cases you cannot establish multiple connections for session clients. This is because AIX , by default, does not permit 32-bit applications to attach to more than 11 shared memory segments per process. Of these 11 shared segments, a maximum of 10 can be used for local DB2 connections. To use EXTSHM with DB2 and avoid stale connections when there are large numbers of session clients, do the following:
  - In DB2 client environment (that is the WebSphere Application Server run time environment in this case):

    ```
    export EXTSHM=ON
    ```

    – In DB2 UDB Server environment:

```
export EXTSHM=ON
db2set DB2ENVLIST=EXTSHM
```

- When using Sybase 11.x, you might encounter the following error when HttpSession persistence is enabled:

```
DBPortability W Could not create database table: "sessions"
com.sybase.jdbc2.jdbc.SybSQLException: The 'CREATE TABLE' command is not
allowed within a multi-statement transaction in the 'database_name' database
```

where *database_name* is the name of the database for holding sessions.

If you encounter the error, issue the following commands at the Sybase command line:

```
use database_name
go
sp_dboption db,"ddl in tran ",true
go
```

-

Sybase 12.0 does not support local transaction modes with a JTA enabled data source. To use a connection from a JTA enabled data source in a local transaction, install Sybase patch EBF9422.

Additional administrative tasks for specific databases

For your convenience, this article provides instructions for enabling some popular database drivers, and performing other administrative tasks often required to provide data access to applications running on WebSphere Application Server. These tasks are performed outside of the WebSphere Application Server administrative tools, often using the database product tools. Always refer to the documentation accompanying your database driver as the authoritative and complete source of driver information.

See the [ ] for the latest information about supported databases, drivers, and operating systems.

Enabling JDBC 2.0: Ensure that your operating system environment is set up to support JDBC 2.0. This action is required to use data sources created through WebSphere Application Server.

The following steps make it possible to find the appropriate JDBC 2.0 driver for use with WebSphere Application Server administration:

Enabling JDBC 2.0 with DB2 on Windows NT systems: To enable JDBC 2.0 use on Windows NT systems:

- For DB2 Version 7.2
    1. Stop the DB2 JDBC Applet Server service.
    2. Run the following batch file:

       ```
       SQLLIB\java12\usejdbc2.bat
       ```

    3. Stop WebSphere Application Server (if it is running) and start it again.
- For DB2 Version 8.1
    – JDBC 2.0 is supported by default, there are no additional steps for you to perform.

Perform the steps once for each system.

Determining the level of the JDBC API in use for DB2 on Windows NT:
systems

To determine the JDBC level in use on your system:
- For DB2 Version 7.2
  - If JDBC 2.0 is in use, this file exists:

    `SQLLIB\java12\inuse`
  - If JDBC 1.0 is in use, this file exists:

    `SQLLIB\java11\inuse`

    or no *java11* directory exists.
- For DB2 Version 8.1
  - Go to directory *SQLLIB\samples\java*, compile and run the class
    db2JDBCVersion.java.

Enabling JDBC 2.0 with DB2 on UNIX systems:
- For DB2 Version 7.2
  - Before starting WebSphere Application Server, call
    $INSTHOME/sqllib/java12/usejdbc2 to use JDBC 2.0. For convenience, you
    might want to put this in your root user's startup script. For example, on AIX,
    add the following to the root user's *.profile*:

    ```
    if [ -f /usr/lpp/db2_07_01/java12/usejdbc2 ] ; then
        . /usr/lpp/db2_07_01/java12/usejdbc2
    fi
    ```
- For DB2 Version 8.1
  - JDBC 2.0 is supported by default, there are no additional steps for you to
    perform.

Determining the level of the JDBC API in use for DB2 on UNIX systems:
- For DB2 Version 7.2
  - To determine if you are using JDBC 2.0, you can echo *$CLASSPATH*. If it
    contains

    `$INSTHOME/sqllib/java12/db2java.zip`
    then JDBC 2.0 is in use.
    If it contains

    `$INSTHOME/sqllib/java/db2java.zip`
    then JDBC 1.0 is in use.
- For DB2 Version 8.1
  - Go to directory *sqllib/samples/java*, compile and run the class
    db2JDBCVersion.java.

Sourcing the db2profile script on UNIX systems: Before starting WebSphere
Application Server to host applications requiring data access, source the *db2profile*:

`. ~db2inst1/sqllib/db2profile`

where *db2inst1* is the user created during DB2 installation.

Using Java Transaction API drivers: Instructions are available for using Java
Transaction API (JTA) drivers on particular operating systems. See your operating
system documentation for more information.

The goal of this section is to provide information about the steps that make DB2 work well with applications utilizing XA classes — that is, those whose *dataSourceClasses* implement *javax.sql.XADataSource*.

Using Java Transaction API drivers for DB2 on Windows NT systems: To enable JTA drivers for DB2 on Windows NT systems, follow these steps:

1. Bind the necessary packages to the database. From the **DB2 Command Line Processor** window, issue the following commands:

   ```
   db2=> connect to mydb2jta
   db2=> bind db2home\bnd\@db2cli.lst
   db2=> bind db2home\bnd\@db2ubind.lst
   db2=> disconnect mydb2jta
   ```
   where *mydb2jta* is the name of the database to enable for the JTA, and *db2home* is the DB2 root installation directory path (for example, *D:\ProgramFiles\SQLLIB\bnd\@db2cli.lst*).

2. Specify the following settings when you use an IBM WebSphere Application Server administrative client (such as the WebSphere Administrative Console) to configure a JDBC driver:

   - **Server class path** = `%DB2_ROOT%/Sqllib/java/db2java.zip`
   - **Implementation class name** = `COM.ibm.db2.jdbc.DB2XADataSource`

Using Java Transaction API drivers for DB2 on UNIX systems: To enable JTA drivers on UNIX systems, follow these steps:

1. Stop all DB2 services.
2. Stop the IBM WebSphere Application Server administrative service.
3. Stop any other processes that use the *db2java.zip* file.
4. Make sure that you already enabled JDBC 2.0.
5. Start the DB2 services.
6. Bind the necessary packages to the database. From the DB2 command-line process or window, issue the following commands:

   ```
   db2=> connect to mydb2jta
   db2=> bind db2home\bnd\@db2cli.lst
   db2=> bind db2home\bnd\@db2ubind.lst
   db2=> disconnect mydb2jta
   ```

7. Specify the following settings when you use an IBM WebSphere Application Server administrative client (such as the WebSphere Administrative Console) to configure a JDBC driver:

   - **Server class path** = `$INSTHOME/sqllib/java12/db2java.zip`

     For example, if *$INSTHOME* is `/home/test`, the path will be `/home/test/sqllib/java12/db2java.zip`
   - **Implementation class name** = `COM.ibm.db2.jdbc.DB2XADataSource`

For Oracle 8.1.7 two phase commit support: You can use the Oracle 8.1.7 thin driver for JTA two-phase support with the following restrictions:

- The thin driver that comes shipped with 8.1.7 might or might not work. Future patches from Oracle might work as well, but are not tested. The driver that was available from the Oracle Technology Network Web site as of February 20, 2001 does work and is the recommended driver. Later versions on this Web site are expected to work, but are not tested.

  To obtain the driver from the Oracle support Web site, visit:

  `http://technet.oracle.com/`

You need to be a registered user for the Oracle Technology Network to get the driver from this site. Contact Oracle for access. After you have access download the 8.1.7 driver for the platforms you use and follow the instructions for installing the new driver.

- You must use the 8.1.7 driver with 8.1.7 databases, 8.1.6 databases do not support the recover() and forget() methods and other problems are encountered running with 8.1.6. Oracle does not support JTA with 8.1.6.
- For Oracle, you can only use JTA with container-managed persistence (CMP) beans.
- For the bean to create the table, you must start the bean with the JTA set to *false*. After the bean creates the table, you can set the JTA back to *true*.
- Configure an entity bean that accesses Oracle with JTA set to *true* as follows:
  - Click **deployment descriptor properties** > **Transactions** > Remote tab. Set the Transaction Attribute to *TX_REQUIRED*.
  - Click **Isolation** > Remote tab. Set the Isolation Level to *TRANSACTION_READ_COMMITTED*.
- Configure a session bean that is used with an entity bean that accesses Oracle with JTA set to *true* as follows:
  - Click **deployment descriptor properties** > **Transactions** > Remote tab. Set the Transaction Attribute to *TX_BEAN_MANAGED*.
  - Click **Isolation** > Remote tab. Set the Isolation Level to *TRANSACTION_READ_COMMITTED*.

Using Java Transaction API drivers for Sybase products on AIX systems: To enable Java Transaction API (JTA) drivers for use with Sybase products on the AIX operating system, follow these steps:

1. Enable the Data Transaction Manager (DTM) by issuing these commands (one per line) at a command prompt:

   ```
   isql -Usa -Ppassword -Sservername
   sp_configure "enable DTM", 1
   go
   ```

2. Stop the Sybase Adaptive Server database and start it again.
3. Grant the appropriate role authorization to the enterprise bean user at a command prompt:

   ```
   isql -Usa -Ppassword -Sservername
   grant role dtm_tm_role to EJB
   go
   ```

Notes about Sybase Java Transaction API drivers: Do not use a Sybase Java Transaction API (JTA) connection in an enterprise bean method with an unspecified transaction context. A Sybase JTA connection does not support the local transaction mode. The implication is that you must use the Sybase JTA connection in a global transaction context.

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:

Steps for this task

1. The container-managed bean (CMP) enterprise bean JAR file has a *Table.ddl* file that the Application Assembly Tool (AAT) generates. Extract the *Table.ddl* file to a working directory such as *C:\temp*.
2. Run this command — **C:\temp>db2cmd**

   A new command window appears in which you enter the following commands.
3. Run this command — **C:\temp>db2 connect <dbname>**
4. Run this command — **C:\temp>db2 -tf Table.ddl** <div/>

   The command runs and creates tables for your CMP enterprise bean.
5. Run this command — **C:\temp>db2 disconnect all** <div/>

Installing Java 2 Connector resource adapters

Steps for this task
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, and the node the file is installed on 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.
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

Steps for this task
1. Assemble an application with resource adapter archive (RAR) modules in it. See "Assembling applications" (not in this document).
2. Install the application following the steps in "Installing a new application" (not in this document).

   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

Creating and configuring a JDBC provider and data source

<div/>

Steps for this task
1. Decide which data source to use: Data source (Version 4.0) or a Version 5.0 data source.
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.
5. Test the connection (for non-container-managed persistence usage). See Test connection.

Verifying a connection

User errors cause most test connection problems. 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, after completing these steps and reviewing the applicable information, your connection is still not successful, check the SystemOut.log for warning or exception messages. Then use the technical support search funtion to find known problems.

Steps for this task

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.

Steps for this task

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.

   Note: Using the administrative console to create a DB2 JDBC provider for a zSeries server causes an exception to occur.

   When you use the administrative console to create a **DB2/390 JDBC Provider** or a **DB2/390 JDBC Provider (XA)**, an exception occurs with the message *Name*

is not a valid entry. To work around this problem, choose the normal **DB2 JDBC Provider** to create a JDBC provider. They both use the same implementation class. When you create a data source under this provider, change the data store helper value from *DB2DataStoreHelper* to *DB2390DataStoreHelper*. Otherwise, your database access does not work.

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 *D:/SQLLIB/java/db2java.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

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.

Steps for this task

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

Chapter 3. Accessing data from applications   **227**

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 source settings

Use this page to create a data source under a JDBC provider which provides the specific JDBC driver implementation class.

Be sure that you want to use a Version 5.0 data source. If you are using the Enterprise JavaBean component model version 1.0 and Servlets 2.2, you must use a Version 4.0 data source.

To view this administrative console page, click **Resources** > **JDBC Providers** > *JDBC_provider* > **Data Sources** > *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 data source.

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

Container managed persistence: Specifies if this data source is used for container managed persistence of enterprise beans.

If the checkbox is selected, a CMP Connector Factory that corresponds to this data source is created for the relational resource adapter.

Data type	Checkbox
Default	Not checked

Description: Specifies a text description for the resource.

Data type	String

Category: Specifies a category string you can use to classify or group the resource.

Data type	String

Statement Cache Size: Specifies the number of free statements that are cached per connection.

The WebSphere Application Server data source optimizes the processing of prepared statements. A prepared statement is a precompiled SQL statement that is stored in a prepared statement object. This object is used to efficiently execute the given SQL statement multiple times.

If the cache is not large enough, useful entries are discarded to make room for new entries. To determine the largest value for your cache size to avoid any cache

discards, add the number of uniquely prepared statements, callable statements (as determined by the *sql* string, concurrency, and the scroll type) for each application that uses this data source on a particular server. This value is the maximum number of possible prepared statements that are cached on a given connection over the life of the server. Setting the cache size to this value means you never have cache discards. In general, the more statements your application has, the larger the cache should be. For example, if the application has 5 SQL statements, set the statement cache size to 5, so that each connection has 5 statements.

You can also use the Tivoli Performance Viewer to minimize cache discards. Use a standard workload that represents a typical number of incoming client requests, use a fixed number of iterations, and use a standard set of configuration settings. **Note:** The higher the statement cache, the more system resources are delayed. Therefore, if you set the number too high, you could lack resources because your system is not able to open that many prepared statements.

In test applications, tuning the statement cache improved throughput by 10-20%. However, because of potential resource limitations, this might not always be possible.

Data type   Integer
Default   Depends on the database. Most are 10.
                                        Informix Version 7.3, 9.2, or 9.3 without latest
                                        fix must also be 0. A default of 0 means there
                                        is no cache statement.

Datasource Helper Classname: Specifies the datastore helper that is used to perform database specific functions.

This is used by the Relational Resource Adapter at runtime.The default DataStoreHelper implementation class is set based on the JDBC driver implementation class, using the structure: *com.ibm.websphere.rsadapter.<database>DataStoreHelper*. For example, if the JDBC provider is DB2, then the default DataStoreHelper class is *com.ibm.websphere.rsadapter.DB2DataStoreHelper*. You can change to your subclass of this DataStoreHelper if necessary.

Data type   String
Default   Dependent on JDBC driver implentation class

Component-managed Authentication Alias: This alias is used for database authentication in run time.

If your resource authentication (res-auth) is set to *Application*, set the alias in the Component-managed Authentication Alias.

If you do not set this field and your database requires the user ID and password to get a connection, then you receive an exception during run time.

If your database (for example, Cloudscape) does not support *user ID* and *password*, then do not set the alias in the Component-managed authentication alias or Container-managed authentication alias fields. Otherwise, you see the warning message in the system log to indicate that the user and password are not valid properties. This message is a only warning message, therefore the data source is created successfully.

Data type   Pick-list

Container-managed Authentication Alias: This alias is used for database
authentication in run time.

If your res-auth is set to *Container*, set the Container-managed Authentication Alias.

If you do not set this field and your database requires the user ID and password to
get a connection, then you receive an exception during run time.

If your database (for example, Cloudscape) does not support *user ID* and *password*,
then do not set the alias in the Component-managed authentication alias or
Container-managed authentication alias fields. Otherwise, you see the warning
message in the system log to indicate that the user and password are not valid
properties. This message is a only warning message, therefore the data source is
created successfully.

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

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 driver on multiple nodes

These are the instructions to create a JDBC Driver in a Network Development (ND) environment. This sample configuration involves two base nodes: node A and node B added to ND on node C.

Steps for this task

1. Open the administrative console.
2. Click **Resources** > **JDBC Drivers**.

   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 Driver at the cell level.

> For the classpath filed while creating the data source, the default is
> *${DB2_JDBC_DRIVER_PATH}\db2java.zip*. Leave it at the default.

5. Finish creating the JDBC Driver.

6. Click **Environment**.

7. Click **Manage WebSphere Variables**.

8. Select each node.

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. If you provided a different variable (like
    ${DB2PATH}) when creating the JDBC Driver, you must create a new variable
    by clicking **New**.

    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 suffices for most invocations: (split for publication)

```
set classpath=%classpath%;D:\WebSphere\AppServer\lib\wsexception.jar;
D:\WebSphere\AppServer\lib\wasjmx.jar
```

The usual program follows these main points:

Steps for this task

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.

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
// (split fo publication)
//
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 XA datasource.
 * The datasource created is for CMP use.
 *
 * We need following to run
 * set classpath=%classpath%;D:\WebSphere\AppServer\lib\wsexception.jar;
 * D:\WebSphere\AppServer\lib\wasjmx.jar;
 * D:\$WAS_HOME\lib\wasx.jar
 * (split for publication)
 */
public class CreateDataSourceCMP {
// ds display name , also jndi name and CF name
   String dsName = "markSection";
 // database name
   String dbName = "SECTION";
// an authentication data alias
   String authDataAlias = "db2admin";
 // userid
   String uid = "db2admin";
 //  password
   String pw  = "db2admin";
 // path to the db driver
   String dbclasspath =
       "D:/SQLLIB/java/db2java.zip";




/**
 * 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);
// use the first node found
         node1 = matches[0];

// Use this group to add to the server1 scoped resource.xml.
         ObjectName server1 =
          ConfigSer
         viceHelper.createObjectName
                 (null, "Server", "server1");
         matches =
         configService.queryConfigObjects
                 (session, null, server1, null);
// use the first server found
         server1 = matches[0];

// Create the JDBCProvider
         String providerName = "DB2 JDBC Provider (XA)";
         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.jdbc.DB2XADataSource"));
         provAttrs.add(new Attribute
            ("description","DB2 JDBC2-compliant XA Driver"));

//create it
    ObjectName jdbcProv = configService.createConfigData
          (session,node1,"JDBCProvider",
                "resources.jdbc:JDBCProvider",provAttrs);
        // now plug in the classpath
configService.addElement(session,jdbcProv,"classpath",dbclasspath,-1);


// Search for RRA so we can link it to the datasource
         ObjectName rra = ConfigServiceHelper.createObjectName(null,
                     "J2CResourceAdapter", null);
         matches = configService.queryConfigObjects(session, node1, rra, null);
// use the first J2CResourceAdapter segment for builtin_rra
         rra = matches[0];

 // 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"));
            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 mark section CMP 2.0 test"));
            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);

    // Add a propertySet.
            AttributeList propSetAttrs = new AttributeList();
            ObjectName resourcePropertySet =
         configService.createConfigData
                (session,dataSource,"propertySet","",propSetAttrs);

     // 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);


    // 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 + " = "
                              + 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")) {
                break;
              }
            }


            if (i >= jras.length) {
               System.out.println("Did not find builtin_rra
                        + J2CResourceAdapter object creating CF anyways" );
            } else {
               System.out.println("Found builtin_rra J2CResourceAdapter
```

```
                                        + object at index " + i + " creating CF" );
            }

 // 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);


// ===== 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);
          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);
// ===== 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
//
// (split for publication)
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 XA datasource.
 * The datasource created is for BMP use.
 *
 * We need following to run
 * set classpath=%classpath%;D:\WebSphere\AppServer\lib\wsexception.jar;
* D:\WebSphere\AppServer\lib\wasjmx.jar;D:\$WAS_HOME\lib\wasx.jar
 */
public class CreateDataSourceBMP {
 // ds display name , also jndi name and CF name
    String dsName = "markSection";
// database name
    String dbName = "SECTION";
// an authentication data alias
    String authDataAlias = "db2admin";
 // userid
   String uid = "db2admin";
  //   password
   String pw  = "db2admin";
 // path to the db driver
  String dbclasspath = "D:/SQLLIB/java/db2java.zip";




  /**
   * 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();
   //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);
 // use the first node found
            node1 = matches[0];


 // 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);
 // use the first server found
            server1 = matches[0];
            // Create the JDBCProvider
            String providerName = "DB2 JDBC Provider (XA)";
            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.jdbc.DB2XADataSource"));
      provAttrs.add(new Attribute
            ("description","DB2 JDBC2-compliant XA Driver"));

  //create it
          ObjectName jdbcProv =
      configService.createConfigData(session,node1,"JDBCProvider",
            "resources.jdbc:JDBCProvider",provAttrs);
  // now plug in the classpath
        configService.addElement
              (session,jdbcProv,"classpath",dbclasspath,-1);


  // Search for RRA so we can link it to the datasource
        ObjectName rra =
         ConfigServiceHelper.createObjectName(null,
                    "J2CResourceAdapter", null);
        matches =
          configService.queryConfigObjects
                (session, node1, rra, null);
 // use the first J2CResourceAdapter segment for builtin_rra
        rra = matches[0];
 // 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"));
          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 mark section CMP 2.0 test"));
          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);

 // Add a propertySet.
        AttributeList propSetAttrs = new AttributeList();
        ObjectName resourcePropertySet =
         configService.createConfigData(session,dataSource,"propertySet","",
                             propSetAttrs);

// 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);

          // ===== 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);
          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);
     // ===== 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()

You need the following to run: set the classpath to *classpath=%classpath%; was_home\lib\wsexception.jar; was_home\lib\wasjmx.jar; was_home/lib/j2ee.jar;*

was_home\lib\admin.jar; was_home\lib\jmxc.jar; was_home\lib\utils.jar. Replace the variable *was_home* with the directory where you installed WebSphere Application Server.

```
//
// "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;


/**
 * Resource adapter test program to make sure that
 *  the MBean interfaces work.
 * Following interfaces are tested
 *
 *  -getPropertiesForDataSource()
 *  -reload()
 *  -testConnectionToDataSource()
 *
 *
 * We need following to run (split for publication)
 * set classpath=%classpath%;D:\WebSphere\AppServer\lib\wsexception.jar;
 *  D:\WebSphere\AppServer\lib\wasjmx.jar;D:\$WAS_HOME\lib\wasx.jar
 *
 */
public class testDS {


   String port = "8880";
   String host = "localhost";
   final static boolean verbose = true;



   /**
    * Main method.
    *
    * @param args    DataBase classpath, DataSource name
    */
   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();
         //ex.getCause().printStackTrace();
      } catch (Exception ex) {
         System.out.println("Caught this " + ex );
         ex.printStackTrace();
      }
```

```
            }


      /**
       * This method tests the DataSourceCfgHelper Mbean.
       *
       * @param args
       * @exception Exception
       */
      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);
           }

   // System.out.println("Connected to the application server"
                                       + handle);

   // now call a method on the Mbean to do the desired operation.
       String dsClassName = "COM.ibm.db2.jdbc.DB2XADataSource";
       String providerLibPath = "D:/SQLLIB/java/db2java.zip";
       String[] signature = { "java.lang.String", "java.lang.String"};
       Object[] params = { dsClassName, providerLibPath};
        Object result = null;

       if (verbose) {
       System.out.println("Calling getPropertiesForDataSource() for "
                   + dsClassName + "\n");
           }
           try {
```

```
        // get the properties
     result = adminClient.invoke(handle,
                     "getPropertiesForDataSource",
               params, signature);
        } 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);
        }

// Pretty print what we found
     Iterator propIterator = ((List)result).iterator();
     System.out.println(format("Name",21)+ "|"
             + format("Default Value",34)
              + "|" + format("Type",17) +"|Reqd");
        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"));
        }
        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

/* For the following to work you must give all permission
/* to the database jar/zip files
/*  else you will see this exception:
        DSRA8040W: Failed to connect to the DataSource.
        Encountered java.lang.ExceptionInInitializerError:
        Chained java.security.AccessControlException:
         access denied (java.util.PropertyPermission ibm.db2.instance.path read).

         e.g.
         Put this in server.policy
```

```
                                      grant codeBase "file:D:/sqllib/java/db2java.zip" {
                                        permission java.security.AllPermission;
                                      };
                        */

                        if (verbose) {
                           System.out.println("\nTesting connection to the database using "
                                         + dsClassName);
                        }


                        String user = "db2admin";
                        String password = "db2admin";
                        Properties props = new Properties();
                        props.setProperty("databaseName", "section");
                        // also pass in the locale i0226.09 onwards
                        String[] signature2 ={ "java.lang.String",
                                       "java.lang.String",
                                       "java.lang.String", "java.util.Properties",
                                       "java.lang.String","java.util.Locale"};
                        Object[] params2 = { dsClassName, user, password,
                                       props ,providerLibPath, Locale.US};
                        Object result2 = null;

                        try {
                // OK lets test.
                 result2 = adminClient.invoke(handle,
                            "testConnectionToDataSource",
                              params2, signature2);
                        } 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);
                           ex.printStackTrace();
                        }

                        if (result2 != null) {
                         System.out.println("ERROR Result= " + result2);
                        } else if (verbose) {
                         System.out.println("OK testConnectionToDataSource()");
                        }

                    } 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

```
#AWE --  Set up XA DB2 data sources, both 5.0 and 4.0

#UPDATE THESE VALUES:
#The classpath that will be used by your database driver
set driverClassPath "c:/sqllib/java/db2java.zip"

set server "server1"

set fvtbase "c:/wssb/fvtbase"

#Users and passwords..
set defaultUser1 "dbuser1"
set defaultPassword1 "dbpwd1"
set aliasName "alias1"

set databaseName1 "jtest1"
set databaseName2 "jtest2"
#END OF UPDATES

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 $defaultUser1]
set password_attr [list password $defaultPassword1]
set attrs [list $alias_attr $desc_attr $userid_attr $password_attr]

set authdata [$AdminConfig  create JAASAuthData $sec $attrs]
$AdminConfig save

puts "Installing DB2 datasource for XA"

puts "Finding the old JDBCProvider.."
#Remove the old jdbc provider...
set jps [$AdminConfig list JDBCProvider]
foreach jp $jps {
 set jpname [lindex [lindex [$AdminConfig show $jp {name}] 0] 1]
 if {($jpname == "FVTProvider")} {
  puts "Removing old JDBC Provider"
  $AdminConfig remove $jp
  $AdminConfig save
 }
}


#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.jdbc.DB2XADataSource"
set attrs1 [subst {{classpath $driverClassPath}
{implementationClassName COM.ibm.db2.jdbc.DB2XADataSource}
{name "FVTProvider2"}
 {description "DB2 JDBC Provider"}}]
set provider1 [$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 $databaseName1}}]
$AdminConfig create J2EEResourceProperty $propSet1 $attrs3

set attrs10 [subst {{jndiName jdbc/fvtDS_1}
{statementCacheSize 10}
{datasourceHelperClassname
 com.ibm.websphere.rsadapter.DB2DataStoreHelper}
{relationalResourceAdapter $rsadapter}
```

```
  {authMechanismPreference "BASIC_PASSWORD"}
  {authDataAlias  $aliasName}}]
$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} }

#Now lets create the 4.0 data sources..
puts "Creating the 4.0 datasource fvtDS_3"
set ds3 [$AdminConfig create WAS40DataSource $provider1
{{name fvtDS_3} {description "FVT 4.0 DataSource"}}]

#Set the properties on the data source
set propSet3
 [$AdminConfig create J2EEResourcePropertySet $ds3 {}]

#These attributes should be the same as fvtDS_1
set attrs4 [subst {{name user} {type java.lang.String}
{value $defaultUser1}}]
set attrs5 [subst {{name password} {type java.lang.String}
 {value $defaultPassword1}}]
$AdminConfig create J2EEResourceProperty $propSet3 $attrs3
$AdminConfig create J2EEResourceProperty $propSet3 $attrs4
$AdminConfig create J2EEResourceProperty $propSet3 $attrs5
set attrs10 [subst {{jndiName jdbc/fvtDS_3}
{databaseName $databaseName1}}]
$AdminConfig modify $ds3 $attrs10

$AdminConfig create WAS40ConnectionPool $ds3 {{orphanTimeout 3000}
 {connectionTimeout 1000} {minimumPoolSize 1} {maximumPoolSize 10}
 {idleTimeout 2000}}

#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 $aliasName}}]
set cf1 [$AdminConfig create CMPConnectorFactory $rsadapter $attrs12]

#Set the properties for the data source.
$AdminConfig create MappingModule $cf1
 {{mappingConfigAlias "DefaultPrincipalMapping"}
 {authDataAlias "alias1"}}

$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 version 5.0.1. 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: The WebSphere Application Server 5.0.1 release 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. Before this release, data source connections could be tested from wsadmin two ways. The 5.0.1 PTF adds a third option.

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 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.

Steps for this task

1. Ensure that a valid *Authorization Data alias* exists and is used 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 *${DB2_JDBC_DRIVER_PATH}/db2java.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.

Steps for this task
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.

```java
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;

/**
 * Resource adapter test program to
 * make sure that the MBean interfaces work.
 * Following interfaces are tested
 *
 *  -getPropertiesForDataSource()
 *  -reload()
 *  -testConnectionToDataSource()
 *
 *
 * We need following to run
 * set classpath=%classpath%;D:\WebSphere\AppServer\lib\wsexception.jar;
D:\WebSphere\AppServer\lib\wasjmx.jar;D:\$WAS_HOME\lib\wasx.jar
 *
 */
public class testDS {

 String port = "8880";
 String host = "localhost";
 final static boolean verbose = true;



    /**
     * Main method.
     *
     * @param args    DataBase classpath, DataSource name
     */
    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();
//ex.getCause().printStackTrace();
       } catch (Exception ex) {
          System.out.println("Caught this " + ex );
          ex.printStackTrace();
       }
    }


    /**
     * This method tests the DataSourceCfgHelper Mbean.
     *
     * @param args
     * @exception Exception
     */
    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);
        }

//System.out.println("Connected to the application server" + handle);

/*************************************************************************/
/**     Call the Mbean to get the data source properties             */
/*************************************************************************/
  String dsClassName = "COM.ibm.db2.jdbc.DB2XADataSource";
        String providerLibPath = "D:/SQLLIB/java/db2java.zip";
        String[] signature = { "java.lang.String", "java.lang.String"};
        Object[] params = { dsClassName, providerLibPath};
        Object result = null;

   if (verbose) {
        System.out.println("Calling getPropertiesForDataSource() for "
                        + dsClassName + "\n");
        }
        try {
   // get the properties
    result = adminClient.invoke(handle,
             "getPropertiesForDataSource", params, signature);
      } 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);
                }

    // Pretty print what we found
            Iterator propIterator = ((List)result).iterator();
            System.out.println(format("Name",21)+ "|" + format("Default Value",34)
                                + "|" + format("Type",17) +"|Reqd");
            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"));
                }
                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()");
            }
/***************************************************************************/
/** For the following to work you must give all permission to the       */
/** database jar/zip files else you will see this exception:            */
/**  DSRA8040W: Failed to connect to the DataSource.  Encountered       */
/** java.lang.ExceptionInInitializerError:                              */
/** Chainned java.security.AccessControlException:                      */
/** access denied (java.util.PropertyPermission ibm.db2.instance.path read). */
 /**                                                                     */
/**  e.g.                                                                */
/**  Put this in server.policy                                          */
/**                                                                      */
/**     grant codeBase "file:D:/sqllib/java/db2java.zip" {              */
 /**        permission java.security.AllPermission;                      */
 /**     };                                                              */
/***************************************************************************/


/***************************************************************************/
/**  Start to test the connection to the database                       */
/***************************************************************************/

            if (verbose) {
                System.out.println("\nTesting connection
                        + to the database using " + dsClassName);
```

```
            }

            String user = "db2admin";
            String password = "db2admin";
            Properties props = new Properties();
            props.setProperty("databaseName", "section");

// There are two ways to pass the locale: In WS 5.0, you can only pass in the
// language and the country in a String format. In WS 5.0.1 release,
// you can also pass
// in a Locale object.
// String[] signature2 =
//     { "java.lang.String", "java.lang.String", "java.lang.String",
//   "java.util.Properties", "java.lang.String","java.util.Locale"};
// Object[] params2 = { dsClassName, user, password,props ,
//       providerLibPath, Locale.US};

         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");
       Object[] params2 = { dsClassName, user,
           password,props ,providerLibPath, "EN", "US"};

         try {

         result2 = adminClient.invoke(handle,
                  "testConnectionToDataSource",
                params2, signature2);
         } 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);
           ex.printStackTrace();
         }

         if (result2 != null) {
            System.out.println
                ("ERROR Result= " + result2);
         } else if (verbose) {
            System.out.println
                ("OK testConnectionToDataSource()");
         }

      } 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.

```
/**
 * Description
 * Resource adapter test program to make
 *  sure that the MBean interfaces work.
 * Following interfaces are tested
 *
 *  ---  testConnection()
 *
 *
 * We need following to run
 * C:\src>java -Djava.ext.dirs=C:\WebSphere\AppServer\lib;
 * C:\WebSphere\AppServer\java\jre\lib\ext testDSGUI
 * must include jre for log.jar and mail.jar, else get cass not found exception
 *
 *
 */

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;
```

```
import com.ibm.ws.rsadapter.exceptions.DataStoreAdapterException;

public class testDSGUI {

//Use port 8880 for base installation or port 8879 for ND installation
String port = "8880";
// String port = "8879";
String host = "localhost";
final static boolean verbose = true;

// eg a configuration ID for 5.0 DataSource
// declared at the node level for base
private static final String resURI =
//"cells/cat/nodes/cat:resources.xml#DataSource_1";

// eg a 4.0 DataSource declared at the node level for base
//     private static final String resURI =
//   "cells/cat/nodes/cat:resources.xml#WAS40DataSource_1";

// eg Cloudscape DataSource declared at the server level for base
//private static final String resURI =
//  "cells/cat/nodes/cat/servers/server1/resources.xml#DataSource_6";

// eg node level DataSource for ND
//private static final String resURI =
//  "cells/catNetwork/nodes/cat:resources.xml#DataSource_1";

// eg server level DataSource for ND
//private static final String resURI =
 "cells/catNetwork/nodes/cat/servers/server1:resources.xml#DataSource_4";

// eg cell level DataSource for ND
//private static final String resURI =
"cells/catNetwork:resources.xml#DataSource_1";

 public static void main(String[] args) {
  testDSGUI cds = new testDSGUI();
  cds.run(args);
 }

/**
 * This method tests the ResourceMbean.
 *
 * @param args
 * @exception Exception
 */
 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
    // eg for a Baseinstallation this is enough
        ObjectName queryName = new
                  ObjectName("WebSphere:type=DataSourceCfgHelper,*");

    // for ND you need to specify which
   // node/process you would like to test from
   // eg run in the server
   //ObjectName queryName = new
     OjectName("WebSphere:cell=catNetwork,node=
          cat,process=server1,type=DataSourceCfgHelper,*");
   // eg run in the node agent
  //ObjectName queryName = new
    ObjectName("WebSphere:cell=catNetwork,node=
        cat,process=nodeagent,type=DataSourceCfgHelper,*");
  // eg run in the Deployment Manager
  //ObjectName queryName = new
    ObjectName("WebSphere:cell=catNetwork,node=
     catManager,process=dmgr,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;

/***********************************************************************/
 /** For the following to work you must give all permission to the database  */
 /** jar/zip files else you will see this exception:                  */
 /** DSRA8040W: Failed to connect to the DataSource.  Encountered      */
 /** java.lang.ExceptionInInitializerError:                            */
 /** Chained java.security.AccessControlException:                     */
 /** access denied (java.util.PropertyPermission ibm.db2.instance.path read). */
 /**                                                                   */
 /** e.g.                                                              */
 /** Put this in server.policy                                         */
 /**                                                                  */
 /** grant codeBase "file:D:/sqllib/java/db2java.zip" {               */
```

```
/**  permission java.security.AllPermission;                        */
/** };                                                                 */
/**                                                                    */
/*********************************************************************/

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();
  // this 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= " + 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);
  }
 }
}
```

Configuring Java 2 Connector connection factories in the administrative

console

<u>Steps for this task</u>

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 create connection pool settings.

To view this administrative console page, click **Resources** > **JDBC Providers** > *JDBC_provider* > **Data Sources** > *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

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.

You can use the Tivoli Performance Viewer to find the optimal number of connections in a pool. If the number of concurrent waiters is greater than 0, but the CPU load is not close to 100%, consider increasing the connection pool size. If the Percent Used value is consistently low under normal workload, consider decreasing the number of connections in the pool.

Data type	Integer
Default	10
Range	0 to max int

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*.
- If you set the purge policy for this data source object to 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.

- If you set the purge policy for this data source object to FailingConnectionOnly, only the connection that caused the *StaleConnectionException* is closed. While 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 Version 4.0 data sources always have a purge policy of **EntirePool**; JCA data sources have the options of **EntirePool** or **FailingConnectionOnly**.

Data type	String
Default	EntirePool
Range	EntirePool or FailingConnectionOnly

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

Steps for this task

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.

Choose from aliases defined under **Security** > **JAAS Configuration** > **J2C 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.

Choose from aliases defined under **Security** > **JAAS Configuration** > **J2C 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)
```

Security of lookups with component managed authentication

External Java clients (stand alone clients or servers from other cells) with Java Naming and Directory Interface (JNDI) access can look up a Java 2 Connector (J2C) resource such as a data source or Java Message Service (JMS) queue. However, they are not permitted to take advantage of the component managed *authentication alias* defined on the resource. This alias is a default value used when the *user* and *password* are not supplied on the *getConnection* call. Therefore, if an external client needs to get a connection, it must assume responsibility for the authentication data. It can either "Passing user and password on the getConnection call", or it can "Using the no-argument getConnection() call" and define the alias with the corresponding *user* and *password* in a file on each system on which the client is expected to run.

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.

Passing user and password on the getConnection call

If the *user* and *password* are supplied on the *getConnection* call by a client running outside the WebSphere Application Server process, the connection factory or datasource looked up for obtaining connections must not have a component managed authentication alias defined on it.

Steps for this task

1. Make sure the connection factory or datasource does not have a component managed authentication alias defined on it. See "Configuring Java 2 Connector connection factories in the administrative console" for details.

Using the no-argument getConnection() call

If a client running outside the WebSphere Application Server process does not pass the *user* and *password* on the *getConnection* call, you must define a text file on each system on which the client runs. This file, whose name is defined by the system property *WAS_AuthDataFile*, has records of the form:

```
alias1,user1,pw1
```

For each unique component managed *authentication alias* defined on a connection factory or datasource and used by the remote client, you must define the alias and corresponding *user* and *password* to match the definition of the alias in the WebSphere Application Server.

To create the remote client text file:

Steps for this task

1. Run *%WAS_HOME%\bin\EncAuthDataFile.bat* to encode the password as follows: *EncAuthDataFile c:\client0.txt c:\client1.txt*

   Be sure to fully qualify the filenames. The file *client1.txt* for the above alias, user, and password should now look like this:

   ```
   alias1,user1,{xor}Lyhu
   ```

2. You must have the client specify the location of the text file in the system property *WAS_AuthDataFile*.

   This is done on the *launchClient* invocation by passing the following, for example:

   ```
   %WAS_HOME\bin\launchClient.bat myApp.ear -CCDWAS_AuthDataFile=c:\client1.txt
   ```

Disabling lookup security

By default, all lookups are secure as described in "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.

Steps for this task

1. Edit *%WAS_HOME%\properties\j2c.properties* (or *$WAS_HOME/properties/j2c.properties* on UNIX 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 Application Assembly Tool (AAT) 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 Application Assembly Tool

Steps for this task

1. Assemble your application client module as described in Assembling Application Client Modules. When you reach the step for creating a resource reference, continue with these steps.
2. Create a new resource reference.
3. Do the following on the **New Resource Reference General Tab**:
   a. 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*.

b. Use the pull-down in the **Type** field and select *javax.sql.DataSource* for JDBC connections.

c. Set the **authentication** field to *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), then set this field to *Container*.

d. Ignore the **sharing scope** setting, it is unused in an application client resource reference. All WebSphere Application Client resources are currently unsharable.

4. Ignore the **JNDI name** field on the *Binding Tab*. It is not used in an application client resource reference. The required information is provided in the ACRCT.

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.

Steps for this task

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 of the General Tab of the Application Assembly Tool. 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 General tab of the Application Assembly Tool is set to *Container*.

Notes: 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.

Configuring Cloudscape Version 5.1

When you install WebSphere Application Server, Cloudscape Version 5.0 is installed by default. To install Cloudscape Version 5.1, run the `Cloudscape51.bat` file on Windows or `Cloudscape51.sh` file on UNIX platforms. You can find this file in the `WAS_HOME/cloudscape51` directory.

The installer replaces the Cloudscape Version 5.0 files with the files from the `WAS_HOME/cloudscape51` directory. It also replaces the Cloudscape `.jar` files, which are located in the `WAS_HOME/lib` directory, with Cloudscape Version 5.1`.jar` files. The installer launches a WebSphere specific Cloudscape installer and defines some Websphere environment variables. The installer also creates an uninstaller, which can undo these changes. You cannot use any Version 4.0 data sources with Cloudscape. Cloudscape Version 5.0 is *deprecated* in WebSphere Application Server Version 5.0.2.

Note: After you install Cloudscape Version 5.1, you cannot run or have any Cloudscape Version 5.0 defined data sources. If you want to run Cloudscape Version 5.0, you must first uninstall Cloudscape Version 5.1, and then remove Cloudscape Version 5.1 data sources or change them to Cloudscape Version 5.0 data sources.

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:
  - inablility 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.

Steps for this task

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/cloudscapeCommon 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 http://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/cloudscape51/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/cloudscape51/bin/networkserver directory. These tools are:
   - sysinfo
   - cview
   - ijf
   - 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.

Cloudscape Version 5.1 post install instructions

After installing Cloudscape Version 5.1, you must complete the following steps before you can access the database.

Steps for this task

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. Use the administrative console to add the following WebSphere Application Server environment variables, if they do not exist, or to modify the ones that do exist:

   - `DB2UNIVERSAL_JDBC_DRIVER_PATH` Enter the location of the **db2jcc.jar** file as the value for this environment variable. For example, if your **db2jcc.jar** file is located in the java directory on the C drive, enter `c:\sqllib\java`.

     If you install DB2 and you run WebSphere Application Server with DB2, use the **db2jcc.jar** file from DB2.

     If you run a Type 4 universal driver and DB2 is not installed, as for example a client machine, copy the **db2jcc.jar** file from the DB2 machine to the client machine, and set the value of this environment variable to the location of the file on the client machine.

     If DB2 is not installed and you run Websphere Application Server with another database, as for example Oracle, Sybase or Informix, set the value of this variable to the location of the **db2jcc.jar** that ships with WebSphere Application Server, which is `WAS_HOME/universalDriverforCloudscape`.

3. 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`.

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 infomation:

- DB2
- DB2 UDB for iSeries
- DB2/390
- Cloudscape
- Informix
- Sybase
- Oracle
- MS SQL Server

1. **5.0.2** **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 FixPak 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* environment variable to point to the **db2jcc.jar** file. See the Cloudscape section for more information on the *DB2UNIVERSAL_JDBC_DRIVER_PATH* environment variable.
   - **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.
   - **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
     - DB2/390

– SQLDS

You must use the right license jar file to access a specific database backend.

Requires `DataStoreHelper` class:
`com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper`
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. I f 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 Datatbase 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. **5.0.2** **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 FixPak 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.
- **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.
- **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
  – DB2/390
  – SQLDS

You must use the right license jar file to access a specific database backend.

Requires `DataStoreHelper` class:

com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper
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
   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
   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
   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`
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/

For information on configuring WebSphere Application Server for DB2 access, see
the Configuring DB2 article.

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`
   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`
   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`
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`
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`
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`
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/.

1. **DB2/390 JDBC Provider**

   The DB2/390 JDBC Provider provides remote JDBC access to DB2/390. The DB2 390 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 390 Local JDBC Provider (RRS) on local z/OS systems. Do not use this JDBC provider to create DB2/390 data sources. To create DB2/390 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`.

   DB2/390 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 properties:

   - **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

2. **DB2/390 JDBC Provider (XA)**

   The DB2/390 JDBC Provider provides remote XA-compliant JDBC access to DB2/390. The DB2 390 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 390 Local JDBC Provider (RRS) on local z/OS systems. Do not use this JDBC provider to create DB2/390 XA data sources. To create DB2/390 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`.

   DB2/390 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 properties:

   - **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

3. **DB2 390 Local JDBC Provider (RRS)**

   The DB2/390 JDBC Provider provides local JDBC access to DB2/390. The DB2 390 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.

   DB2 390 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`
Requires properties:

- **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

For more information on DB2/390, visit the DB2 Web site at:
http://www.ibm.com/software/data/db2/.

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/cloudscape51/lib` directory).
   Requires `DataStoreHelper` class:

   `com.ibm.websphere.rsadapter.CloudscapeDataStoreHelper`

   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/cloudscape51/lib` directory).
   Requires `DataStoreHelper` class:

   `com.ibm.websphere.rsadapter.CloudscapeDataStoreHelper`
   Requires properties:

   - **databaseName** The name of the database from which the data source obtains connections. Example: *Sample* or *c:\sample*.

3. **Cloudscape JDBC Provider Version 5.0 — *Deprecated***

   This JDBC driver provider is the same as Cloudscape JDBC Provider. This JDBC driver provider is *deprecated*. Use the Cloudscape JDBC Provider instead of this one. You cannot use any Version 4.0 data sources with Cloudscape.

   Cloudscape JDBC Provider 5.0 supports one phase data source:

   `com.ibm.db2j.jdbc.DB2jConnectionPoolDataSource`
   Requires JDBC driver files: **db2j.jar**. (This file ships with WebSphere Application Server. You can find this file in the `WAS_HOME/cloudscape50/lib` directory).
   Requires `DataStoreHelper` class:

   `com.ibm.websphere.rsadapter.CloudscapeDataStoreHelper`
   Requires properties:

   - **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

4. **Cloudscape JDBC Provider 5.0 (XA)** — *Deprecated*

This JDBC driver provider is the same as the Cloudscape JDBC Provider (XA). This JDBC driver provider is *deprecated*. Use the Cloudscape JDBC Provider (XA) instead of this one. You cannot use any Version 4.0 data sources with Cloudscape.

Cloudscape JDBC Provider 5.0 (XA) supports two phase data source:

`com.ibm.db2j.jdbc.DB2jXADataSource`

Requires JDBC driver files: **db2j.jar** (This file ships with WebSphere Application Server. You can find this file in the `WAS_HOME/cloudscape50/lib` directory).

Requires `DataStoreHelper` class:

`com.ibm.websphere.rsadapter.CloudscapeDataStoreHelper`

Requires properties:

- **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

5. **5.0.2** **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, and that is now available in WebSphere Application Server Version 5.0.2. 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. If DB2 is not installed, and you do not plan to run DB2 and Cloudscape together, copy the **db2jcc.jar** file to the WebSphere Application Server machine, and set the *DB2UNIVERSAL_JDBC_DRIVER_PATH* environment variable to point to the **db2jcc.jar** file. This guarantees that only one version of this `.jar` file is loaded in the WebSphere Application Server environment. If you upgrade or install any fixes on DB2, you must also update this file on your client machine. If you are not using DB2 but another database such as Sybase, Oracle, or Informix, access a copy of the file from the `WAS_HOME/universalDriverforCloudscape` directory, and set the *DB2UNIVERSAL_JDBC_DRIVER_PATH* environment variable to point to this file. If you are not running DB2, use the **db2jcc.jar** file from the `WAS_HOME/universalDriverforCloudscape` directory.

  Note: When you define two datasources, one for Cloudscape and another for DB2, you are running DB2 and Cloudscape together.

- **db2jcc_license_c.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.

Requires `DataStoreHelper` class:

`com.ibm.websphere.rsadapter.CloudscapeNetworkServerDataStoreHelper`
Note: The administrative console incorrectly lists the DB2UniversalDataStoreHelper as the default value for the `DataStoreHelper` class. See the WebSphere Application Server Version 5.0.2 Release Notes for more information. 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 properties:

- **databaseName** The name of the database and the list of Cloudscape attibutes. 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 Achitecture (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/

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 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 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/.

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 properties:

   - **serverName** The name of the database server. Example: *myserver.mydomain.com*.
   - **5.0.1** **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 properties:

   - **serverName** The name of the database server. Example: *myserver.mydomain.com*
   - **5.0.1** **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 [ ]. 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 properties:

- **serverName** The name of the database server. Example:
  myserver.mydomain.com

- **5.0.1** **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 [ ]. 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 properties:

   - **serverName** The name of the database server. Example:
     myserver.mydomain.com

   - **5.0.1** **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/.

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: **classes12.zip**. (Note: If you require Oracle trace,
   use **classes12_g.zip**.)
   Requires DataStoreHelper class:

   com.ibm.websphere.rsadapter.OracleDataStoreHelper
   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: **classes12.zip**. (Note: If you require Oracle trace, use **classes12_g.zip**.)
Requires DataStoreHelper class:

com.ibm.websphere.rsadapter.OracleDataStoreHelper
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 [ ] 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: **classes12.zip**. (Note: If you require Oracle trace, use **classes12_g.zip**.)
   Requires DataStoreHelper class:

   com.ibm.websphere.rsadapter.OracleDataStoreHelper
   Requires properties:

   - **URL**

type	*java.lang.String*
value	*" "*
description	This is a required property. 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 [ ] 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: **classes12.zip**. (Note: If you require Oracle trace, use **classes12_g.zip**.)
   Requires DataStoreHelper class:

   com.ibm.websphere.rsadapter.OracleDataStoreHelper
   Requires properties:

   - **URL**

type	*java.lang.String*
value	*" "*

description	This is a required property. 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 [ ] 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: `classes12.zip`. (Note: If you require Oracle trace, use `classes12_g.zip`.)
   Requires DataStoreHelper class:

   `com.ibm.websphere.rsadapter.OracleDataStoreHelper`
   Requires properties:

   • **URL**

type	*java.lang.String*
value	*" "*
description	This is a required property. 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 [ ] 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: `classes12.zip`. (Note: If you require Oracle trace, use `classes12_g.zip`.)
   Requires DataStoreHelper class:

   `com.ibm.websphere.rsadapter.OracleDataStoreHelper`
   Requires properties:

   • **URL**

type	*java.lang.String*
value	*" "*

description	This is a required property. 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/

SQLServer

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 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 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 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 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
at ftp://ftp.software.ibm.com/software/websphere/info/
tools/DataDirect/install.html#JTA
```

You can download the latest patches and upgrades to the WebSphere embedded ConnectJDBC drivers from the following FTP site (split for publication):

```
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 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 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 (split for publication):

   `How to install SequeLink Server from the WebSphere CD`

   available at:

   `ftp://ftp.software.ibm.com/software/websphere/`
   `info/tools/DataDirect/install.html#SLServer`

   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 (split for publication):

   `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. **5.0.1** **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 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. **5.0.1** **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 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

WebSphere Business Integration Adapters

WebSphere Application Server Enterprise provides support for the suite of WebSphere Business Integration Adapters (WBIA), which include adapters for SAP R3, PeopleSoft, Siebel, Oracle applications, SWIFT, and other applications. These adapters connect applications, technologies, data sources, and mainframes in local and remote Enterprise Information Systems. See the WebSphere Business Integration Adapters InfoCenter for more information.

WebSphere Business Integration Adapters tools generate Web Services Description Language (WSDL) files for the services offered by an adapter and XML Schema Definition (XSD) files for business objects associated with the adapter. You can use the WebSphere Studio Application Developer Integration Edition (WSADIE) enterprise services tools on these files, to deploy the services to the WebSphere Application Server Enterprise or to generate WebSphere Application Server Enterprise applications, which can then invoke the services. See the WebSphere Studio Application Developer Integration Edition documentation for information on enterprise services.

Access the services through the JMS messaging system using WebSphere MQ. A generated service supports the following four interaction patterns between WebSphere Application Server and WebSphere Business Integration Adapter:

- WebSphere Application Server initiated:
  - one-way
  - request/response
- WebSphere Business Integration Adapter or Enterprise Information Systems (EIS) initiated:
  - one-way
  - request/response

You can manage the WebSphere Business Integration Adapter through the Websphere Application Server Enterprise administrative console and through the wsadmin scripting environment. This management function is available if you select the **Additional Enterprise Extensions** feature when you install WebSphere Application Server Enterprise.

The WBIA feature provides six administrative commands for changing the state of the WebSphere Business Integration Adapter. See the Managing the WebSphere Business Integration Adapters article for a description of the administrative commands. The WBIA wsadmin commands article provides WBIA wsadmin command examples.

Enabling WebSphere Business Integration Adapter sessions

Before starting a WebSphere Business Integration Adapter (WBIA) session, you must first ensure that the WebSphere MQ JMS Provider is configured correctly. You do not have to add any new information to enable WBIA support. Install and configure the Websphere MQ JMS Provider in the usual manner as documented in article, (Installing WebSphere MQ as the JMS Provider).

To install and configure WBIA session support, complete the following steps:

Steps for this task
1. (Create a new WebSphere MQ Queue Connection Factory).
2. Update the following fields in the new WebSphere MQ Queue Connection Factory to enable WBIA support:

   Name
   > This is a required field. Enter a user-defined value.

   JNDI Name
   > This is a required field. Enter a user-defined value.

Description
>   This is an optional field.

Category
>   This is an optional field.

Component-managed Authentication Alias
>   Enter a value for this field if you have Global Security enabled on your
>   WebSphere server, and you want to assign a password and userid
>   when you make the connection to the MQ Queue Connection Factory.
>   If you use this field, you must first create an alias, which then appears
>   in this drop-down list. You create the alias by traversing the following
>   path in the administrative console:
>
>   **Security** > **JAAS Configuration** > **J2C Authentication Data** > **New**
>   Then enter an alias name, userid, and password. Select an existing
>   Component-managed Authentication Alias if you want the WBIA
>   management function to use the userid and password specified in the
>   alias when using this WebSphere MQ Queue Connection Factory.

Container-managed Authentication Alias
>   This field does not affect authentication for the WBIA resource.

Update the remaining fields in the WebSphere MQ Queue Connection Factory
in the same manner as for general JMS support.

3. (Create new WebSphere MQ Queue destinations).

   You must create two new MQ Queue destinations, one for the WBIA AdminIn
   Queue and another for the WBIA AdminOut Queue. These two queues must be
   unique for each WBIA resource. These queues process the adminstrative
   commands that flow between WebSphere Application Server and the
   WebSphere Business Integration Adapter.

   Do not confuse the AdminIn and AdminOut queues when you create the
   destinations. The WBIA resource represents the remote WBIA connector process
   so the names you specify must reflect the WBIA adapter. The AdminIn queue is
   the queue on which WebSphere Application Server places outgoing
   adminstrative messages that are read by the WBIA adapter. The AdminOut
   queue is the queue on which WebSphere Application Server listens for
   responses to the administrative messages.

4. Update the following fields in the new WebSphere MQ Queue destinations.

Name
>   This is a required field. Enter a user-defined value.

JNDI Name
>   This is a required field. Enter a user-defined value.

Persistence
>   Do not change the default setting, which is **APPLICATION DEFINED**.

Priority
>   Do not change the default setting, which is **APPLICATION DEFINED**.

Specified Priority
>   If the **Priority** field is set to **APPLICATION DEFINED**, then the value
>   in this field does not apply.

Expiry Do not change the default setting, which is **APPLICATION DEFINED**.

Specified Expiry

If the **Expiry** field is set to **APPLICATION DEFINED**, then the value in this field does not apply.

Base Queue Name

This is a required field. Enter the name of the WebSphere MQ queue.

Update the remaining fields in the WebSphere MQ Queue destinations in the same manner as for general JMS support.

5. Configure the WBIA resource.

   a. Expand the **Resources** category in the administrative console.

   b. Select **WebSphere Business Integration Adapter** and click **New** to configure the resource.

   c. Complete the required fields, and select the applicable WebSphere MQ Queue Connection Factory and WebSphere MQ Destinations from the drop-down lists.

   After you click **Apply** or **OK**, WebSphere Application Server creates a new WBIA resource.

6. Start the WBIA service.

WebSphere Business Integration Adapters collection

Use this page to select a WebSphere Business Integration Adapter (WBIA) resource , which represents a mechanism that allows you to communicate through the Java Messaging Service (JMS) with the connector component of the WebSphere Business Integration Adapter. The connector component processes administrative commands that control the state of the connector.

To view this administrative console page, click **Resources** > **WebSphere Business Integration Adapters**.

Scope

Specifies the level to which this WebSphere Business Integration Adapters resource is visible — the cell, node, or server level.

WebSphere Application Server Version 5.0.x 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 WBIA resources. 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 WBIA 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 WBIA 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 WBIA resource.

A string with no spaces meant to be a meaningful text identifier for the WBIA resource.

Data type   String

Description
Specifies a text description of the WBIA resource.

A free-form text string to describe the WBIA resource and its purpose.

Data type   String

Category
An optional field that is used to classify or group the WBIA resources.

Data type   String

Queue Connection Factory JNDI Name
Specifies a list of JNDI names, which are used to create connections to the associated WebSphere Business Integration Adapters JMS destinations.

Data type   Pick-list

WebSphere Business Integration Adapters settings
Use this page to specify settings for a WebSphere Business Integration Adapters resource.

The WebSphere Business Integration Adapters (WBIA) resource represents a mechanism that allows you to communicate through the Java Messaging Service (JMS) with the connector component of the WebSphere Business Integration Adapter. The connector component processes administrative commands that control the state of the connector.

To view this administrative console page, click **Resources** > **WebSphere Business Integration Adapters** > **New**.

Name: Specifies the name of the WBIA resource.

A string with no spaces meant to be a meaningful text identifier for the WBIA resource.

Data type   String

Description: Specifies a text description of the WBIA resource.

A free-form text string to describe the WBIA resource and its purpose.

Data type   String

Category: An optional field that is used to classify or group the WBIA resources.

Data type   String

Queue Connection Factory JNDI Name: Specifies a list of JNDI names, which are used to create connections to the associated WebSphere Business Integration Adapters JMS destinations.

Data type   Pick-list

Administrative Input Queue JNDI Name: Specifies a list of JNDI names for the queues WebSphere Application Server uses to send administrative commands to the Websphere Business Integration Adapter.

Data type   Pick-list

Administrative Output Queue JNDI Name: Specifies a list of JNDI names for the queues WebSphere Application Server uses to receive administrative messages from the Websphere Business Adapter.

Data type   Pick-list

Message Timeout: Specifies the number of seconds WebSphere Application Server waits to receive administrative messages from the Websphere Business Adapter before a timeout occurs for that session. The default value is 30 seconds.

Data type   Seconds

Administering the WebSphere Business Integration Adapter service

WebSphere Business Integration Adapter is a service, which is accessed like the other services that run in WebSphere Application Server and provide support for enterprise applications. In the adminstrative console, you access all the available services from the **Servers** entry.

To access the WBIA service, complete the following steps:

Steps for this task
1. Select **Application Server** > *serverName* > **Additional Properties** > **WBIA Service**.
2. Check the **Startup** box, if you want WebSphere Application Server to start the WBIA service when WebSphere starts.
3. Select the **Manage the WBIA resources** from the **Additional Properties** section.

    The **Manage the WBIA resources** panel displays all the WBIA resources that you configured previously. You can now manage the WBIA resources by clicking on a command. See article, Managing the WebSphere Business

Integration Adapters, for a description of the WBIA commands. The WBIA wsadmin commands article provides WBIA wsadmin command examples.

Examples: Managing Websphere Business Integration Adapters with wsadmin

These examples demonstrate how to manage the WebSphere Business Integration Adapters (WBIA) using the wsadmin scripting environment. To use this environment, you must first locate the instance of the MBean, which is named *WBIAAdminMBean*. This bean has one operation, **executeCommand**, which takes the following arguments:

- WBIA Adapter Factory JNDI name
- name of the command

The following examples describe how to identify the MBean, locate and select configured WBIA resource factories, show and set attributes and variables, and invoke WBIA commands:

- Identify the MBean:

  ```
  set wbiambean [$AdminConfig queryNames *:*,type=WBIAAdminMBean]
  ```

  Example output:

  ```
  wbiambean:platform=common,cell=mycell,
  version=5.02,name=WBIAAdminMBean,mbeanIdentifier=
  WBIAAdminMBean,type=WBIAAdminMBean,node=mynode,process=server1
  ```

- Locate the configured WBIA Adapter resource factories:

  ```
  $AdminConfig list WebSphereBusinessIntegrationAdapterFactory
  ```

  Example output: (shows two configured)

  ```
  MyAdapter1(cells/mycell/nodes/mynode:resources-
  pme502.xml#WebSphereBusinessIntegrationAdapterFactory_1055014328949)
          MyAdapter2(cells/mycell/nodes/mynode:resources-
  pme502.xml#WebSphereBusinessIntegrationAdapterFactory_1055008720284)
  ```

- Select the WBIA Adapter resource factory, and set it to a variable name:

  ```
  set MyAdapter1 MyAdapter1(cells/mycell/nodes/mynode:resources-
  pme502.xml#WebSphereBusinessIntegrationAdapterFactory_1055014328949)
  ```

  Example output:

  ```
  MyAdapter1(cells/mycell/nodes/mynode:resources-
  pme502.xml#WebSphereBusinessIntegrationAdapterFactory_1055014328949)
  ```

- Show the current attribute values to get the WBIA Adapter resource factory JNDI Name:

  ```
  $AdminConfig show $MyAdapter1
  ```

  Example output:

  ```
  {adminInQueueJNDIName jms/AdminInQueue}
          {adminOutQueueJNDIName jms/AdminOutQueue}
          {jndiName wbia/MyAdapter1_1055008683211}
          {messageTimeout 30}
          {name MyAdapter1}
          {provider WebSphereBusinessIntegrationAdapterProvider
      (cells/mycell/nodes/mynode:resources-
      pme502.xml#WebSphereBusinessIntegrationAdapterProvider_1055008720284)}
          {queueConnectionFactoryJNDIName jms/MyAdapter1QCF}
  ```

- Set a variable for the WBIA Adapter resource factory JNDI Name:

  ```
  set MyAdapter1JNDIName [list "wbia/MyAdapter1_1055008683211"]
  ```

Example output:
```
wbia/MyAdapter1_1055008683211
```
- Set a variable for the command:
```
set command [list "status"]
```

Example output:
```
 status
```
- Set an attribute that contains the WBIA Adapter resource factory JNDI Name and the command:
```
set MyAdapter1status [list $MyAdapter1JNDIName $command]
```

Example output:
```
wbia/MyAdapter1_1055008683211 status
```
- Invoke the command:
```
$AdminControl invoke $wbiambean executeCommand $MyAdapter1status
```

Example output if the MyAdapter1 connector is active:
```
WBIA0017I: The status command has been acknowledged as
being received by the MYAdapter1 adapter.  The status is active.
```

Use the same format to invoke all the WBIA commands:
- Status
- Activate
- Deactivate
- Suspend
- Resume
- Shutdown

WebSphere Business Integration Adapters service settings in administrative

console

Use this page to enable or disable the the WebSphere Business Integration Adapters (WBIA) service. This page also provides a link to the **Manage the WBIA resources** console page, which allows you to send administrative commands to eligible WBIA resources.

To view this administrative console page, click **Servers** > **Application Servers** > *serverName* > **WBIA Service**.

Startup
Specifies whether WebSphere Application Server attempts to start this service when the server starts.

Additional Properties, Manage the WBIA resources
Select **Manage the WBIA resources** to reach the adminstrative console page, which allows you to send administrative commands to eligible WBIA adapter resources.

Managing the WebSphere Business Integration Adapters resources

Use the administrative commands on this page to manage the WebSphere Business Integration Adapters (WBIA) resources.

To view this administrative console page, click **Servers** > **Application Servers** > *serverName* > **WBIA Service** > **Manage the WBIA resources**.

The following commands are available for managing WBIA resources:

Status The **Status** command provides the current state of the connector. Possible connector states are:

- active
- inactive
- suspended

Activate

The **Activate** command changes the state of the connector from inactive to active. In the active state, the connector performs both event delivery and request processing.

Deactivate

The **Deactivate** command changes the state of the connector from active to inactive. In the inactive state, the connector process continues to run; however, the connector no longer accepts new requests.

Suspend

The **Suspend** command changes the state of the connector from active to suspended. In the suspended state, the connector does not perform event delivery; however, it continues to accept and process requests.

Resume

The **Resume** command changes the state of the connector from suspended to active. When it returns to the active state, the connector resumes event delivery.

Shutdown

The **Shutdown** command shuts down the connector process. You cannot restart the connector process from the WebSphere Application Server Enterprise administrative console. Instead, you must restart the connector process with startup scripts installed with the WebSphere Business Integration Adapter product.

Use the commands on the WBIA resources. Click on a resource to select it from the list, and then click on a command button to submit that command for the resource. When you select a resource, a detail page describing the configured fields appears. The resource detail page displays in *read only* mode.

Name
Specifies the name of the WBIA resource.

The property value is a string with no spaces, which is meant to be a meaningful text identifier for the WBIA resource.

Data type   String

Description
Specifies a text description of the WBIA resource.

The property value is a free-form text string, which describes the WBIA resource and its purpose.

Data type   String

Category
An optional field that is used to classify or group the WBIA resources.

Data type   String

Message Timeout
Specifies the number of seconds WebSphere Application Server waits to receive administrative messages from the Websphere Business Adapter before a timeout occurs for that session. The default value is 30 seconds.

Data type   Seconds

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.

View links to additional information about:
* Program Specifications
* CMP persistence functions
* Container managed relationships
* Local interfaces
* Resource references
* WebSphere Studio Application Developer (WSAD)
* WebSphere Studio Application Developer Integration Edition (WSADIE)
* WebSphere Version 4.0 InfoCenter
* IBM Cloudscape
* Oracle
* DB2 database software
* Supported hardware, software, and APIs
* **What's new in the Enterprise**
  You can also download the specification itself from this URL.
* **JavaTM 2 Platform, Enterprise Edition (J2EETM)**
* **JavaTM Management Extensions (JMX).**
* **Enterprise JavaBeansTM 2.0 Container-Managed Persistence Example**
* **Enterprise JavaBeansTM 2.0 Container-Managed Persistence Example**.
* **Enterprise JavaBeanTM 2.0 Specification Changes**
* **Accessing Databases from Web Applications**
* **WebSphere Studio Application Developer**
* **WebSphere Studio Application Developer Integration Edition (WSADIE)**
* **IBM WebSphereTM Version 4.0 InfoCenter**
* **IBM Cloudscape**

- **Oracle**
- **DB2**
- **Supported hardware, software, and APIs**

Chapter 4. Resource environment entries

<div/>

Steps for this task
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 phyiscal 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

Chapter 5. Using mail

Before you begin

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: (split
for publication)

```
javax.naming.InitialContext ctx =
new javax.naming.InitialContext();

javax.mail.Session mail_session = (javax.mail.Session)
ctx.lookup("java:comp/env/mail/MailSession3");
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.

Steps for this task

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.

   Usage scenarioIn 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 Application Assembly Tool (AAT) shipped with WebSphere Application Server. To learn more about using AAT see the InfoCenter article Starting the Application Assembly Tool.

   <u>Usage scenario</u>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.

   <u>Usage scenario</u>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.

   f. Enable debugger for a mail session.

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:

Steps for this task
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**.

What to do next

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.

Enabling debugger for a mail session

At times you might need to debug a JavaMail application. One option you can use is turning on the JavaMail debugging feature. With this option, the JavaMail API prints interactions with the mail servers, as well as the properties of the mail session, to the stdout output stream, which is redirected to the SystemOut.log file for the specific application server.

The mail debug feature functions, on a per session basis. To enable the JavaMail debugging feature:

Steps for this task
1. Open the administrative console.
2. Click **Resources**>**Mail Providers**>*mail_session*> **Mail Session**>*mail session*.
3. Click **Debug**.

   Debug is enabled for just that session.
4. Click **Apply** or **OK**.

A sample of the JavaMail debugging output is as follows:

```
DEBUG: not loading system providers in <java.home>/lib
DEBUG  not loading optional custom providers file:
            /META-INF/javamail.providers
DEBUG: successfully loaded default providers

DEBUG  Tables of loaded providers
DEBUG: Providers listed by Class Name:
{com.sun.mail.smtp.SMTPTransport=
javax.mail.Provider[TRANSPORT,smtp,com.sun.mail.smtp.SMTPTransport,Sun
Microsystems, Inc],
com.sun.mail.imap.IMAPStore=
javax.mail.Provider[STORE,imap,com.sun.mail.imap.IMAPStore,Sun
Microsystems, Inc],
com.sun.mail.pop3.POP3Store=
javax.mail.Provider[STORE,pop3,com.sun.mail.pop3.POP3Store,Sun
Microsystems, Inc]}
DEBUG:  Providers Listed By Protocol:
{imap=javax.mail.Provider[STORE,imap,com.sun.mail.imap.IMAPStore,
Sun Microsystems, Inc],
 pop3=
javax.mail.Provider[STORE,pop3,com.sun.mail.pop3.POP3Store,Sun
Microsystems, Inc],
 smtp=
javax.mail.Provider[TRANSPORT,smtp,com.sun.mail.smtp.SMTPTransport,Sun
Microsystems, Inc]}
DEBUG: not loading optional address map file:
          /META-INF/javamail.address.map
*** In SessionFactory.getObjectInstance,
  The default SessionAuthenticator is based on:
   store_user = john_smith
   store_pw = abcdef
*** In SessionFactory.getObjectInstance, parameters in the new session:
    mail.store.protocol="imap"
    mail.transport.protocol="smtp"
    mail.imap.user="john_smith"
    mail.smtp.host="smtp.coldmail.com"
```

```
        mail.debug="true"
        ws.store.password="abcdef"
        mail.from="john_smith@coldmail.com"
        mail.smtp.class="com.sun.mail.smtp.SMTPTransport"
        mail.imap.class="com.sun.mail.imap.IMAPStore"
        mail.imap.host="coldmail.com"
DEBUG: mail.smtp.class property exists and points to
            com.sun.mail.smtp.SMTPTransport
DEBUG SMTP: useEhlo true, useAuth false
DEBUG: SMTPTransport trying to connect to host
        "smtp.coldmail.com", port 25

javax.mail.SendFailedException: Sending failed;
    nested exception is:
    javax.mail.MessagingException: Unknown SMTP host: smtp.coldmail.com;
    nested exception is
    java.net.UnknownHostException: smtp.coldmail.com
    at javax.mail.Transport.send0(Transport.java:219)
    at javax.mail.Transport.send(Transport.java:81)
     at ws.mailfvt.SendSaveTestCore.runAll(SendSaveTestCore.java:48)
     at testers.AnyTester.main(AnyTester.java:130)
```

This sample output illustrates a connection failure to a Simple Mail Transfer
Protocol (SMTP) server because a fictitious name, smtp.coldmail.com, is specified
as the server name.

The following are tips on how to read the debugging output:
- The lines headed by *DEBUG* are printed by the JavaMail run-time, while the two
  lines headed by *** are printed by the WebSphere environment run-time.
- The first two lines say that some configuration files are skipped. At run-time the
  JavaMail API attempts to load a number of configuration files from different
  locations. All those files are not required. If a required file cannot be accessed,
  the JavaMail API throws an exception. In this sample, there is no exception and
  the third line announces that default providers are loaded.
- The next few lines, headed by either *Providers listed by Class Name* or *Providers
  Listed by Protocols*, show the protocol providers loaded. The three providers listed
  here are the default protocol providers that come under the WebSphere built-in
  mail provider, for protocols SMTP, IMAP, and POP3, respectively. If you have
  installed special protocol providers (or, in JavaMail terminology, service
  providers) and these providers are used in the current mail session, you should
  see them listed here with the default providers. If you do not see the special
  protocol providers, there is a problem.
- The two lines headed by *** and the few lines below them are printed by the
  WebSphere environment to show the properties used to configure the current
  mail session. Although these properties are listed by their internal name rather
  than that used in the administrative console interface, it is not difficult to
  establish the similarities between them. For example, the property
  mail.store.protocol corresponds to the Protocol Name property in the Store Access
  section of the mail session configuration page. Make sure to review the listed
  properties and values to make sure they correspond.
- The few lines above the exception stack show the JavaMail activities when
  sending a message. First, the JavaMail API recognizes the transport protocol is
  set to SMTP and the provider com.sun.mail.smtp.SMTPTransport exists. Next, the
  parameters used by SMTP, useEhlo and useAuth, are shown. Finally, the log
  shows the SMTP provider trying to connect to the mail server
  smtp.coldmail.com. Listed next is the exception stack. By now it should occur to
  us that the specified mail server either does not exist or is not functioning.

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 InfoCenter 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.

<p/>

Mail migration tip

Syntax

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: (split for publication)

```
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";
};
```

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${/}activation.jar", "read";
permission java.io.FilePermission
"${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.

- **JavaMail documentation**
- **JavaMail 1.2 API documentation (Sun's javadoc)**

Chapter 6. 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.

Steps for this task

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.

View links to additional information about:

*   Programming specifications
*   **W3C Architecture - Naming and Addressing: URIs, URLs**.
*   [ ]

Notices

This information was developed for products and services offered in the U.S.A. IBM might not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

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
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Department LZKS
11400 Burnet Road
Austin, TX 78758
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written.

These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these

programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

AIX
CICS
Crossworlds
DB2
IBM
Informix
iSeries
MQSeries
OS/390
Redbooks
SupportPac
VisualAge
WebSphere
zSeries

Java, JavaBeans, 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.

IBM ®

Printed in USA