Following our example through, using the JMS queue defined in “JMS queue configuration” on page 539, we know that the BankJMSQueue object was bound into the JNDI name space with the name jms/BankJMSQueue. This JMS queue object maps on to the BankJSQueue on the SamplesBus service integration bus.

Therefore, if a message-driven bean is associated with this JMS activation specification, it would be invoked when messages arrived at the BankJSQueue destination on the SamplesBus. See Figure 10-35.

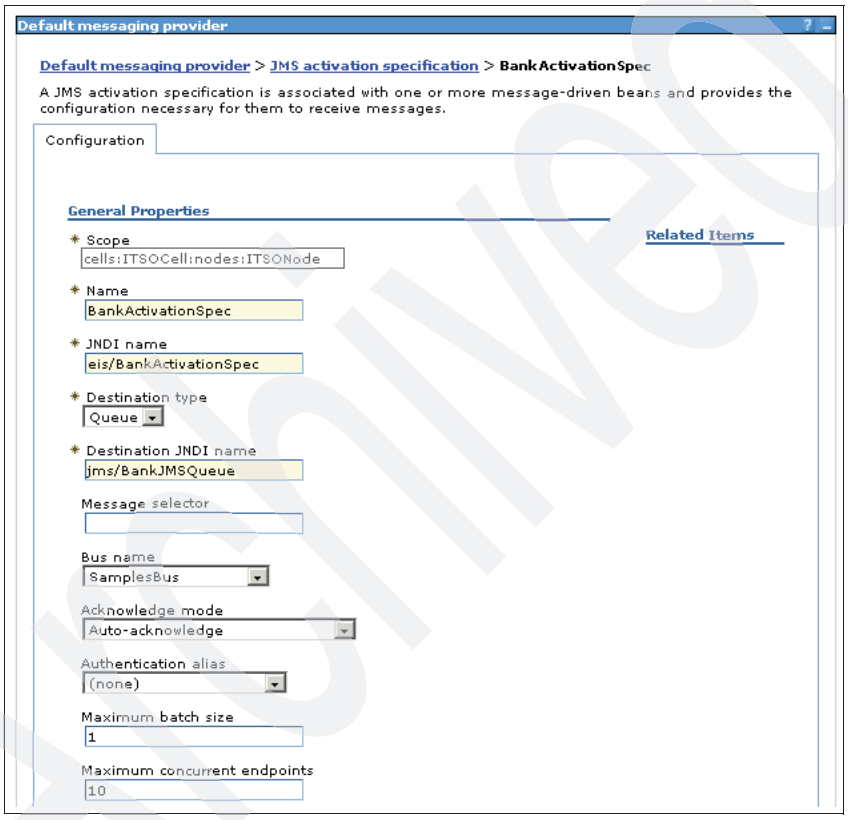


Figure 10-35 Default messaging JMS activation specification properties

6. Enter the required configuration properties for the JMS activation specification.
7. Click **OK**.
8. Save the changes and synchronize them with the nodes.
9. For the changes to become effective, restart any application servers within the scope of the resources.

Archived

10.6.3 Configuring the WebSphere MQ JMS provider

The WebSphere MQ JMS provider can be configured to communicate with WebSphere MQ using a bindings or client connection. These two connectivity options are described below:

- Bindings connection

  When used in bindings mode, the WebSphere MQ JMS provider uses the Java Native Interface (JNI) to call directly into the existing queue manager API, rather than communicating through a network. This provides better performance when connecting to WebSphere MQ than using a client connection.

  However, to use a bindings connection, WebSphere MQ and WebSphere Application Server V6 must be installed on the same machine.

- Client connection

  If it is not possible to collocate WebSphere Application Server V6 and WebSphere MQ on the same machine, the WebSphere MQ JMS provider must be configured to connect to WebSphere MQ using TCP/IP. Using a client connection allows you to perform authorization checks.

  Additional considerations must be taken into account when configuring the WebSphere MQ JMS provider to use a client connection, for instance:

  – Whether the connection needs to be secured by encrypting the data that flows over the connection

  – Whether the connection will go through a firewall

The sections that follow describe the properties exposed by WebSphere MQ connection factories and destinations, and also how to configure connection factories and destinations for the WebSphere MQ JMS provider.

Note: As discussed in section 10.5.2, “Managing the WebSphere MQ JMS provider” on page 519, WebSphere MQ resources such as queue managers, channels, and queues must be created using the tools provided with WebSphere MQ.

WebSphere MQ connection factory properties

A WebSphere MQ connection factory is used to create connections to WebSphere MQ. These connections form part of the common interfaces described in section 10.2.3, “JMS domains” on page 471 and can be used by a JMS client to interact with WebSphere MQ using both the Point-to-Point and Publish/Subscribe messaging models.

However, because the WebSphere MQ connection factory is not specific to either JMS domain, it encapsulates all of the configuration information that might be required to communicate using either messaging model. Consequently, a large number of properties are exposed by the WebSphere MQ connection factory object. Fortunately, default values are defined for many of these properties.

The sections that follow describe some of the more important properties that are exposed by the WebSphere MQ connection factory object. These properties have been grouped as follows:

- Bindings connection properties
- Client connection properties
- Queue connection specific properties
- Topic connection specific properties
- Connection security properties
- Advanced connection properties

Note: Not all of the properties of the WebSphere MQ connection factory are described. For a full description of all of the properties please refer to the *WebSphere Information Center* and the *WebSphere MQ Using Java* manual, links for which are contained in section 10.8, "References and resources" on page 590.

Bindings connection properties

With respect to the number of properties, setting up a bindings connection between a WebSphere MQ connection factory and WebSphere MQ is the simplest configuration. The properties required to configure a bindings connection for a WebSphere MQ connection factory object are shown in Table 10-28.

Table 10-28 WebSphere MQ connection factory bindings connection properties

Property	Description
Transport type	Use this property to specify whether WebSphere MQ client TCP/IP connection or interprocess bindings connection is to be used to connect to the WebSphere MQ queue manager. Inter-process bindings can only be used to connect to a queue manager on the same physical machine. Transport type defaults to `BINDINGS`.
Queue manager	This property is the name of the WebSphere MQ queue manager for this connection factory. Connections created by this factory connect to the specified queue manager on the local machine. If no queue manager is specified, the connections created by this factory will connect to the default queue manager on the local machine if one exists.

Client connection properties

The properties required to configure a basic client connection for a WebSphere MQ connection factory object are shown in Table 10-29.

Table 10-29 WebSphere MQ connection factory client connection properties

Property	Description
Transport type	Use this property to specify whether WebSphere MQ client TCP/IP connection or interprocess bindings connection is to be used to connect to the WebSphere MQ queue manager. To configure a WebSphere MQ client TCP/IP connection a value of CLIENT must be specified.
Host	This property is the name of the host on which the WebSphere MQ queue manager runs.
Port	This property defines the TCP/IP port number used for connection to the WebSphere MQ queue manager. This port number should match the listener port defined for the queue manager. The default value for the port property is 0 (zero). The default port for a WebSphere MQ queue manager listener is 1414.
Channel	Specify the name of the channel used for connection to the WebSphere MQ queue manager. If no channel is specified, the channel defaults to a standard server connection channel defined by all queue managers, called `SYSTEM.DEF.SVRCONN`.
Local server address	In some network configurations, firewalls are configured to prevent connection attempts unless they originate from specific ports or range of ports. The local server address property allows a port or range of ports to be specified for the WebSphere MQ connection factory to use when creating the outbound client connection. The local server address property defaults to null.

Queue connection properties

A number of the properties defined by the WebSphere MQ connection factory object are specific to WebSphere MQ queue destinations. Table 10-30 on page 555 describes these properties.

Archive

Table 10-30 WebSphere MQ connection factory queue connection specific properties

Property	Description
Enable message retention	Check this box to specify that unwanted messages are to be left on the queue. Otherwise, unwanted messages are dealt with according to their disposition options. By default, this means that a message is sent to the queue manager's dead-letter queue. It is also possible to specify that unwanted messages be discarded. The default value for the enable message retention property is true. The box is checked.
Model queue definition	This property is the name of the model queue from which WebSphere MQ dynamic queues are created. The model queue acts a template for the WebSphere MQ dynamic. WebSphere MQ dynamic queues are created as a result of the JMS client invoking the createTemporaryQueue method on the Session object. If no model queue definition is specified, it defaults to a standard model queue defined by all queue managers called SYSTEM.DEFAULT.MODEL.QUEUE.
Temporary queue prefix	The prefix that is used to form the name of a WebSphere MQ dynamic queue. The prefix must end in an asterisk (*) and be no more than 33 characters in length, including the asterisk. If no temporary queue prefix is specified, it defaults to AMQ.*.

Topic connection properties

A large number of the properties defined by the WebSphere MQ connection factory object are specific to WebSphere MQ topic destinations. Table 10-31 describes some of the more important topic connection specific properties.

Table 10-31 WebSphere MQ connection factory topic connection specific properties

Property	Description
Broker queue manager	Use this property to define the name of the WebSphere MQ queue manager that is hosting WebSphere Business Integration Event Broker or WebSphere Business Integration Message Broker. This can be different from the value specified for the queue manager property. However, if it is different, server channels must be defined between the two queue managers. If no broker queue manager is specified, it defaults to having the same value as the queue manager property.
Broker control queue	Define the name of the queue on the broker queue manager to which subscription requests should be sent. If no broker control queue is specified, it defaults to a standard control queue on the broker queue manager called SYSTEM.CONTROL.BROKER.QUEUE.

Property	Description
Broker publication queue	This property defines the name of the queue on the broker queue manager to which publications should be sent. If no broker publication queue definition is specified, it defaults to a standard publication queue on the broker queue manager called SYSTEM.BROKER.DEFAULT.STREAM.
Broker subscription queue	Specify the name of the queue on the broker queue manager from which non-durable subscription messages are retrieved. If no broker subscription queue is specified, it defaults to a SYSTEM.JMS.ND.SUBSCRIBER.QUEUE.
Client ID	Define the client identifier used when creating durable subscriptions to a topic. The client identifier is ignored for point-to-point connections.

Connection security properties

Security is an additional consideration when configuring a bindings connection between a WebSphere MQ connection factory and WebSphere MQ. Table 10-32 describes the properties of a WebSphere MQ connection factory that relate to security.

Table 10-32 WebSphere MQ connection factory connection security properties

Property	Description
Component-managed authentication alias	The component-managed authentication alias drop down can be used to specify a J2C authentication data entry. If the resource reference used within the JMS client application specifies a `res-auth` of Application, the user ID and password defined by the J2C authentication data entry will be used to authenticate the creation of a connection. component-managed authentication alias defaults to none. If no component-managed authentication alias is specified and the WebSphere MQ queue manager requires the user ID and password to get a connection, then an exception will be thrown when attempting to connect.
SSL cipher suite	Enter the SSL cipher suite used to encrypt the communication with the queue manager. If set, the value of this property must be a valid CipherSuite provided by the JSSE provider configured within WebSphere Application Server. It must also be equivalent to the CipherSpec specified on the server connection channel within WebSphere MQ, named by the CHANNEL property. By default, no value is specified for this property.

Archive

Property	Description
SSL CRL	The SSL CRL property specifies zero or more Certificate Revocation List (CRL) servers. These are LDAP servers that are used to check whether a SSL certificate has been revoked. If SSLCRL is not set, which is the default, no such checking is performed. Also, SSL CRL is ignored if no SSL cipher suite is specified.
SSL peer name	The SSL peer name property specifies a distinguished name which must match the SSLPEER parameter specified on the server connection channel named by the CHANNEL property. If the SSL peer name property is not set, which is the default, no such checking is performed. Also, SSL peer name is ignored if no SSL cipher suite is specified.

Advanced connection properties

The WebSphere MQ connection factory object also exposes a number of properties that affect how the WebSphere MQ JMS provider interacts with WebSphere MQ. In order to fully understand these properties, an advanced knowledge of WebSphere MQ is required. Some of the more important properties are described in Table 10-33.

Table 10-33 WebSphere MQ connection factory advanced properties

Property	Description
CCSID	Use this property to define the coded-character-set-ID to be used on connections. The value for this property defaults to null. This indicates to the WebSphere MQ JMS provider that its default CCSID should not be overridden. The default CCSID within the WebSphere MQ JMS provider is 819, which represents the ASCII character set. Changing this value affects the way in which the queue manager that this connection factory creates connections for translates information in the WebSphere MQ headers.
XA enabled	Specify whether the resources of WebSphere MQ can be enlisted into a distributed transaction. The default value for the XA enabled property is true. The box is checked. If the XA enabled check box is not selected, 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, unless last participant support is used, this means that only a single resource can be enlisted in a transaction in WebSphere Application Server.

Property	Description
Enable return methods during shutdown	Define whether a JMS client application returns from a method call if the queue manager has entered a controlled shutdown. The default value for the enable return methods during shutdown property is true (the check box is selected).
Polling interval	The polling interval property specifies the interval, in milliseconds, between scans of all receivers during asynchronous message delivery. The polling interval property defaults to 5000.
Rescan interval	The rescan interval property specifies the interval in milliseconds between which a queue is scanned to look for messages that have been added to a queue out of order. This interval controls the scanning for messages that have been added to a queue out of order with respect to a WebSphere WebSphere MQ browse cursor. The rescan interval property defaults to 5000.
Enable MQ connection pooling	This property specifies whether MQ connection pooling should be used to pool the connections to the WebSphere MQ queue manager. If MQ connection pooling is used, when a connection is no longer required, instead of destroying it, it can be pooled, and later reused. This can provide a substantial performance enhancement for repeated connections to the same queue manager. The default value for the enable MQ connection pooling property is true. The box is checked.

WebSphere MQ connection factory configuration

To configure a connection factory for the WebSphere MQ JMS provider, complete the following steps:

1. In the navigation tree, expand **Resources** → **JMS Providers**.
2. Click **WebSphere MQ**.
3. Set the **Scope** for the connection factory.
4. Click **WebSphere MQ connection factories** in the **Additional Properties** section. A list of any existing connection factories defined at this scope will be displayed. This is shown in Figure 10-36.

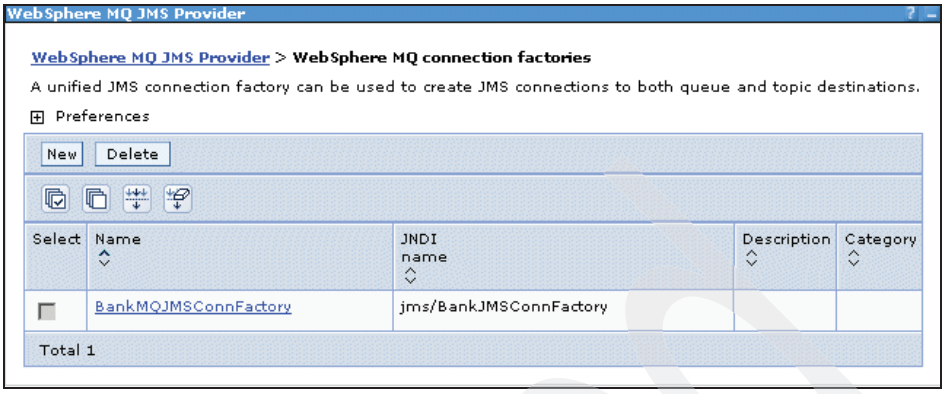


Figure 10-36 WebSphere MQ connection factory administered objects

In this example, we already have one WebSphere MQ connection factory object defined, called BankMQJMSConnFactory. This connection factory object has all of the necessary properties configured in order to connect to a full WebSphere MQ JMS provider using a client connection.

5. To create a new connection factory object, click **New**. Alternatively, to change the properties of an existing connection factory, click one of the connection factories displayed. Figure 10-37 on page 560 shows the top portion of the configuration page for BankMQJMSConnFactory object.

   Other than the standard JMS administered object properties, **Name** and **JNDI name**, the only other properties that we must specify values for, in order to configure a client connection to WebSphere MQ, are as follows:

   – **Transport type**

     A transport type of CLIENT has been specified to indicate that we will connect to WebSphere MQ using a WebSphere MQ client TCP/IP connection.

   – **Host**

     The WebSphere MQ queue manager is running on host kaa5070.

   – **Port**

     The WebSphere MQ queue manager listener is listening on port 1414.

   The BankMQJMSConnFactory object uses the default value for the Channel property, which is SYSTEM.DEF.SVRCONN.

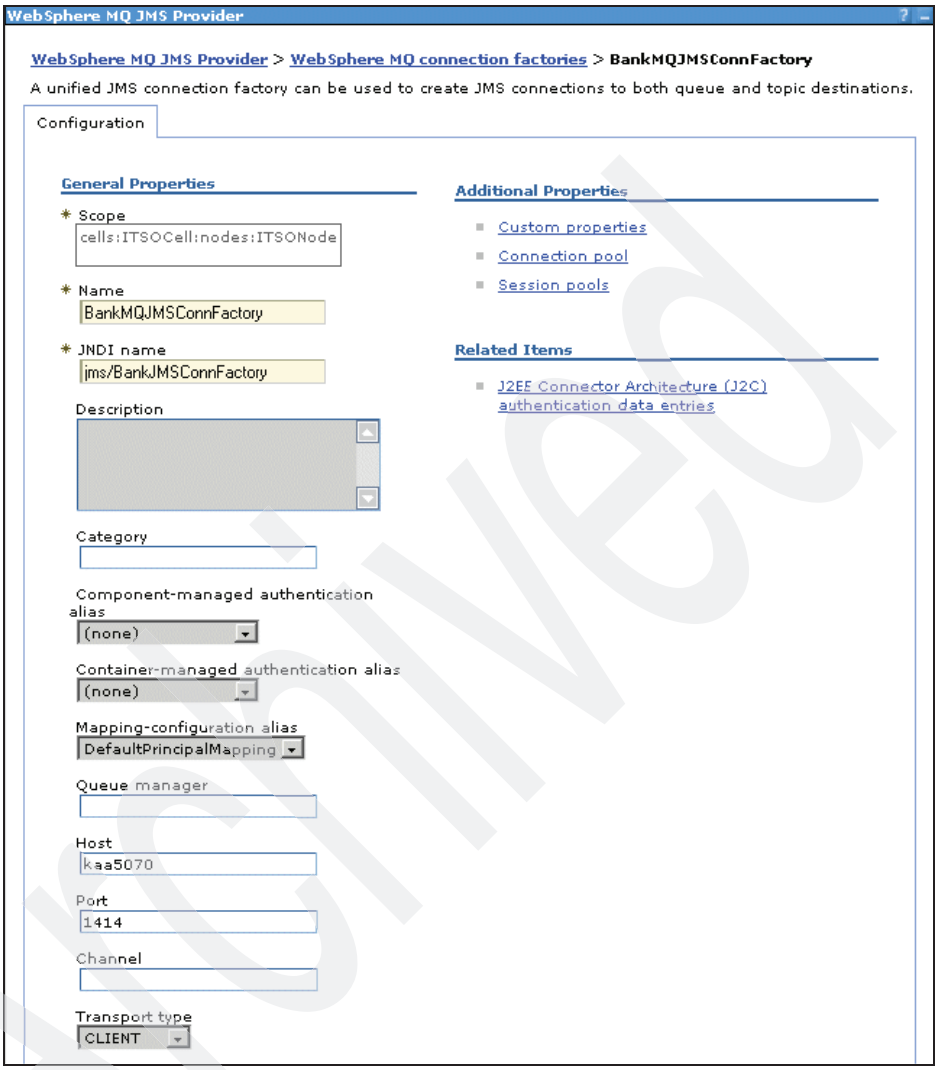


Figure 10-37 WebSphere MQ connection factory properties

6. Enter the required configuration properties for the WebSphere MQ connection factory.
7. Click **OK**.
8. Save the changes and synchronize them with the nodes.
9. For the changes to become effective, restart any application servers within the scope of the resources.

Archived

WebSphere MQ destination properties

Both queue and topic destinations can be configured for the WebSphere MQ JMS provider. The sections that follow describe the properties of the queue and topic destinations. These properties have been grouped as follows:

- Basic destination connection properties
- Queue specific destination properties
- Topic specific destination properties
- Advanced destination properties

Basic destination properties

The WebSphere MQ queue and WebSphere MQ topic destinations share a number of basic common properties. These properties are described in Table 10-34.

Table 10-34 Basic WebSphere MQ destination properties

Property	Description
Persistence	Use this property to specify whether the messages sent to this destination are persistent, non-persistent, or have their persistence defined by the application or queue. The default value for the persistence property is APPLICATION DEFINED. This specifies that the messages on the destination have their persistence defined by the application that put them onto the queue.
Priority	Use this property to specify whether the message priority for this destination is defined by the application, queue or the Specified priority property. The default value for the priority property is APPLICATION DEFINED. This specifies that the priority of messages on this destination is defined by the application that put them onto the destination.
Specified priority	If the Priority property is set to **Specified**, the value of this property determines the message priority for messages sent to this destination. Priorities range from 0 (lowest) through 9 (highest).
Expiry	Specify whether the expiry timeout for this destination is defined by the application or the Specified Expiry property, or messages on the destination never expire (have an unlimited expiry timeout). The default value for the expiry property is APPLICATION DEFINED. This specifies that the expiry timeout of messages on this destination is defined by the application that put them onto the destination.

Archived

Property	Description
Specified expiry	If the Expiry Timeout property is set to **Specified**, the value of this property determines the number of milliseconds (greater than 0) after which messages on this destination expire.

Queue specific destination properties

The properties specific to WebSphere MQ queue destination objects are shown in Table 10-35.

Table 10-35 WebSphere MQ queue destination properties

Property	Description
Base queue name	Use this property to specify the name of the queue to which messages are sent, on the queue manager specified by the Base Queue Manager Name property.
Base queue manager name	Specify 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. The default value for this property is null, in which case the queue manager is assumed to be that of the connection factory object used to connect to WebSphere MQ.

Topic specific destination properties

The properties specific to WebSphere MQ topic destination objects are shown in Table 10-36.

Table 10-36 WebSphere MQ topic destination properties

Property	Description
Base topic name	Use this property to specify the name of the topic on the underlying queue manager that JMS clients will publish or subscribe to.
Broker durable subscription queue	Define the name of the brokers queue from which durable subscription messages are retrieved. The subscriber specifies the name of the queue when it registers a subscription.
Broker CC durable subscription queue	Specify the name of the brokers queue from which durable subscription messages are retrieved for a ConnectionConsumer.
Enable multicast transport	Indicate whether or not this topic destination uses multicast transport if supported by the connection factory.

Archived

Advanced destination properties

The WebSphere MQ queue and WebSphere MQ topic destinations share a number of advanced common properties. These properties are described in Table 10-37.

Table 10-37 Advanced WebSphere MQ destination properties

Property	Description
CCSID	Use this property to identify 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.
Use native encoding	Indicate whether or the destination should use native encoding, appropriate encoding values for the Java platform.
Integer encoding	If native encoding is not enabled, select whether integer encoding is normal or reversed.
Decimal encoding	If native encoding is not enabled, select whether decimal encoding is normal or reversed.
Floating point encoding	If native encoding is not enabled, select the type of floating point encoding.
Target client	Indicate whether the receiving application is JMS-compliant or is a traditional WebSphere MQ application.

WebSphere MQ queue destination configuration

To configure a queue destination for the WebSphere MQ JMS provider, complete the following steps:

1. In the navigation tree, expand **Resources** → **JMS Providers**.
2. Click **WebSphere MQ**.
3. Set the **Scope** for the queue destination.
4. Click **WebSphere MQ queue destinations** in the **Additional Properties** section. A list of any existing queue destinations defined at this scope will be displayed. This is shown in Figure 10-38 on page 564.

Archive

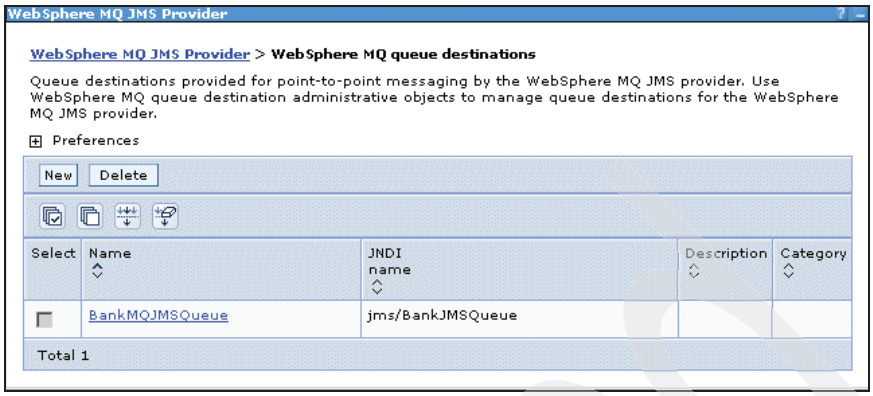


Figure 10-38 WebSphere MQ queue destination administered objects

In this example, we already have one WebSphere MQ queue destination object defined, called BankMQJMSQueue.

5. To create a new queue destination object, click **New**. Alternatively, to change the properties of an existing queue destination, click one of the queue destinations displayed. Figure 10-39 on page 565 shows the top portion of the configuration page for BankMQJMSQueue object.

   Other than the standard JMS administered object properties, **Name** and **JNDI name**, the only property that we must specify a value for is **Base queue name**. In Figure 10-39, the value specified for the Base queue name property is BankJSQueue. This must match the name of the queue defined on the WebSphere MQ queue manager to which we are connecting.

Archive

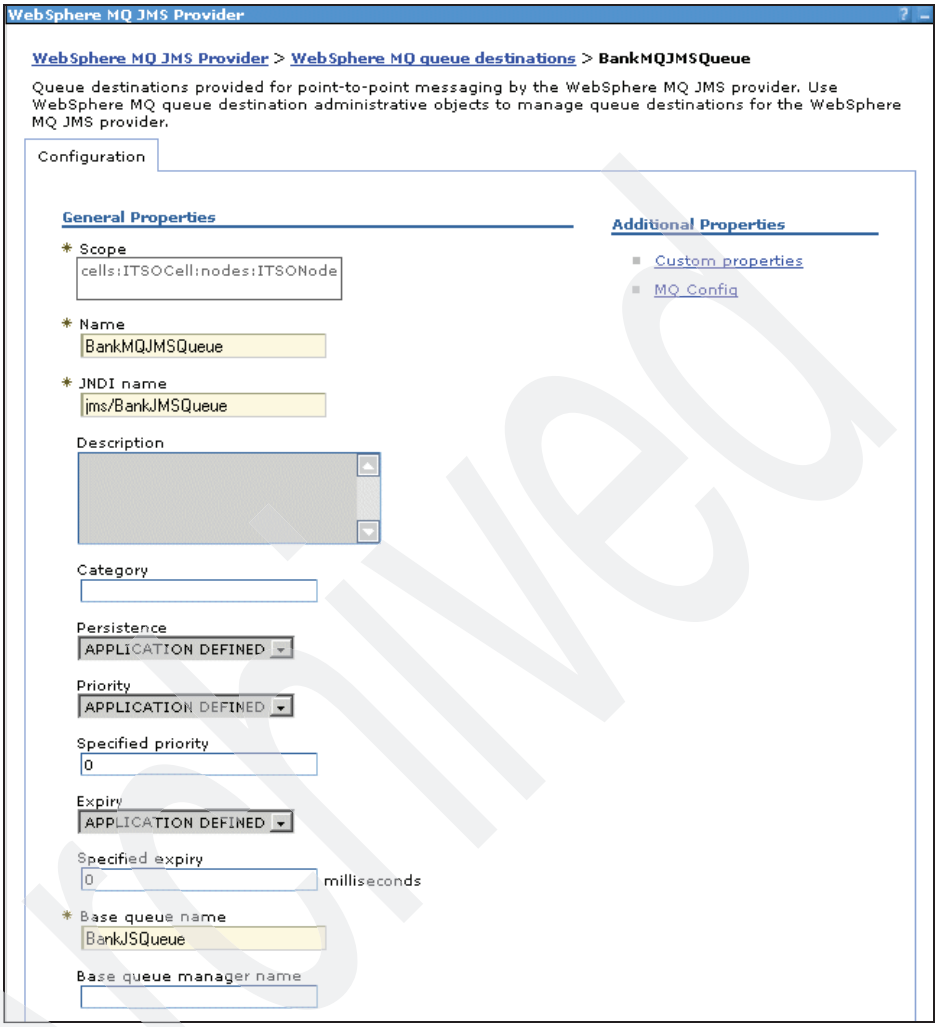


Figure 10-39 WebSphere MQ queue destination properties

6. Enter the required configuration properties for the WebSphere MQ queue destination.
7. Click **OK**.
8. Save the changes and synchronize them with the nodes.
9. For the changes to become effective, restart any application servers within the scope of the resources.

Archived

WebSphere MQ topic destination configuration

To configure a topic destination for the WebSphere MQ JMS provider, complete the following steps:

1. In the navigation tree, expand **Resources** → **JMS Providers**.
2. Click **WebSphere MQ**.
3. Set the **Scope** for the queue destination.
4. Click **WebSphere MQ topic destinations** in the **Additional Properties** section. A list of any existing topic destinations defined at this scope will be displayed. This is shown in Figure 10-40.

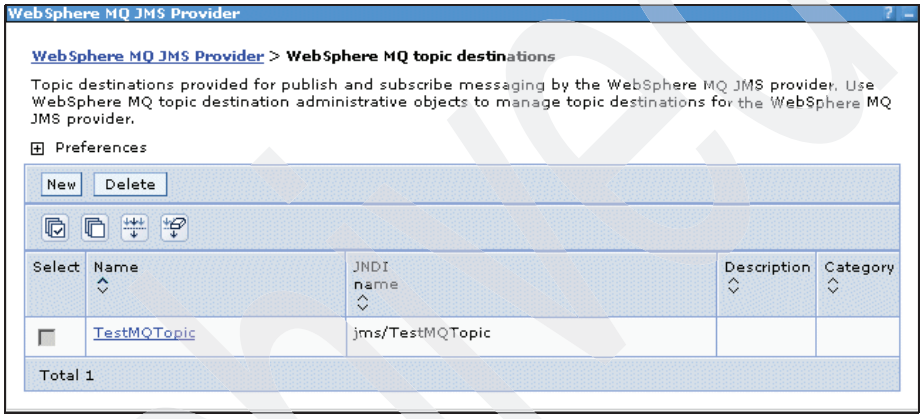


Figure 10-40 WebSphere MQ topic destination administered objects

In this example, we already have one WebSphere MQ topic destination object defined, called TestMQTopic.

5. To create a new topic destination object, click **New**. Alternatively, to change the properties of an existing topic destination, click one of the topic destinations displayed. Figure 10-41 on page 567 shows the top portion of the configuration page for TestMQTopic object.

   Other than the standard JMS administered object properties, **Name** and **JNDI name**, the only property that we must specify a value for is **Base topic name**. In Figure 10-41, the value specified for the Base topic name property is `TestTopic`. This must match the name of the topic defined on the broker.

Archived

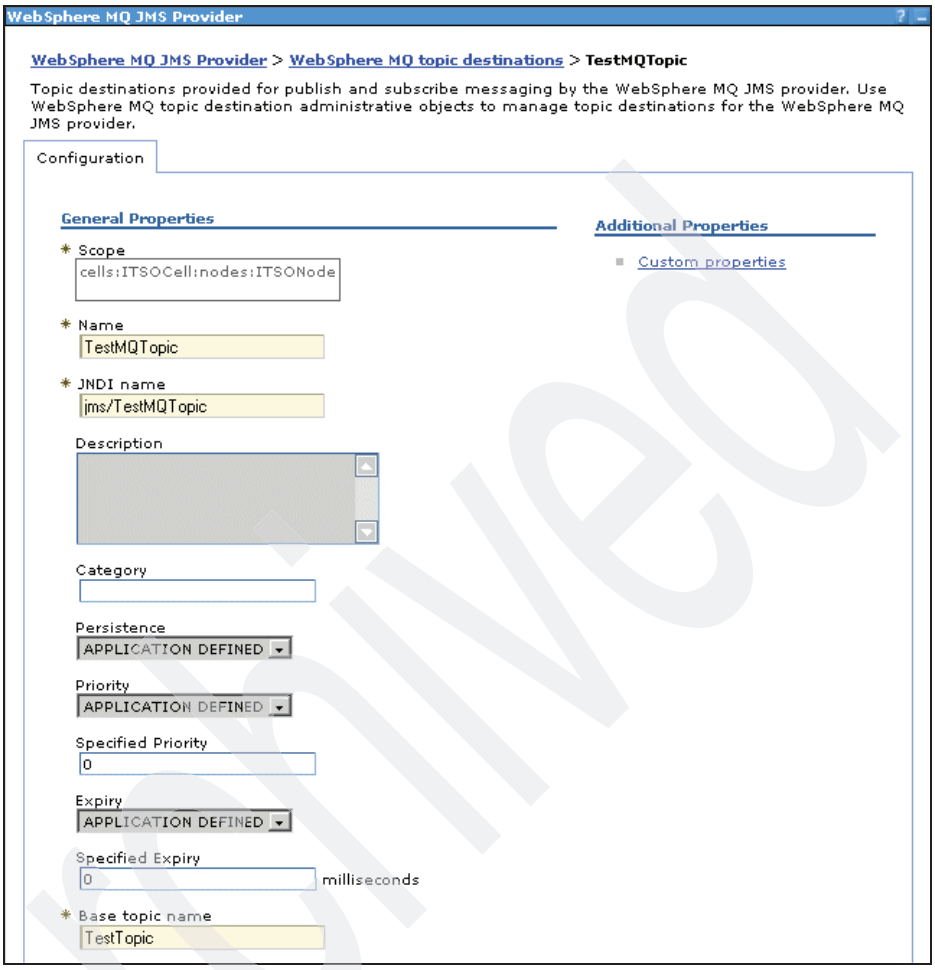


Figure 10-41 WebSphere MQ topic destination properties

6. Enter the required configuration properties for the WebSphere MQ topic destination.
7. Click **OK**.
8. Save the changes and synchronize them with the nodes.
9. For the changes to become effective, restart any application servers within the scope of the resources.

Archived

10.6.4 Configuring listener ports

As discussed in section 10.4.7, “Associating a message-driven bean with a destination” on page 509, a listener port is used to associate a message-driven bean with a connection factory and a destination for the WebSphere MQ JMS provider. A listener must be defined on the application server on which the message-driven application will be installed. To configure a listener port, complete the following steps:

1. In the navigation tree, expand **Servers**.
2. Click **Application servers**.
3. A list of the application servers defined within the cell will be displayed. This is shown in Figure 10-42.

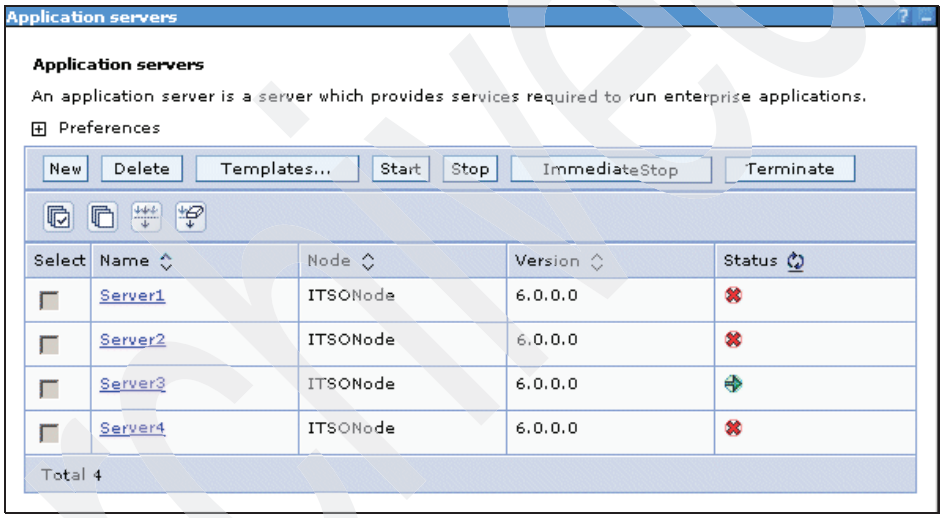


Figure 10-42 Application servers defined within the cell

4. Click the application server on which to create the listener port.
5. The configuration properties for the application server will be displayed. In the **Communications** section, expand **Messaging**.
6. Click **Message Listener Service**, as shown in Figure 10-43 on page 569.

Archived

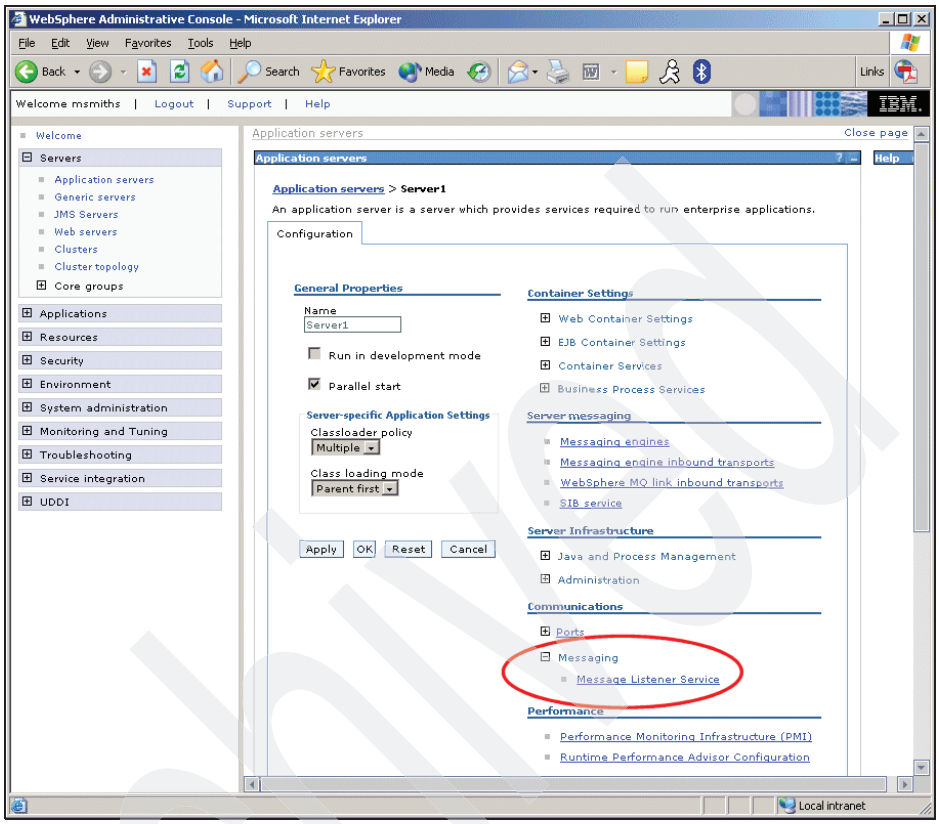


Figure 10-43 Message listener service link

7. The configuration properties for the message listener service will be displayed.
8. Click **Listener Ports**. A list of listener ports that are currently defined for this application server will be displayed. This is shown in Figure 10-44 on page 570.

Archived

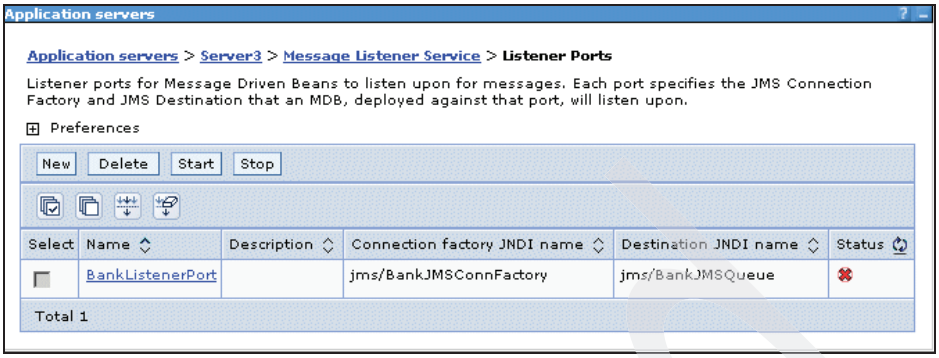


Figure 10-44 Listener ports

In this example, we already have one listener port defined, called `BankListenerPort`.

9. To create a new listener port, click **New**. Alternatively, to change the properties of an existing listener port, click one of the listener ports displayed. Figure 10-45 on page 571 shows the configuration page for the BankListenerPort object. Values must be specified for the **Name**, **Initial State**, **Connection factory JNDI name** and **Destination JNDI name** properties.

Archive

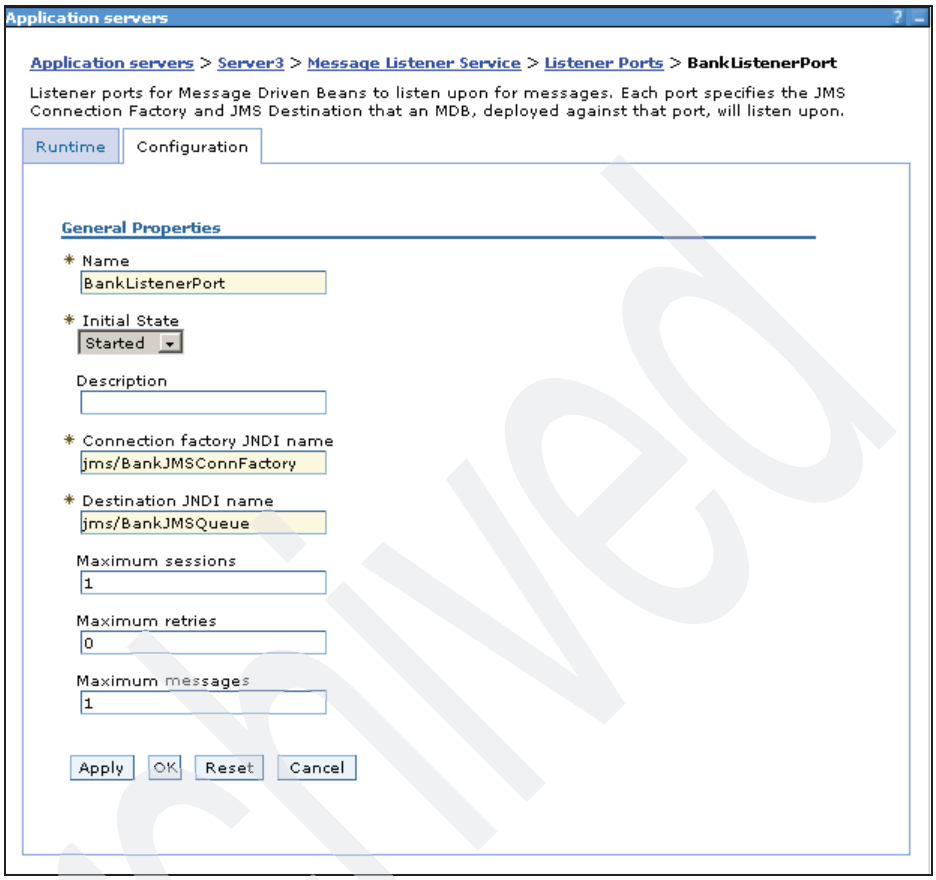


Figure 10-45 Listener port properties

10. Enter the required configuration properties for the JMS activation specification.
11. Click **OK**.
12. Save the changes and synchronize them with the nodes.
13. For the changes to become effective, restart any application servers within the scope of the resources.

Following our example through, using the WebSphere MQ connection factory defined in “WebSphere MQ connection factory configuration” on page 558 and the WebSphere MQ queue defined in “WebSphere MQ queue destination configuration” on page 563, we know that the BankMQJMSConnFactory object was bound into the JNDI name space with the name jms/BankJMSConnFactory. This JMS connection factory maps to a WebSphere MQ Queue Manager running

Archived

on host kaa5070 and listening on port 1414. We also know that the BankMQJMSQueue object was bound into the JNDI name space with the name jms/BankJMSQueue. This JMS queue maps on to the BankJSQueue on this WebSphere MQ Queue Manager.

Therefore, if a message-driven bean is associated with this listener port, it would be invoked when messages arrived at the BankJSQueue destination on the WebSphere MQ Queue Manager listening on port 1414 of host kaa5070.

10.6.5 Configuring the generic JMS provider

If you use a generic JMS provider, the WebSphere administrative console can still be used to configure JMS administered objects within the JNDI name space of the application server. The sections that follow describe how the WebSphere administrative console can be used to configure JMS connection factories and JMS destinations for a generic JMS provider.

JMS connection factory configuration

To configure a JMS connection factory for a generic JMS provider, complete the following steps:

1. In the navigation tree, expand **Resources** → **JMS Providers**.
2. Click **Generic**.
3. Set the scope to the level at which the generic JMS provider has been configured. The generic JMS provider should appear in the list providers defined at this level.
4. Select the generic JMS provider for which you want to configure the JMS connection factory.
5. Click **JMS connection factories** in the **Additional Properties** section to display a list of the connection factories that have been defined for the generic JMS provider.
6. To create a new connection factory object, click **New**. Alternatively, to change the properties of an existing connection factory, click one of the connection factories displayed. Figure 10-46 on page 573 shows the configuration page for a connection factory object.

Archived

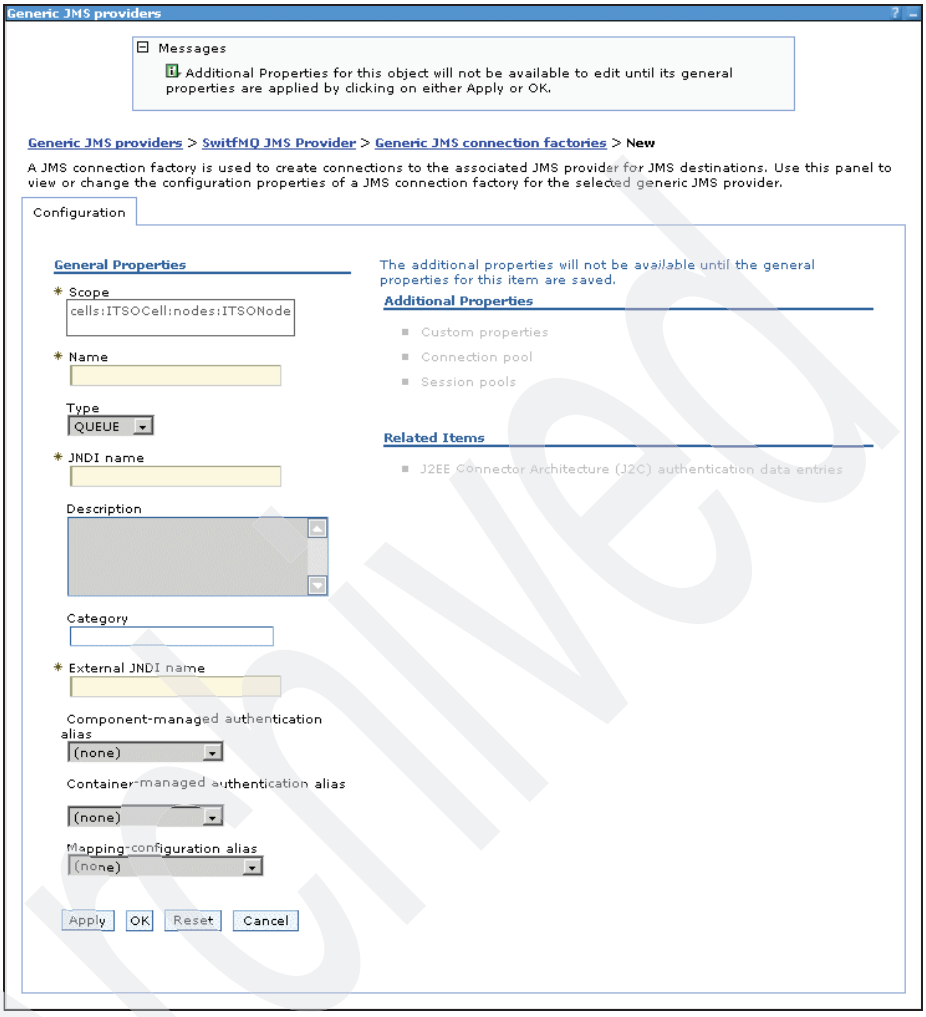


Figure 10-46 Generic JMS provider connection factory configuration panel

7. Enter the required configuration properties for the JMS connection factory. The common properties are described in 10.6.1, “Common administration properties” on page 526. The properties specific to the generic JMS connection factory object are shown in Table 10-38 on page 573.

Table 10-38 Generic JMS provider connection factory properties

Property	Description
Type	Use this property to specify whether the connection factory is for JMS queues (QUEUE) or JMS topics (TOPIC).

Archived

Property	Description
External JNDI name	Specifiy the JNDI name used to bind the JMS connection factory into the name space of the messaging provider.
Component managed authentication alias	The component-managed authentication alias list can be used to specify a Java 2 Connector authentication data entry. If the resource reference used within the JMS client application specifies a `res-auth` of Application, the user ID and password defined by the Java 2 Connector authentication data entry will be used to authenticate the creation of a connection. The component-managed authentication alias defaults to none. If no component-managed authentication alias is specified and the messaging provider requires the user ID and password to get a connection, then an exception will be thrown when attempting to connect.

8. Click **OK**.
9. Save the changes and synchronize them with the nodes.
10. For the new connection factory to be bound into the JNDI name space at the correct scope, restart the relevant application servers.

JMS destination configuration

To configure a JMS destination for a generic JMS provider, complete the following steps:

1. In the navigation tree, expand **Resources** → **JMS Providers**.
2. Click **Generic**.
3. Set the scope to the level at which the generic JMS provider has been configured. The generic JMS provider should appear in the list providers defined at this level.
4. Select the generic JMS provider for which you want to configure the JMS destination.
5. Click **JMS destinations** in the **Additional Properties** section to display a list of the destinations that have been defined for the generic JMS provider.
6. To create a new destination, click **New**. Alternatively, to change the properties of an existing destination, click one of the destinations displayed.
   Figure 10-47 on page 575 shows the configuration page for a destination object.

Archive

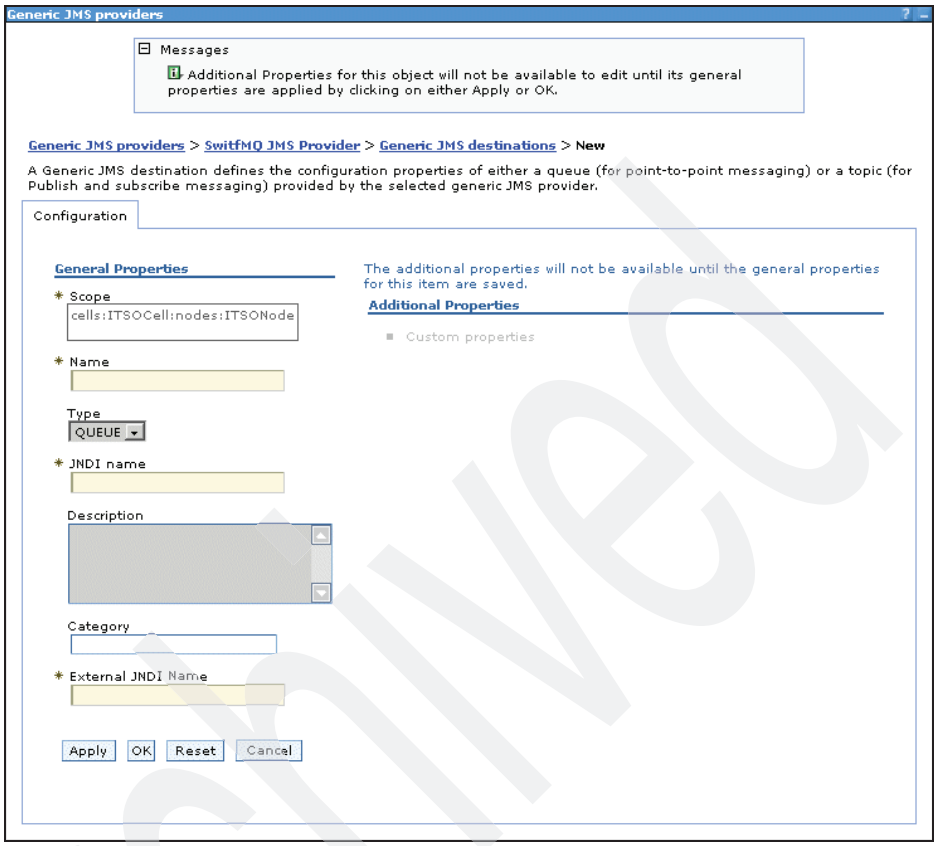


Figure 10-47 Generic JMS provider destination configuration panel

7. Enter the required configuration properties for the JMS destination. The common properties are described in 10.6.1, “Common administration properties” on page 526. The properties specific to the generic JMS destination object are shown in Table 10-39.

Table 10-39 Generic JMS provider destination properties

Property	Description
Type	Use this property to specify whether the connection factory is for JMS queues (QUEUE) or JMS topics (TOPIC).
External JNDI name	Define the JNDI name used to bind the JMS connection factory into the name space of the messaging provider.

8. Click **OK**.

Archived

9. Save the changes and synchronize them with the nodes.
10. For the new destination to be bound into the JNDI name space at the correct scope, restart the relevant application servers.

10.7 Connecting to a service integration bus

A JMS client obtains connections to a service integration bus using a suitably configured JMS connection factory, defined for the default messaging JMS provider. However, the selection of which messaging engine within a particular service integration bus a JMS client will connect to, depends on the connection properties defined within the JMS connection factory. The options available can range from simply connecting to any suitable messaging engine within the named service integration bus, to using a highly specific connection selection algorithm. The sections that follow describe the mechanisms used to determine the most suitable messaging engine when a JMS client is connecting to a service integration bus.

Note: None of the messaging engine selection processes discussed in this section effect the JMS client in any way. As far as the JMS client is concerned, the ConnectionFactory simply returns a connection to the underlying messaging provider, in this case a service integration bus. The process of configuring a ConnectionFactory in order to tailor the messaging engine that is selected, is a purely administrative task.

10.7.1 JMS client runtime environment

Regardless of the environment in which a JMS client is executing, it will always perform the same steps in order to connect to a JMS provider. These steps are:

1. Obtain a reference to a JMS connection factory from the JNDI name space.
2. Invoke the createConnection method on the JMS connection factory.

The important point here is that the JMS connection factory object will always execute within the same process as the JMS client. However, the JMS client, and therefore the JMS connection factory, might be executing inside of a WebSphere Application Server V6 process, or they might be executing within a stand-alone JVM. In the case of the connection factory for the default messaging JMS provider, the behavior of the connection factory depends on the environment in which it is executing.

- Clients running inside of WebSphere Application Server V6

  When the connection factory is executing within the WebSphere Application Server V6 environment, it is able to communicate with components of the

WebSphere runtime in order to determine which messaging engines are defined within the specified service integration bus, and where these messaging engines are currently located. The relevant connection properties configured on the connection factory can then be used to select a suitable messaging engine to which to connect.

Note: The connection factory is only able to determine the location of messaging engines that are defined within the same WebSphere cell. If the target bus is defined within another cell then a list of suitable provider endpoints must be configured on the connection factory.

- Clients running outside of WebSphere Application Server V6

  When the connection factory is executing outside of the WebSphere Application Server V6 environment, or in a WebSphere Application Server V6 environment in a different cell to the target bus, it is not able to determine which messaging engines are defined within the specified service integration bus or where they are currently located. In order to obtain this information, the connection factory must connect to an application server within the same cell as the target bus. This application server is known as a *bootstrap server*.

  A bootstrap server is simply an ordinary application server that is running the SIB service. The SIB service is the component within an application server that manages the service integration bus resources for that application server. It is the SIB service that enables an application server to act as bootstrap server for default messaging JMS provider connection factories. However, while the bootstrap server needs to be running the SIB service, it does not necessarily need to be hosting any messaging engines. This is shown in Figure 10-48 on page 578.

Archive

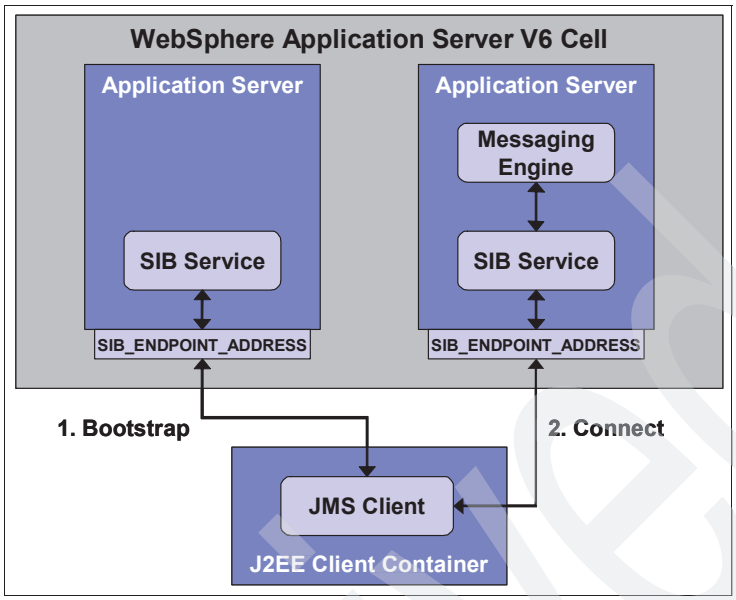


Figure 10-48 Using a bootstrap server with a messaging engine

Use the **provider endpoints** property to configure the bootstrap servers to which a connection factory can connect. See Figure 10-28 on page 535.

Provider endpoints

The provider endpoints property of the connection factory allows an administrator to specify a comma-separated list of suitable bootstrap servers for the connection factory. Each bootstrap server in the list is specified as a triplet of the form:

```
hostname : port : transport chain
```

The different elements are:

- `hostname` is the name of the host on which the bootstrap server is running. If a hostname is not specified the value will default to localhost.
- `port` is the port number that the SIB service for the bootstrap server is listening on. This can be determined from the relevant messaging engine inbound transport that will be used for the bootstrap request. If no port is specified the value will default to 7276 (the default port number for SIB_ENDPOINT_ADDRESS).
- `transport chain` specifies the transport chain that will be used to send the bootstrap request to the bootstrap server. Valid values for transport chain are:

– **BootstrapBasicMessaging**

  The bootstrap request will be sent to the bootstrap server using a standard TCP/IP connection to the InboundBasicMessaging transport chain.

– **BootstrapSecureMessaging**

  The bootstrap request will be sent to the bootstrap server over a secure TCP/IP connection to the InboundSecureMessaging transport chain.

– **BootstrapTunneledMessaging**

  The bootstrap request will be tunneled to the bootstrap server over an HTTP connection. Before you can use this transport chain, you must define a corresponding transport chain on the bootstrap server.

– **BootstrapTunneledSecureMessaging**

  The bootstrap request will be tunneled to the bootstrap server over a secure HTTP connection. Before you can use this transport chain, you must define a corresponding transport chain on the bootstrap server.

If no transport chain is specified the value will default to BootstrapBasicMessaging.

If no value is specified for the provider endpoint property, the connection factory will use the following default provider endpoint address:

```
localhost:7276:BootstrapBasicMessaging
```

Dedicated bootstrap servers

Because the location of a bootstrap server is defined explicitly within the provider endpoints property of a connection factory, consideration must be given to the availability of the bootstrap server. By specifying a list of bootstrap servers in the provider endpoints property, a connection factory is able to transparently bootstrap to another server in the list in the event that one of the bootstrap servers fails. The connection factory attempts to connect to a bootstrap server in the order in which they are specified in the provider endpoints list. However, you want to avoid specifying a long list of bootstrap servers. Consider configuring only a few highly available application servers as dedicated bootstrap servers.

10.7.2 Controlling messaging engine selection

The remaining connection properties that can be specified on a connection factory for the default messaging JMS provider are used to control how the connection factory selects the messaging engine to connect to on the specified service integration bus. The sections that follow discuss these properties in more detail.

Bus name

The only connection property that is required when configuring a connection factory for the default messaging JMS provider is the bus name property. The value of the bus name property specifies the name of the bus to which the connection factory will create JMS connections.

In the absence of any other connection properties, the connection factory returns a connection to any available messaging engine in the bus. However, despite the freedom to connect to any available messaging engine in the bus, the connection factory applies a few simple rules to find the most suitable messaging engine with which to connect. The rules are as follows:

1. The connection factory looks for a messaging engine within the specified service integration bus that is in the same server process as the JMS client. If a messaging engine within the specified bus is found in the same application server process, then a direct *in-process* connection is made from the JMS client to the messaging engine. This is shown in Figure 10-49.

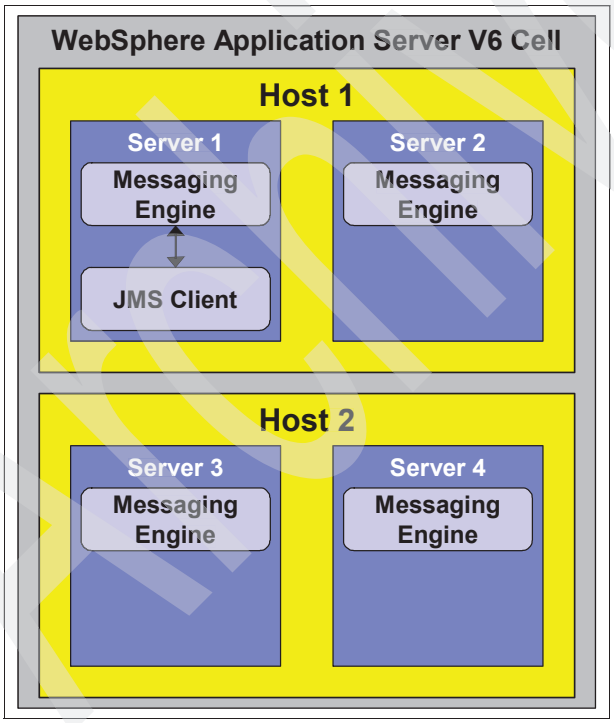


Figure 10-49 In-process connection for a JMS client and a messaging engine

Note: A direct in-process connection provides the best performance when connecting a JMS client to a messaging engine.

2. If it is not possible for the connection factory to create a connection to a messaging engine in the same application server process, the connection factory looks for a messaging engine that is running on the same host as the JMS client. If a messaging engine within the specified bus is found on the same host, then a remote connection is made from the JMS client to the messaging engine. This is shown in Figure 10-50.

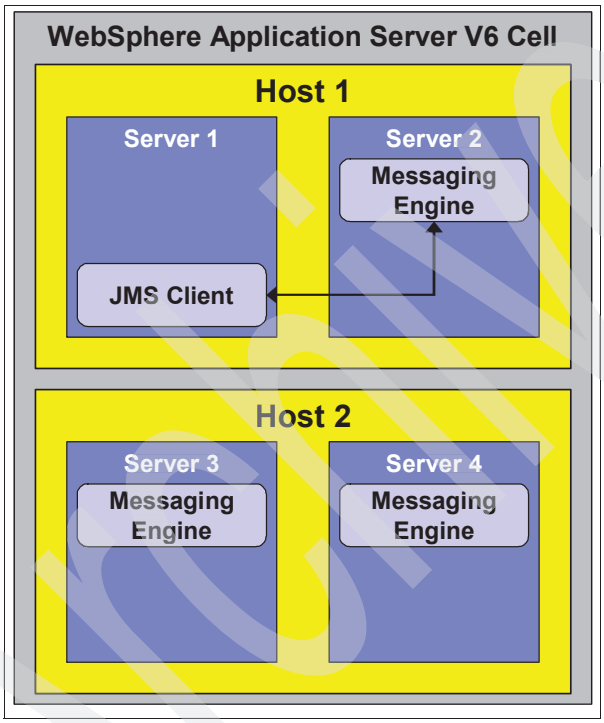


Figure 10-50 Remote connection on the same host

Note: If multiple messaging engines are available on the same host as the JMS client, new connections to the target bus will be load-balanced across them.

3. If it is not possible for the connection factory to create a connection to a messaging engine on the same host as the JMS client, the connection factory

looks for any other messaging engine that is part of the specified service integration bus. This is shown in Figure 10-51.

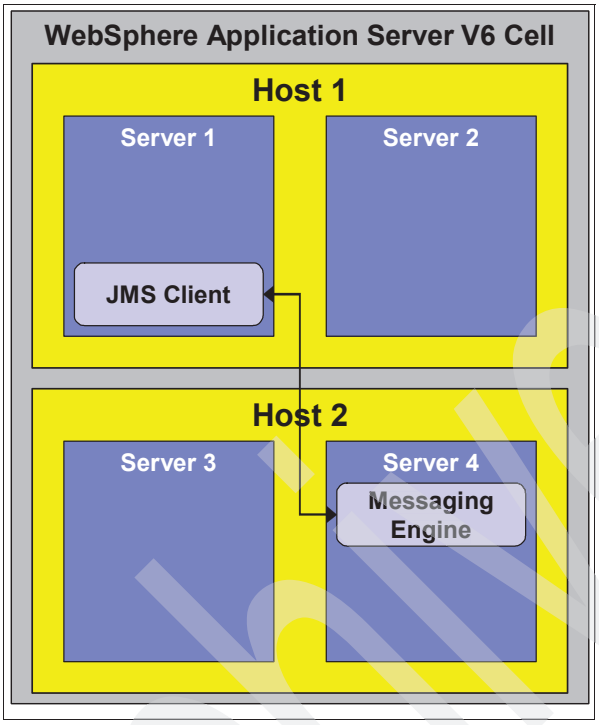


Figure 10-51 Remote connection on a different host

Note: If multiple messaging engines are available within the target bus, new connections to the target bus will be load balanced across them.

4. If it is not possible for the connection factory to create a connection to any of the messaging engines that make up the specified service integration bus, the connection factory throws a javax.jms.JMSException to the JMS client. The javax.jms.JMSException contains a linked exception to a service integration bus specific exception, similar to that shown in Example 10-21.

Example 10-21 Failure to connect to a messaging engine

```
com.ibm.websphere.sib.exception.SIResourceException: CWSITO019E: No suitable
messaging engine is available in bus SamplesBus.
```

Target inbound transport chain

The target inbound transport chain property for a connection factory specifies the transport chain that the JMS client should use when establishing a remote connection to a messaging engine. Suitable values for this property are:

- **InboundBasicMessaging**

  The JMS client establishes a standard TCP/IP connection to the messaging engine. This is the default value for the target inbound transport chain property.

- **InboundSecureMessaging**

  The JMS client establishes a secure TCP/IP connection to the messaging engine.

The process of selecting a suitable messaging engine takes into account the inbound transport chains that are currently available to those messaging engines under consideration. There is no point in selecting a messaging engine which cannot be contacted using the target transport chain specified, so a final selection is made only from those messaging engines which have the specified target transport chain available to them.

Connection proximity

The messaging engine selection process performed by the connection factory can be subtly altered by specifying different connection proximities. The connection proximity property is used to restrict the set of available messaging engines considered for selection by the connection factory. The set of available messaging engines is restricted based on their proximity to the JMS client or the bootstrap server acting on behalf of the JMS client. The valid values for the connection proximity property are as follows:

- **Bus**

  The set of available messaging engines will include all messaging engines defined within the target service integration bus. This is the default value for the connection proximity property and, in effect, does not restrict the set of available messaging engines in any way. When a connection proximity of Bus is specified, the messaging engine selection process described in “Bus name” on page 580, is used.

- **Cluster**

  The set of available messaging engines for the target service integration bus I only includes those messaging engines defined within the same cluster as the JMS client or bootstrap server.

Archived

- **Host**

  The set of available messaging engines for the target service integration bus only includes those messaging engines running on the same host as the JMS client or bootstrap server.

- **Server**

  The set of available messaging engines for the target service integration bus only includes those messaging engines running on the within the same application server process as the JMS client or bootstrap server.

To see how the value of the connection proximity property effects the messaging engine selection process, consider the configuration shown in Figure 10-52. All of the messaging engines shown in Figure 10-52 exist within the same service integration bus.

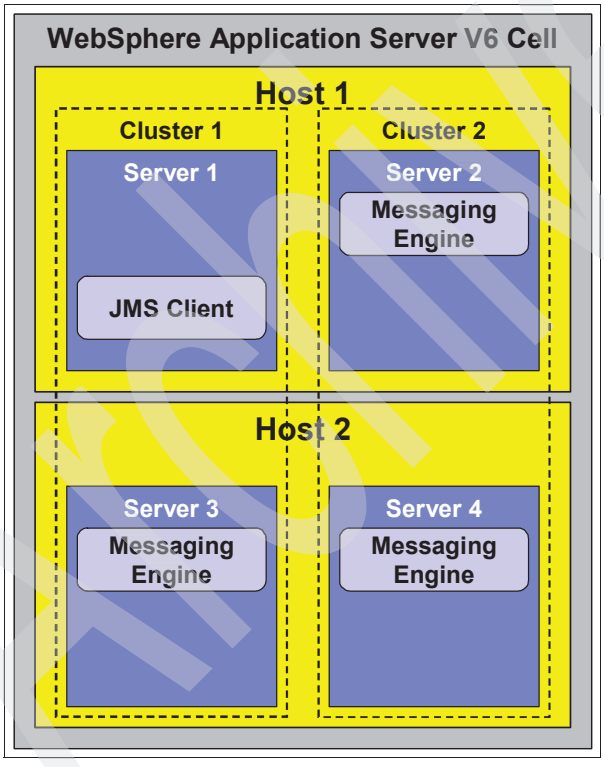


Figure 10-52 Sample topology for a service integration bus

The effect of the value of the connection proximity property on messaging engine selection is described in Table 10-40 on page 585.

Table 10-40 Effect of connection proximity on messaging engine selection

Connection proximity value	Messaging engine selected
Bus	The JMS client connects to the messaging engine on Server 2, following the rules described in “Bus name” on page 580.
Cluster	The JMS client connect to the messaging engine on Server 3, because this is the only messaging engine in the same cluster as the client.
Host	The JMS client connects to the messaging engine on Server 2, because this is the only messaging engine on the same host as the client.
Server	The JMS client fails to connect to the service integration bus, because there is no messaging engine in the same server as the client.

Target groups

Target groups provide a further means of controlling the selection of a suitable messaging engine by restricting the messaging engines available for consideration during the connection proximity check. Before the connection proximity search is performed, the set of messaging engines which are members of the specified target group is determined. The connection proximity check is then restricted to these messaging engines.

The use of target groups is controlled through the target, target type and target significance properties of the connection factory, the descriptions for which are as follows:

- Target

  The target property identifies a group of messaging engines that should be used when determining the set of available messaging engines. If no target group is specified, then no sub-setting of the available messaging engines takes place and every messaging engine within the bus is considered during the connection proximity check. By default, no target group is specified.

- Target type

  The target type property specifies the type of the group identified by the target property. Valid values for the target type property are:

  - **Bus member name**

    Bus member name indicates that the target property specifies the name of a bus member. Because bus members can only be application servers or application server clusters, the value of the target property must be an

application server name of the form <node name>.<server name> or the name of the cluster.

- **Custom messaging engine group name**

  This value indicates that the target property specifies the name of a user defined custom group of messaging engines. A messaging engine is registered with a custom group by specifying the name of the group in the target groups property for the messaging engine. The registration of the messaging engine takes place when the messaging engine is started.

- **Messaging engine name**

  Choosing this value indicates that the target property specifies the name of a specific messaging engine. This is the most restrictive target type that can be specified.

- Target significance

  The target significance property allows the connection factory to relax the rules that are applied regarding the target group. The valid values for this property are as follows:

  - **Preferred**

    Use Preferred to indicate that a messaging engine be selected from the target group. A messaging engine in the target group is selected if one is available. If a messaging engine in the target group is not available, an available messaging engine within the specified service integration bus, but outside of the target group, is selected.

  - **Required**

    Use Required to indicate that a messaging engine be selected from the target group. A messaging engine in the target group is selected if one is available. If a messaging engine in the target group is not available, the connection process fails.

To see how the values of the target group properties affect the messaging engine selection process, consider the configuration shown in Figure 10-53 on page 587. All of the messaging engines shown in Figure 10-53 exist the same service integration bus.

Archive

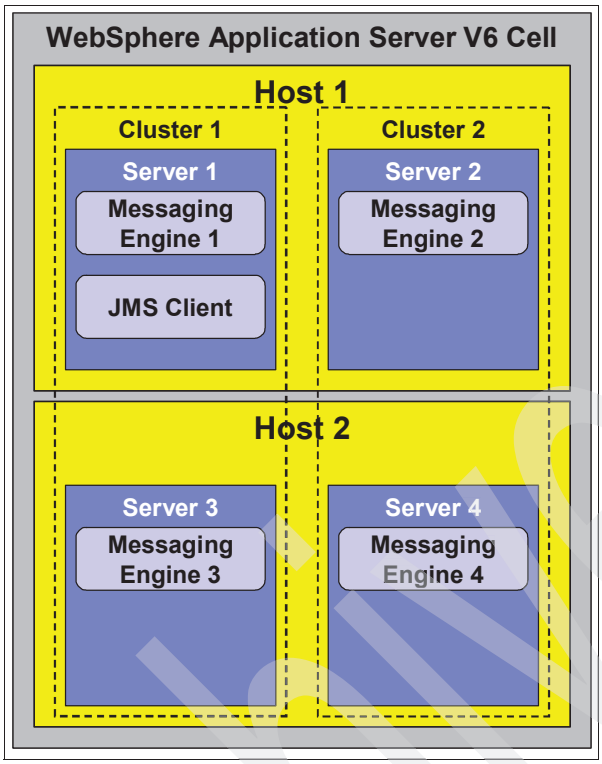


Figure 10-53 Sample topology for a service integration bus

The effect of the value of the connection proximity property on messaging engine selection is described in Table 10-41.

Table 10-41 Effect of target group properties on messaging engine selection

Connection property		Messaging engine selected
Name	**Value**	
Target	Cluster 2	The set of available messaging engines in the target group, Cluster 2, is: {Messaging Engine 2, Messaging Engine 4}. Because a connection proximity of Bus has been specified, the JMS client would connect to Messaging Engine 2. This is the only messaging engine in the set that is on the same host as the client.
Target type	Bus member name	
Target significance	Required	
Connection proximity	Bus	

Archived

Connection property		Messaging engine selected
Name	**Value**	
Target	Cluster 2	The set of available messaging engines in the target group, Cluster 2, is: {Messaging Engine 2, Messaging Engine 4}. Because a connection proximity of Server and a target significance of Required have been specified, the JMS client would fail to connect to the service integration bus, because there are no messaging engines in the target group that are on the same server as the client.
Target type	Bus member name	
Target significance	Required	
Connection proximity	Server	
Target	Cluster 2	By relaxing the target significance to Preferred, the JMS client is now able to connect to an alternative messaging engine that does not necessarily meet the connection proximity constraint. In this case, the JMS client would connect to Messaging Engine 1.
Target type	Bus member name	
Target significance	Preferred	
Connection proximity	Server	

10.7.3 Load balancing bootstrapped clients

JMS clients that connect to a service integration bus using a bootstrap server, which is itself running a suitable messaging engine, always connect to the messaging engine running in the bootstrap server. This is because this messaging engine is the closest suitable messaging engine to the bootstrap server.

Note: The term *suitable messaging engine* describes a messaging engine that matches all of the target group and connection proximity rules described in 10.7.2, “Controlling messaging engine selection” on page 579.

If there are many JMS clients using the same connection factory, they all bootstrap using the same list of bootstrap servers. Because the connection factory attempts to connect to a bootstrap server in the order in which they are specified in the provider endpoints list, it is likely that all of the JMS clients will be connected to the same messaging engine in the first available bootstrap server. The JMS clients will not be load-balanced across the set of suitable messaging engines. This is shown in Figure 10-54 on page 589.

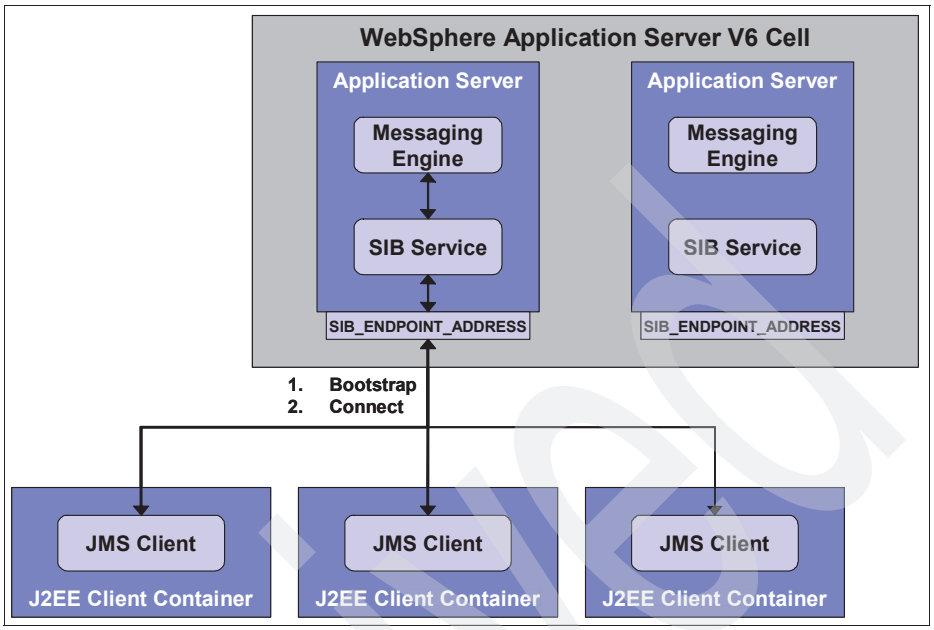


Figure 10-54 Bootstrapped JMS clients connecting to a single messaging engine

A solution to this problem is to make use of a dedicated bootstrap server that is not a running a messaging engine for the target bus. This ensures that the connections established for JMS client are load-balanced across the available messaging engines for the target bus. This is shown in Figure 10-55 on page 590.

We expect that a future release will support the automatic load-balancing of bootstrapped JMS clients across the set of suitable messaging engines thus reducing the tendency for bootstrapped JMS clients to congregate at a single bootstrap server.

Archive

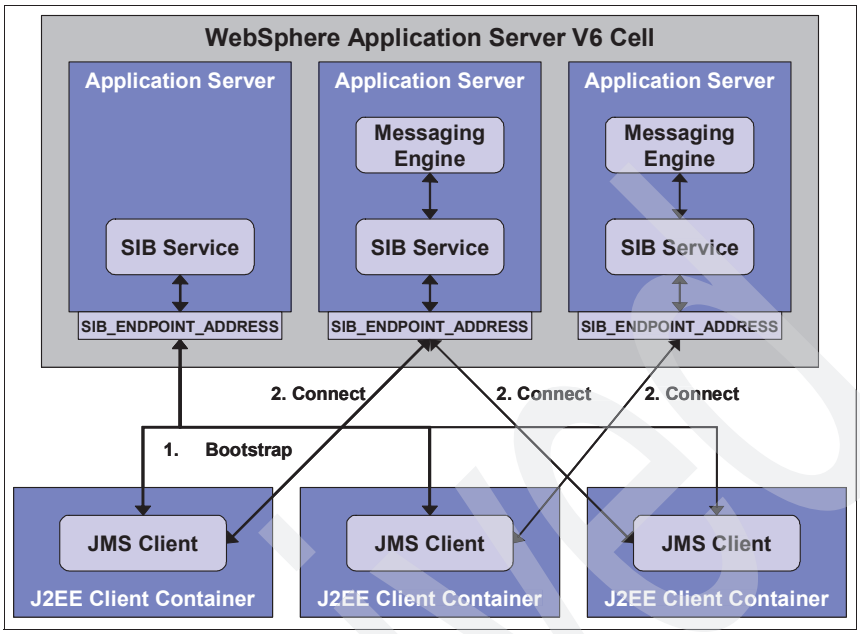


Figure 10-55 Load balancing of connections for bootstrapped JMS clients

10.8 References and resources

These documents and web sites are also relevant as further information sources:

- WebSphere Information Center

  `http://www.ibm.com/software/webservers/appserv/infocenter.html`

- *Java 2 Platform Enterprise Edition Specification, v1.4*

  `http://java.sun.com/j2ee/j2ee-1_4-fr-spec.pdf`

- J2EE Connector Architecture

  `http://java.sun.com/j2ee/connector/`

- *WebSphere MQ Using Java*

  `http://www-306.ibm.com/software/integration/mqfamily/library/manualsa/manuals/crosslatest.html`

- Java Message Service (JMS)

  `http://java.sun.com/products/jms`

Archive

- *Enterprise Messaging Using JMS and WebSphere* (Kareem Yusuf), Prentice Hall, ISBN: 0-13-146863-4
- *Java Message Service* (Monson-Haefel, Chappell), O'Reilly, ISBN: 0-596-00068-5
- *Professional JMS* (Grant, Kovacs, et al), Wrox Press Inc., ISBN: 1861004931
- *Enterprise JavaBeans, Fourth Edition* (Monson-Haefel, Burke, Labourey), O'Reilly, ISBN: 0-596-00530-X
- *EJB Design Patterns* (Marinescu), Wiley, ISBN: 0471208310

Archived

Archived

11

Default messaging provider

WebSphere Application Server V6 introduces a new component called the service integration bus. In this chapter, we describe the concepts behind the service integration bus, focusing on its role as the default messaging provider within WebSphere Application Server V6. We cover:

- Concepts and architecture
- Runtime components
- High availability and workload management
- Service integration bus topologies
- Service integration bus and message-driven beans
- Service integration bus security
- Problem determination
- Configuration and management

© Copyright IBM Corp. 2005. All rights reserved.

11.1 Concepts and architecture

The service integration bus provides a managed communications framework which supports a variety of message distribution models, reliability options and network topologies. It provides support for traditional messaging applications as well as enabling the implementation of service-oriented architectures within the WebSphere Application Server V6 environment.

The service integration bus is the underlying messaging provider for the default messaging JMS provider, replacing the embedded messaging provider that was supported in WebSphere Application Server V5.

The service integration bus introduces a number of new concepts. The sections that follow discuss each of these concepts in more detail.

11.1.1 Buses

A *service integration bus*, or *bus*, within WebSphere Application Server V6 is simply an architectural concept. It gives an administrator the ability to group a collection of resources together that provide the messaging capabilities of the bus. At runtime, the bus presents these cooperating messaging resources to applications as a single entity, hiding from those applications the details of how the bus is configured and where on the bus the different resources are located.

A bus is defined at the cell level within WebSphere Application Server V6. It is anticipated that, in a standard configuration, no more than one service integration bus will be required within a WebSphere Application Server V6 cell. However, a WebSphere cell can contain any number of buses.

Resources are created within, or added to, the scope of a specific bus. Simply defining a bus within a WebSphere cell has no runtime impact on any of the components running within a cell. It is not until members are added to a bus that any of the runtime components within an application server are affected.

Figure 11-1 on page 595 shows a service integration bus defined within a WebSphere Application Server V6 cell.

Archived

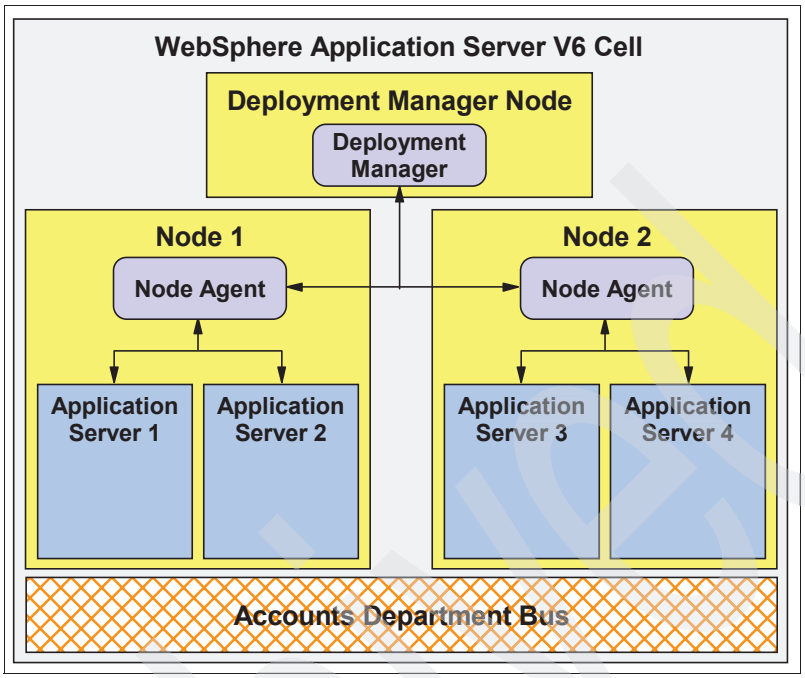


Figure 11-1 Service integration buses within a WebSphere Application Server V6 cell

11.1.2 Bus members

A *bus member* is simply an application server, or cluster of application servers, that has been added as a member of a bus. Adding an application server, or cluster of application servers, as a member of a bus automatically defines a number of resources on the bus member in question. In terms of the functionality provided by a service integration bus, the most important of the resources that are automatically defined is a messaging engine.

11.1.3 Messaging engines

A *messaging engine* is the component within an application server that provides the core messaging functionality of a service integration bus. At runtime, it is the messaging engines within a bus that communicate and cooperate with each other to provide the messaging capabilities of the bus. A messaging engine is responsible for managing the resources of the bus and it also provides a connection point to which local and remote client applications can connect.

A messaging engine is associated with a bus member. When an application server is added as a member of a bus, a messaging engine is automatically

created and associated with this application server. Figure 11-2 on page 596 shows a WebSphere Application Server V6 cell that contains two buses, each of which has two application servers defined as bus members.

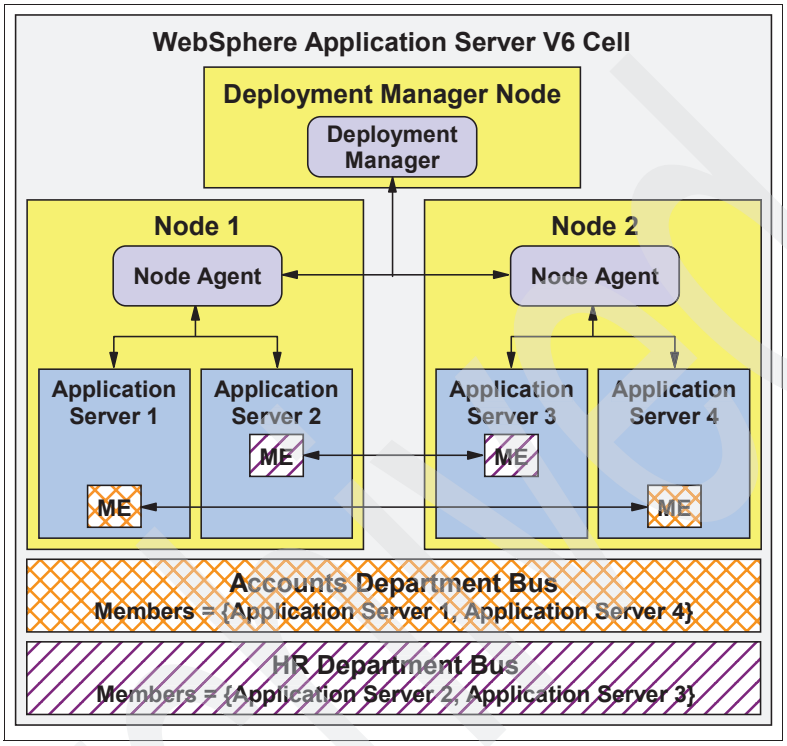


Figure 11-2 Messaging engines within service integration bus members

A messaging engine is a relatively lightweight runtime object. This allows a single application server to host several messaging engines. If an application server is added as a member of multiple buses, that application server is associated with multiple messaging engines, one messaging engine for each bus of which it is a member. This is shown in Figure 11-3 on page 597.

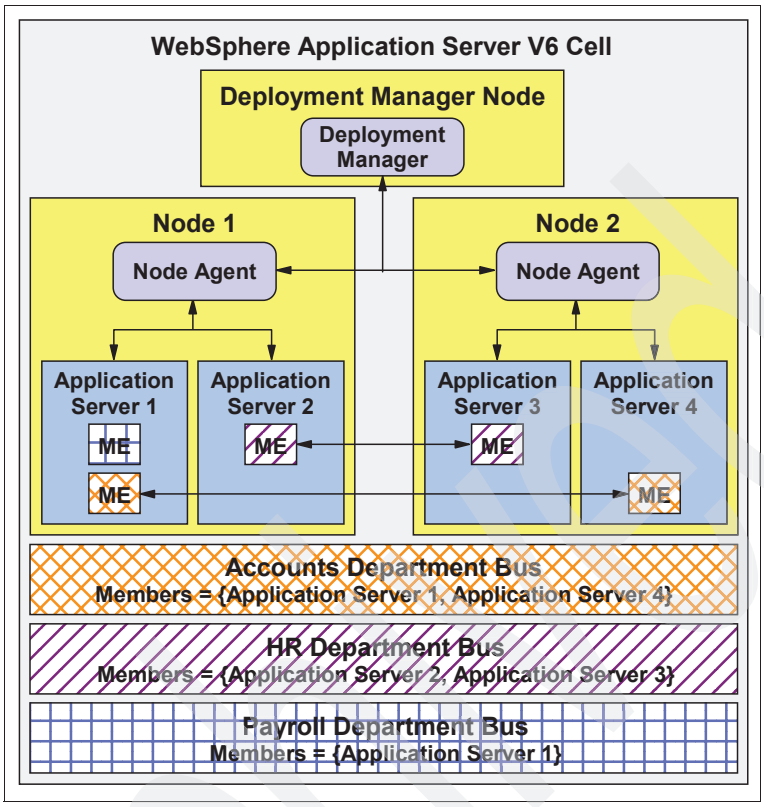


Figure 11-3 Multiple messaging engines within a single application server

When a cluster of application servers is added as a member of bus, a single messaging engine is automatically created and associated with the application server cluster, regardless of the number of application servers defined as members of the cluster. At runtime, this messaging engine is activated within a single application server within the cluster. The application server that is chosen to host the messaging engine will be the first cluster member to start. This is shown in Figure 11-4.

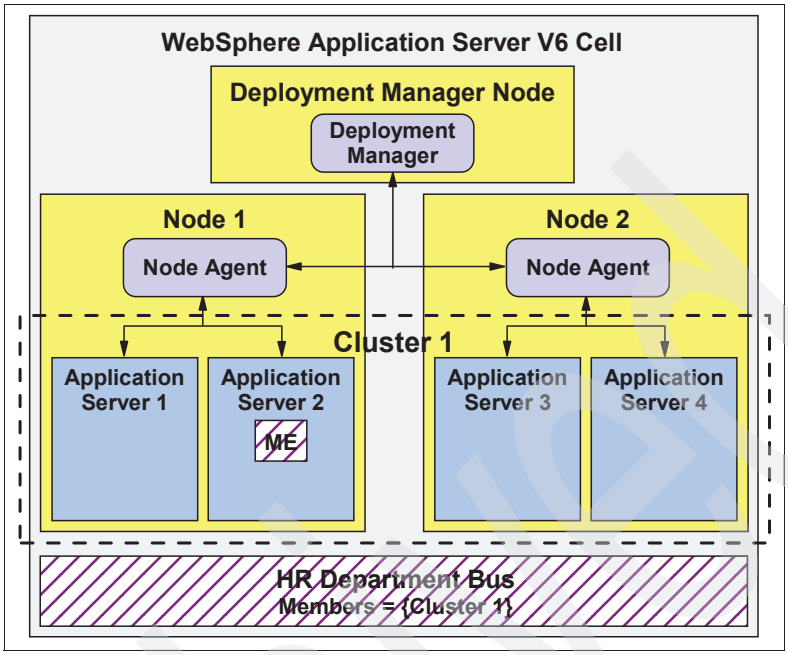


Figure 11-4 An application server cluster as a bus member

However, this messaging engine is able to run within any of the application servers defined as members of the cluster. If the messaging engine, or the application server within which it is running, should fail, the messaging engine is activated on another available server in the cluster. Therefore, adding an application server cluster as a member of a bus enables failover for messaging engines that are associated with that cluster. This is shown in Figure 11-5 on page 599.

Aiohn

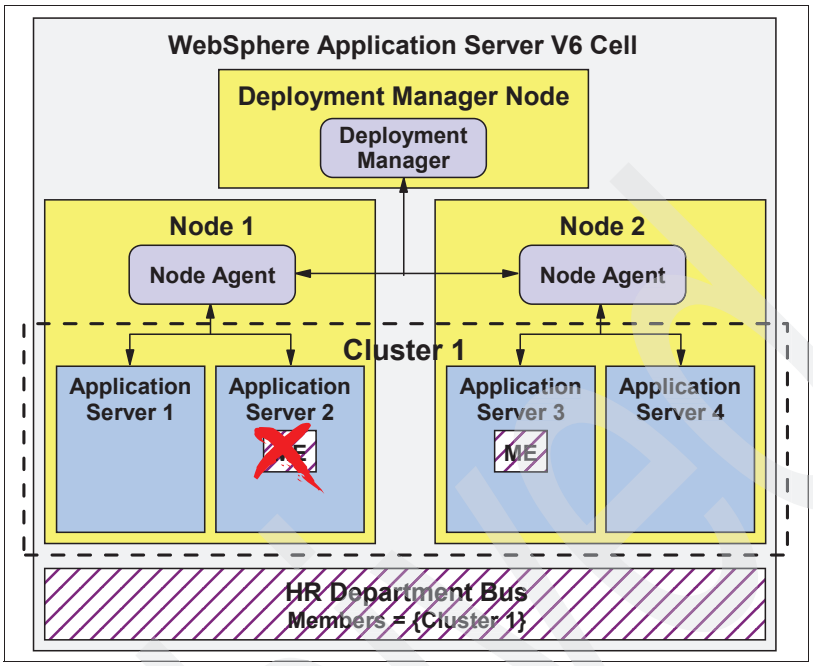


Figure 11-5 Messaging engine fail-over within an application server cluster

Once an application server cluster has been added as a member of a bus, it is also possible to create additional messaging engines and associate them with the cluster. These additional messaging engines can then be configured to run within a specific cluster member, if required. Such a configuration enables a bus to be scaled to meet the needs of applications that generate high message volumes. It also improves the availability of the bus in question. This is shown in Figure 11-6 on page 600.

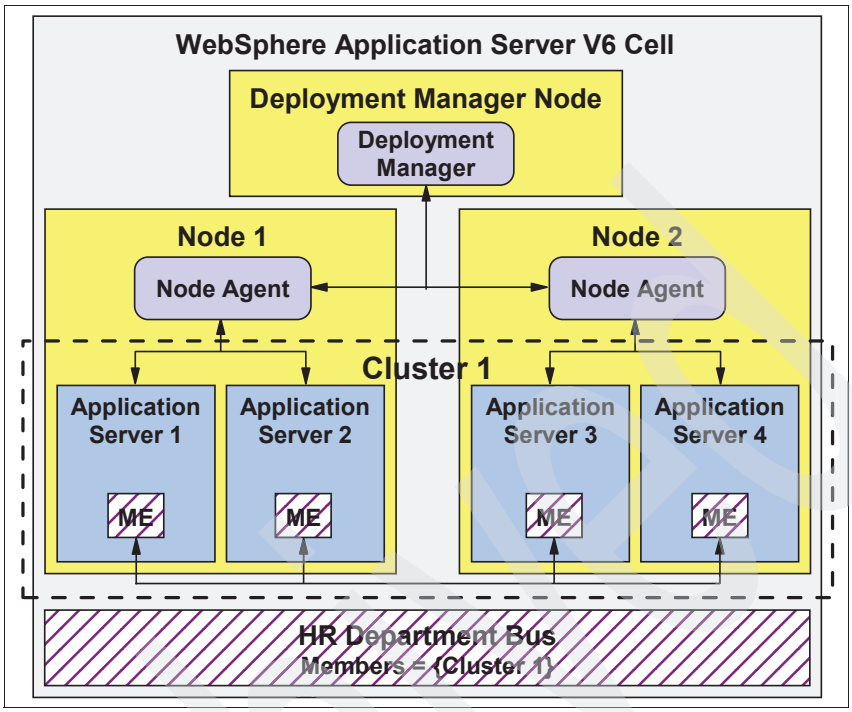


Figure 11-6 Messaging engine scalability within an application server cluster

For more information about fail-over and scalability within the service integration bus, refer to 11.3, “High availability and workload management” on page 638.

Messaging engine naming

As discussed previously, when a member is added to a service integration bus, a messaging engine is automatically created and associated with the new bus member. The name of the new messaging engine is generated based on the details of the new bus member, as follows:

- Application server bus members

  The format of the messaging engine name generated when an application server is added as a member of a bus is as follows:

  <Node Name>.<Server Name>-<Bus Name>

  The elements are defined as:

  – <Node Name> is the name of the node on which the new bus member is defined.
  – <Server Name> is the name of the new application server bus member.

– <Bus Name> is the name of the service integration bus to which the new bus member has been added.

We can use it in an example, such as:

ITSONode.Server 1-ITSOBus

- Application server cluster bus members

  The format of the messaging engine name generated when an application server cluster is added as a member of a bus is as follows:

  <Cluster Name>.XXX-<Bus Name>

  The elements of this format are:

  – <Cluster Name> is the name of the new application server cluster bus member.
  – XXX is a number that is used to uniquely identify the messaging engine within the cluster. This value is incremented each time a new messaging engine is added to the cluster.
  – <Bus Name> is the name of the service integration bus to which the new bus member has been added.

  We can use it in an example, such as:

  ITSOCluster.000-ITSOBus

11.1.4 Data stores

Every messaging engine defined within a bus has a *data store* associated with it. A messaging engine uses this data store to persist durable data, such as persistent messages and transaction states. Durable data written to the data store survives the orderly shutdown, or failure, of a messaging engine, regardless of the reason for the failure.

It can also use the data store to reduce runtime resource consumption. For example, the messaging engine can write non-persistent messages to the data store in order to reduce the size of the Java heap when handling high message volumes. This is known as *spilling*.

Figure 11-7 on page 602, shows messaging engines associated with data stores. Two of the messaging engines shown in Figure 11-7 are associated with data stores that exist within the same database. There are certain considerations you must take into account when deciding the data store topology. These considerations are discussed in more detail in 11.2.3, “Data stores” on page 620, as part of the description of the runtime components of the service integration bus.

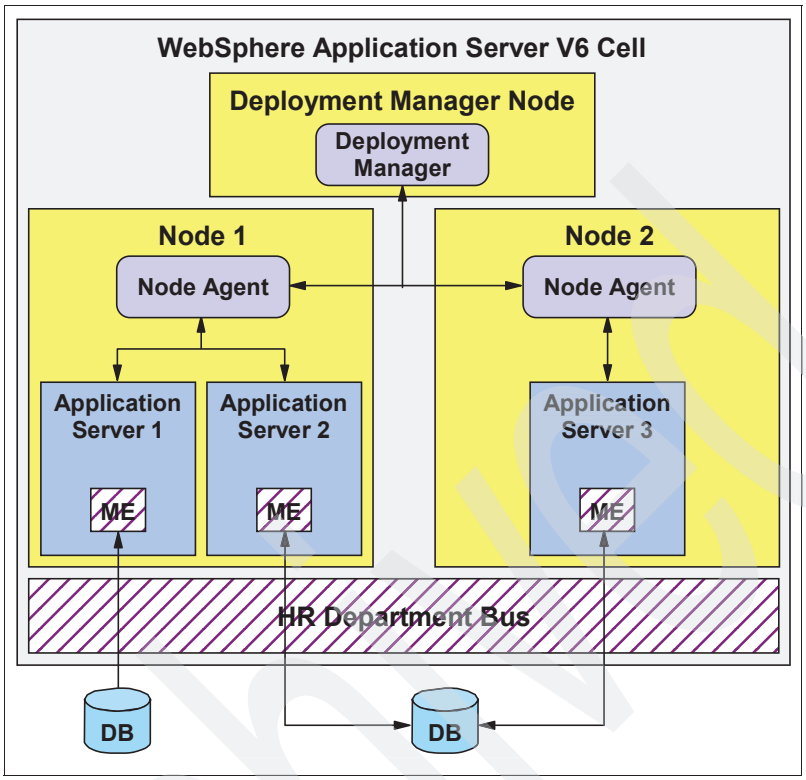


Figure 11-7 Messaging engine data stores

11.1.5 Destinations

A *destination* within a service integration bus is a logical address to which applications can attach as message producers, message consumers, or both, in order to exchange messages. The main types of destination that can be configured on a bus are:

- Queue destinations

  Queue destinations are destinations that can be configured for point-to-point messaging.

- Topic space destinations

  Topic space destinations are destinations that can be configured for publish/subscribe messaging.

- Alias destinations

  Alias destinations are destinations that can be configured to refer to another destination, potentially on a foreign bus. They can provide an extra level of

indirection for messaging applications. An alias destination can also be used to override some of the values specified on the target destination, such as default reliability and maximum reliability. Foreign buses are discussed in section 11.1.7, “Foreign buses” on page 606.

- Foreign destinations

  Foreign destinations are not destinations within a service integration bus, but they can be used override the default reliability and maximum reliability properties of a destination that exists on a foreign bus. Foreign buses are discussed in section 11.1.7, “Foreign buses” on page 606.

Message points

Recall that a service integration bus is simply an architectural concept within a WebSphere Application Server V6 cell. Similarly, when a destination is configured on a service integration bus, it simply defines a logical address to which applications can attach. Queue and topic space destinations must be associated with a messaging engine in order for any persistent messages directed at those destinations to be persisted to an underlying data store. These destinations are associated with a messaging engine using a *message point*. A message point is a physical representation of a destination defined on a bus. A message point can be configured to override some of the properties inherited from the bus destination.

The two main types of message point that can be contained with a messaging engine are:

- Queue points

  A *queue point* is the message point for a queue destination. When creating a queue destination on a bus, an administrator specifies the bus member which will hold the messages for the queue. This action automatically defines a queue point for each messaging engine associated with the specified bus member.

  If the bus member is an application server, a single queue point will be created and associated with the messaging engine on that application server. All of the messages that are sent to the queue destination will be handled by this messaging engine. In this configuration, message ordering is maintained on the queue destination.

  If the bus member is a cluster of application servers, a queue point is created and associated with each messaging engine defined within the bus member. The queue destination is partitioned across the available messaging engines within the cluster. In this configuration, message ordering is not maintained on the queue destination. For more information about partitioned destinations within the service integration bus, please refer to 11.3, “High availability and workload management” on page 638.

- Publication points

  A *publication point* is the message point for a topic space. When creating a topic space destination, an administrator does not need to specify a bus member to hold messages for the topic space. Creating a topic space destination automatically defines a publication point on each messaging engine within the bus.

Figure 11-8 on page 604 shows a queue destination and a topic space destination and their associated queue and publication points.

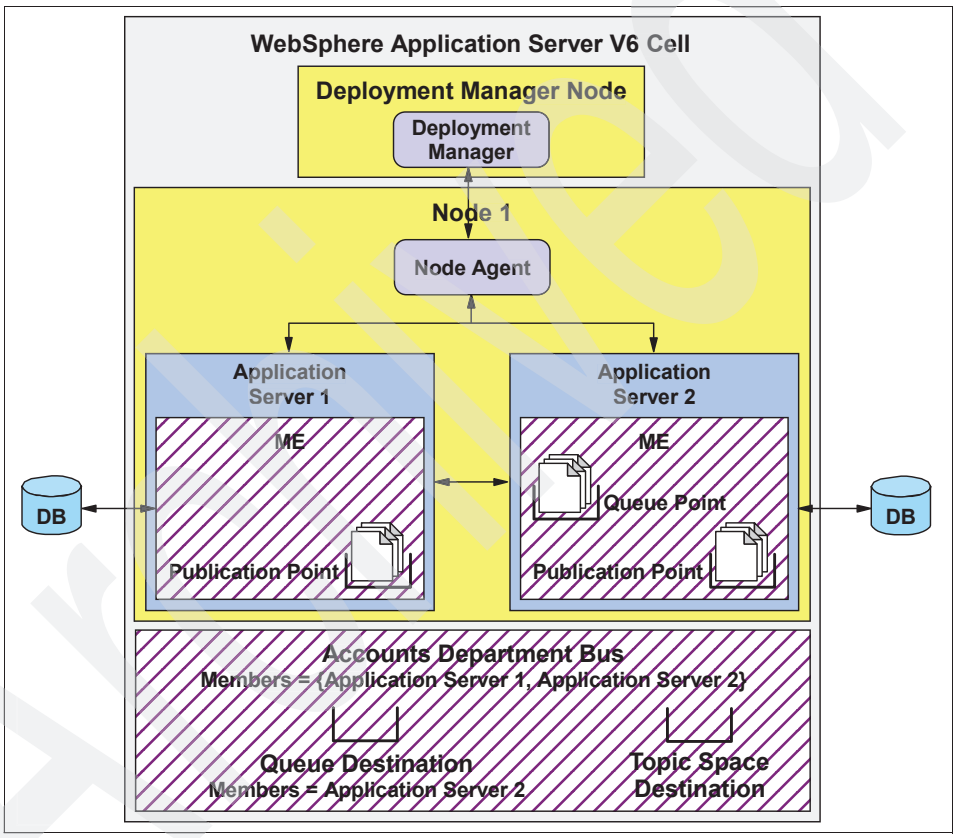


Figure 11-8 Queue and publication points in the service integration bus

Reliability

It is on a destination that an administrator specifies the default quality of service levels that will be applied when a message producer or message consumer interacts with the destination. An administrator is able to configure a default reliability and a maximum reliability for each service integration bus destination.

There are five levels of reliability that can be specified for these properties. These are described in Table 11-1.

Table 11-1 Service integration bus destination reliabilities

Least reliable



Most reliable

Reliability	Description
Best Effort nonpersistent	Messages that are sent to this destination are discarded when the messaging engine with which it associated is stopped, or if it fails. Messages can also be discarded if the connection used to send them becomes unavailable or as a result of constrained system resources. Messages delivered asynchronously to non-transactional MessageListeners or message-driven beans will not be redelivered if an exception is thrown.
Express nonpersistent	Messages that are sent to this destination are discarded when the messaging engine with which it is associated is stopped or if it fails. Messages can also be discarded if the connection used to send them becomes unavailable.
Reliable nonpersistent	Messages that are sent to this destination are discarded when the messaging engine with which it is associated is stopped or if it fails.
Reliable persistent	Messages that are sent to this destination can be discarded when the messaging engine with which it is associated fails, but are persisted if the messaging engine is stopped normally.
Assured persistent	Messages that are sent to this destination are never discarded.

Note: Reliability should be chosen according to your messaging needs. More reliable qualities of service might not perform as well as less reliable qualities of service.

Administrators can also allow message producers to override the default reliability that is specified on a destination. The mechanism that is used to achieve this depends on the type of the message producer. For instance, a JMS message producer can use the quality of service properties on the default messaging JMS provider connection factory to map the JMS PERSISTENT and NON_PERSISTENT delivery modes onto the required service integration bus reliabilities. This is discussed in more detail in “Quality of service properties” on page 530.

Note: The reliability specified by a message producer can never exceed the maximum reliability specified on a service integration bus destination. In the case of a JMS message producer, attempting to do this will cause a JMS exception to be thrown to the client application.

11.1.6 Mediations

A *mediation* processes in-flight messages between the production of a message by one application, and the consumption of a message by another application. Mediations enable the messaging behavior of a service integration bus to be customized. Examples of the processing that can be performed by a mediation are:

- Transforming a message from one format into another
- Routing messages to one or more target destinations that were not specified by the sending application
- Augmenting messages by adding data from a data source
- Distributing messages to multiple target destinations
- Discarding messages

A mediation is defined within a specific service integration bus. This mediation can then be associated with a destination on the bus. A corresponding *mediation point* is automatically created and associated with the destination as a result of this process. A mediation point is a specialized type of message point. A destination with which the mediation is associated is referred to as a *mediated destination*.

11.1.7 Foreign buses

A service integration bus can be configured to connect to, and exchange messages with, other messaging networks. In order to do this, a *foreign bus* must be configured.

A foreign bus encapsulates information related to the remote messaging network, such as the type of the foreign bus and whether messaging applications are allowed to send messages to the foreign bus. A foreign bus can represent:

- A service integration bus in the same WebSphere Application Server V6 cell as the local bus
- A service integration bus in a different WebSphere Application Server V6 cell from the local bus
- A WebSphere MQ network

The ability of a service integration bus to be able to communicate with other messaging networks provides several benefits, examples of which are:

- It enables the separation of resources for different messaging applications which only need to communicate with each other infrequently. This simplifies the administration of the resources for each individual messaging application.
- It enables a bus to be integrated with preexisting messaging networks.

When buses are interconnected, applications can send messages to destinations that are defined on other buses. Published messages can also span multiple buses, if the links between the buses are configured to allow it.

Note: Care must be taken to avoid creating circular link dependencies (Bus A → Bus B → Bus C → Bus A), when configuring foreign buses within complex topologies. Circular links are not supported by the service integration bus.

Routing definition types

During foreign bus configuration, an administrator defines a routing definition that specifies the type of the foreign bus. This information is used at runtime to determine the protocol that will be used to communicate with the foreign bus. The three types of routing definition that can be defined are:

- Direct, service integration bus link

  This routing definition type indicates that the local bus will connect directly to another service integration bus. This is shown in Figure 11-9, where the Accounts Department Bus is linked to the HR Department Bus within its own WebSphere Application Server V6 cell and the Payroll Department Bus within another WebSphere Application Server V6 cell.

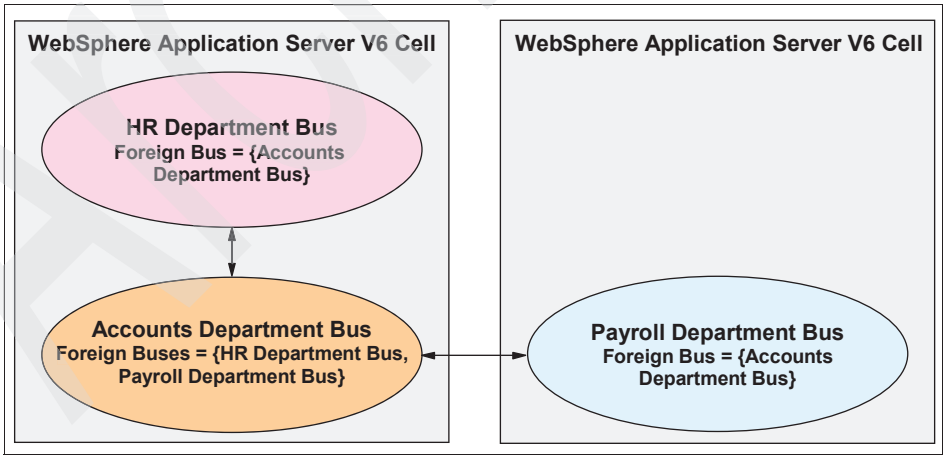


Figure 11-9 Direct, service integration bus links

- Direct, WebSphere MQ link

  This routing definition type indicates that the local bus will connect directly to a WebSphere MQ gateway queue manager. This WebSphere MQ queue manager might itself be connected to several other queue managers in a WebSphere MQ network. This is shown in Figure 11-10.

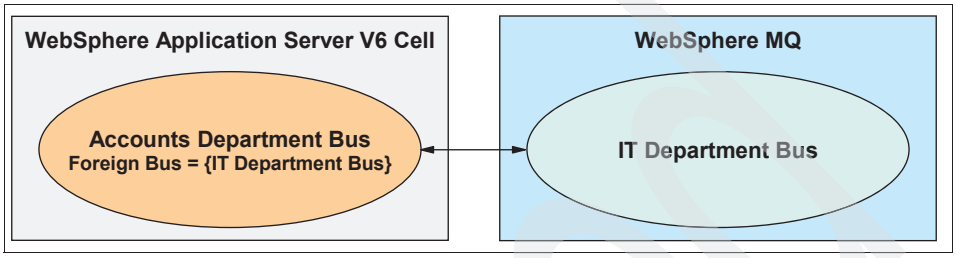


Figure 11-10 Direct, WebSphere MQ link

- Indirect

  The indirect routing definition type indicates that the foreign bus being configured is not directly connected to the local bus. In this situation, the administrator specifies the name of the next bus in the route. This bus can be another service integration bus or a WebSphere MQ network, but it must already be defined in order to configure an indirect routing definition. Ultimately, a message could travel through several intermediate buses before it reaches its destination.

  This is shown in Figure 11-11, where the Accounts Department Bus is linked indirectly to the Payroll Department Bus via the HR Department Bus.

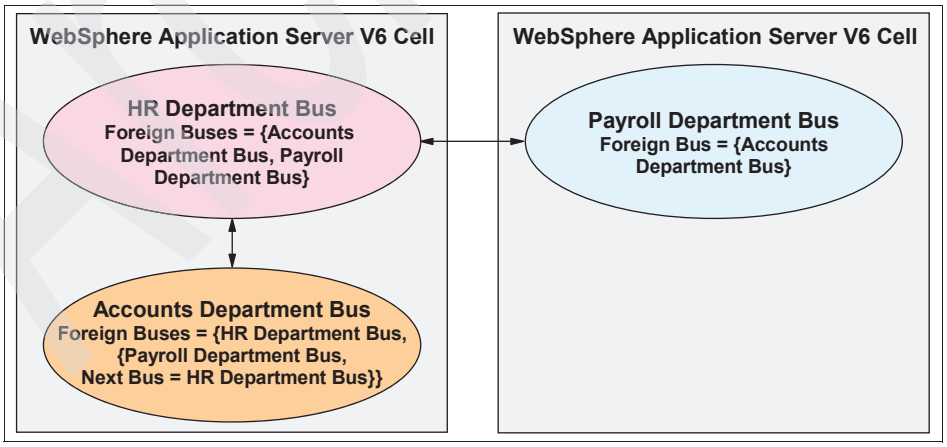


Figure 11-11 Indirect foreign bus link

Foreign bus links

Recall that a service integration bus is simply an architectural concept within a WebSphere Application Server V6 cell. Similarly, when a foreign bus is configured on a service integration bus, it simply describes a link between the two buses at an architectural level.

In order for the two buses to be able to communicate with each other at runtime, links must be configured between a specific messaging engine within the local bus and a specific messaging engine, or queue manager, within the foreign bus. When configuring a direct service integration bus link, these links must be configured in both directions in order for the two buses to be able to communicate. At runtime, messages that are routed to a foreign bus will flow across the corresponding link. This is shown in Figure 11-12.

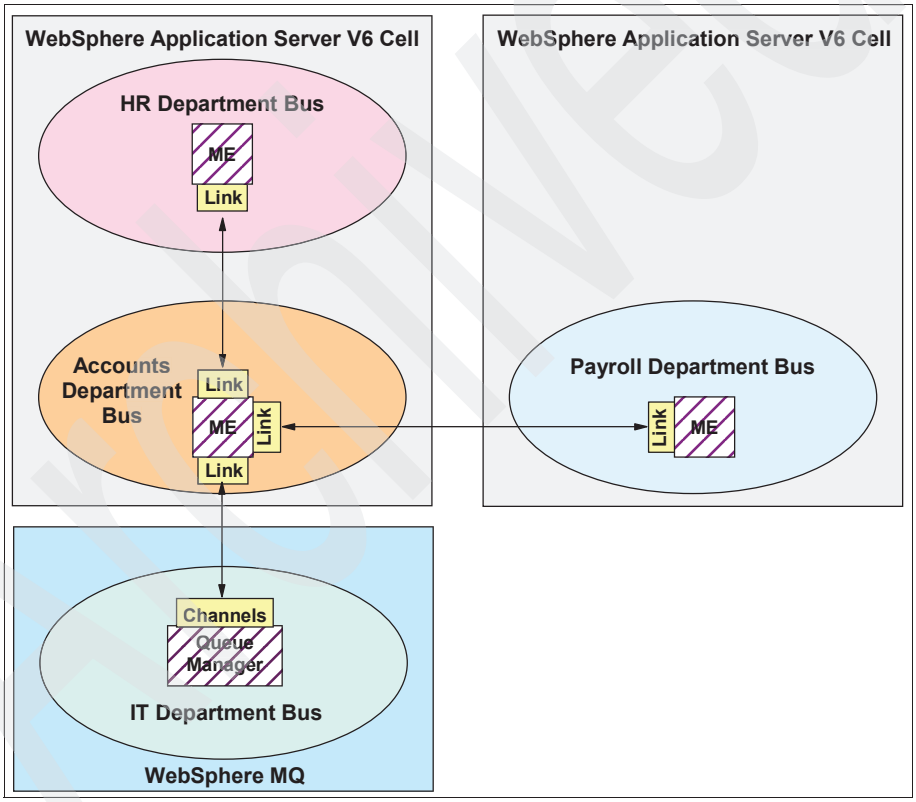


Figure 11-12 Runtime view of foreign buses

Note: It is not possible to define multiple links between the local bus and a specific foreign bus.

Foreign buses and point-to-point messaging

Messaging applications that make use of the Point-to-Point messaging model, with destinations that are defined on a local bus, are able to act as both message producers and message consumers. This is shown in Figure 11-13.

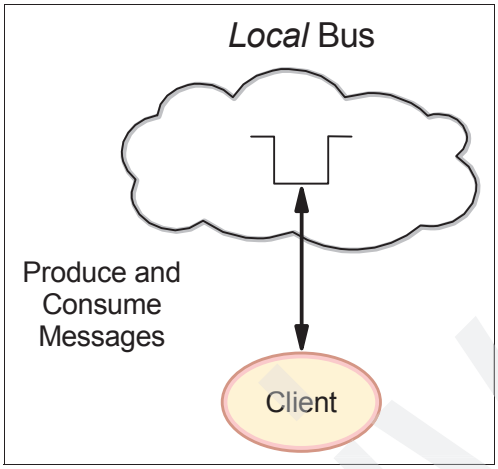


Figure 11-13 Point-to-point messaging on the local bus

However, when a messaging application is making use of the Point-to-Point messaging model with destinations that are defined on a foreign bus, it is only able to act as a message producer. This is shown in Figure 11-14.

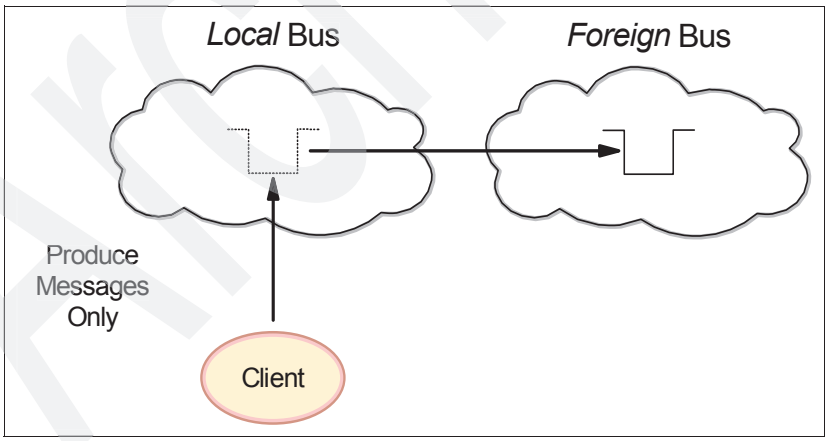


Figure 11-14 Point-to-point message producer for a foreign bus

If a messaging application is required to consume messages from a destination that is defined on a foreign bus, the messaging application must connect directly to the foreign bus. This is shown in Figure 11-15 on page 611.

Archived

This is similar to the restrictions placed on WebSphere MQ messaging clients, where a client application is only able to consume messages from a queue by connecting directly to the WebSphere MQ queue manager on which the queue is defined.

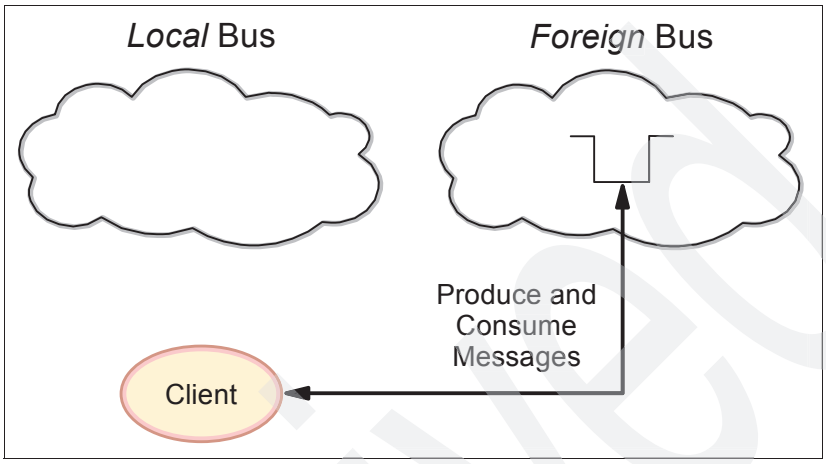


Figure 11-15 Point-to-point messaging on a foreign bus

If the messaging application is unable to connect directly to the foreign bus, then the destinations on the foreign bus must be configured to forward messages to destinations on the local bus. The messaging application is then able to connect to the local bus to consume the messages. This is shown in Figure 11-16.

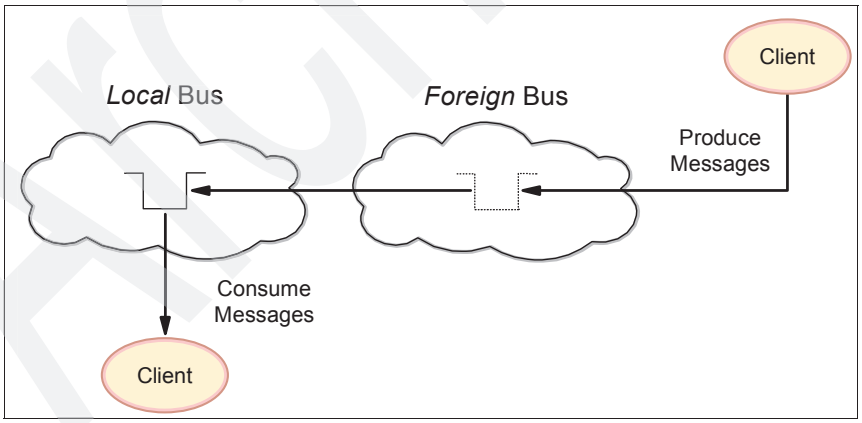


Figure 11-16 Forwarding messages for consumption from the local bus

Archived

Foreign buses and Publish/Subscribe messaging

By default, foreign bus links will not flow messages that are produced by messaging applications using the Publish/Subscribe messaging model. It is possible to configure a foreign bus link such that messages published to topic spaces on the local bus will be published on the foreign bus.

11.2 Runtime components

At runtime, a service integration bus is comprised of a collection of cooperating messaging resources. The sections that follow describe the runtime aspects of these messaging resources in more detail.

11.2.1 SIB service

The *SIB service* is a WebSphere Application Server V6 component that is responsible for managing all of the messaging resources that have been associated with a particular application server. However, the SIB service is not associated with a specific service integration bus or messaging engine. Its management tasks include:

- Managing the life cycle of any messaging related transport chains that have defined within the application server
- Handling inbound connection requests from external messaging applications

Figure 11-17 shows a SIB service within an application server environment.

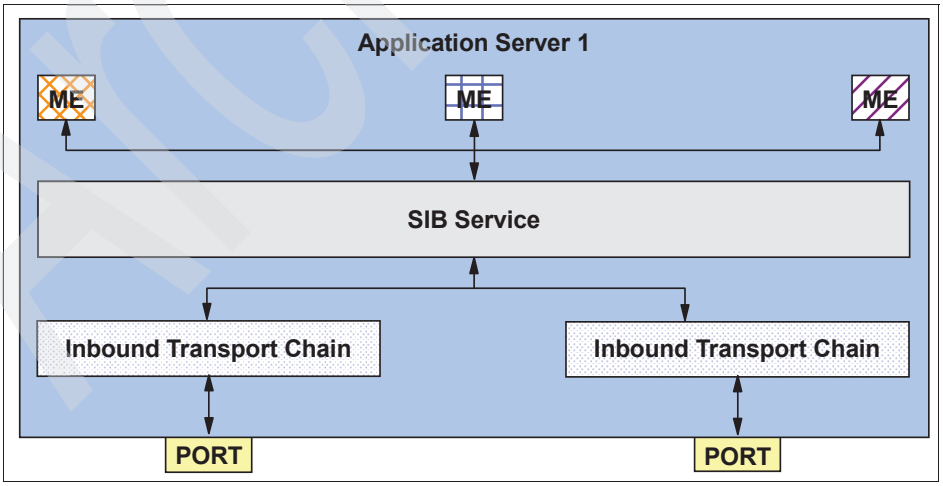


Figure 11-17 SIB service

Every application server has exactly one SIB service. However, by default the SIB service within an application server is disabled. This ensures that the SIB service does not consume resources unnecessarily, if the application server is not hosting any messaging resources.

The process of adding an application server as a member of a service integration bus automatically enables its SIB service. This ensures that the SIB service is available to manage the messaging resources that are created as a result of adding the application server as a bus member.

The SIB service can also be manually enabled within an application server that is not a member of a service integration bus. An application server configured in this manner is able to act as a bootstrap server for clients that are running outside of the WebSphere Application Server V6 environment, or for messaging engines that are running in a different WebSphere Application Server V6 cell. Refer to section 10.7, “Connecting to a service integration bus” on page 576 for more information regarding bootstrap servers.

Configuration reload

The SIB service also allows certain configuration changes to be applied to a service integration bus, without requiring a restart of the application servers that are hosting components associated with that bus. The configuration changes that can be applied without an application server restart are:

- Creation, modification or deletion of a destination
- Creation, modification or deletion of a mediation

For example, if a new destination is created on a service integration bus, that destination can be made available for use without needing to restart application servers or messaging engines associated with the bus.

However, the configuration changes that require the affected application servers or messaging engines to be restarted before the changes come into effect include:

- Creation of a new service integration bus
- Creation of a new messaging engine
- Creation of a service integration bus link
- Creation of a WebSphere MQ link

Note: Each service integration bus that requires this functionality must also be configured to support configuration reload. By default, each service integration bus has configuration reload support enabled. See Section 11.8.1, “SIB service configuration” on page 655 for more information.

11.2.2 Service integration bus transport chains

The SIB service and any messaging engines running within an application server make use of a variety of transport chains in order to communicate with each other and with client applications. The sections that follow describe the inbound and outbound transport chains used by service integration bus components.

Inbound transport chains

When an application server is created using the default template, a number of inbound transport chains are automatically defined. These transport chains enable messaging clients to communicate with a messaging engine. A messaging client can be a client application or another messaging engine. Table 11-2 describes these transport chains.

Table 11-2 Messaging engine inbound transport chains

Transport chain & associated port	Default port	Client Types	Description
InboundBasicMessaging SIB_ENDPOINT_ADDRESS	7276	Remote messaging engines JMS client applications running in the J2EE client container and using the default messaging JMS provider	This chain allows clients of the specified type to communicate with a messaging engine using the TCP protocol
InboundSecureMessaging SIB_ENDPOINT_SECURE_ADDRESS	7286	Remote messaging engines JMS client applications running in the J2EE client container and using the default messaging JMS provider	This chain allows clients of the specified type to communicate securely with a messaging engine using the secure sockets layer (SSL) protocol over a TCP connection. The SSL configuration information for this chain is based on the default SSL repertoire for the application server.
InboundBasicMQLink SIB_MQ_ENDPOINT_ADDRESS	5558	WebSphere MQ queue manager sender channels JMS client applications running in the J2EE client container and using the WebSphere MQ JMS provider	This chain allows clients of the specified type to communicate with a messaging engine using the TCP protocol.

Archive

Transport chain & associated port	Default port	Client Types	Description
InboundSecureMQLink SIB_MQ_ENDPOINT_ SECURE_ADDRESS	5578	WebSphere MQ queue manager sender channels JMS client applications running in the J2EE client container and using the WebSphere MQ JMS provider	This chain allows clients of the specified type to communicate securely with a messaging engine using the secure sockets layer (SSL) protocol over a TCP connection. The SSL configuration information for this chain is based on the default SSL repertoire for the application server.

Note: Table 11-2 describes the default port numbers that are associated with the ports:

- SIB_ENDPOINT_ADDRESS
- SIB_ENDPOINT_SECURE_ADDRESS
- SIB_MQ_ENDPOINT_ADDRESS
- SIB_MQ_ENDPOINT_SECURE_ADDRESS

However, on nodes that are hosting multiple application servers, these ports might have been automatically configured to use different port numbers to avoid conflicts with existing application servers.

As discussed in section 11.2.1, “SIB service” on page 612, the SIB service is responsible for managing the life cycle of the messaging-related inbound transport chains within an application server. Certain transport chains can be started even if the application server is not hosting any messaging engines. When a transport chain starts, it binds to the TCP port to which it has been assigned and listens for network connections. Table 11-3 describes the circumstances under which the inbound transport chains are started by the SIB service.

Table 11-3 Default transport chain initialization during application server startup

Application server configuration	Transport chains	
	InboundBasicMessaging InboundSecureMessaging	InboundBasicMQLink InboundSecureMQLink
SIB service disabled	Not started	Not started

Application server configuration	Transport chains	
	InboundBasicMessaging InboundSecureMessaging	InboundBasicMQLink InboundSecureMQLink
SIB service enabled No WebSphere MQ links No WebSphere MQ client links	Started	Not started
SIB service enabled WebSphere MQ links or WebSphere MQ client links defined	Started	Started

Figure 11-18 shows the InboundBasicMessaging and InboundSecureMessaging transport chains, and the corresponding ports that they are bound to, within an application server.

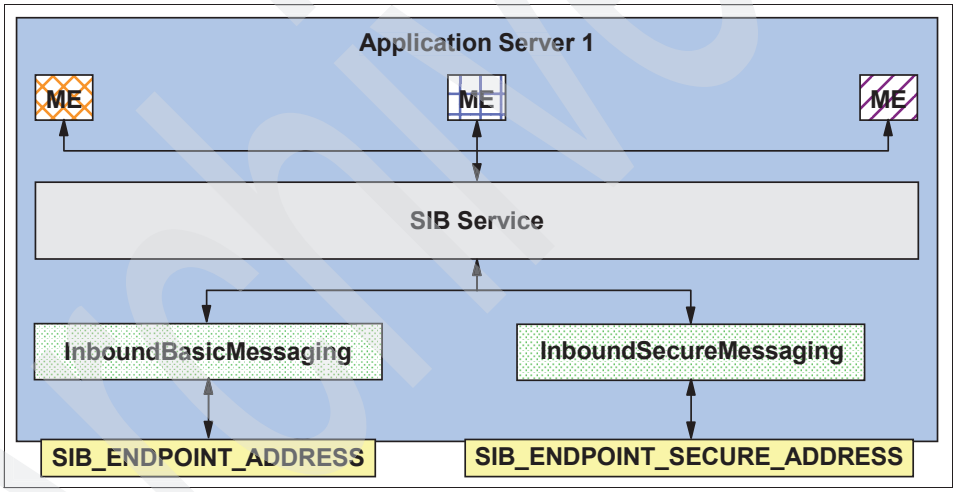


Figure 11-18 Messaging engine inbound transport chains

Outbound transport chains

When you create an application server using the default template, a number of outbound transport chains are automatically defined. These transport chains are also available to JMS client applications running within the J2EE client container. Outbound transport chains are used by messaging clients to establish network connections to bootstrap servers or to WebSphere MQ queue manager receiver channels. Table 11-4 on page 617 describes these transport chains.

Table 11-4 Default messaging engine outbound transport chains

Transport chain	Description
BootstrapBasicMessaging	This chain is suitable for establishing a bootstrap connection to inbound transport chains within an application server that are configured to use the TCP protocol. An example of such a transport chain is the InboundBasicMessaging chain.
BootstrapSecureMessaging	This chain is suitable for establishing a bootstrap connection to inbound transport chains within an application server that are configured to use SSL over a TCP connection. An example of such a transport chain is the InboundSecureMessaging transport chain. Success in establishing such a connection is dependent on a suitably compatible set of SSL credentials being associated with both this bootstrap outbound transport chain and also the inbound transport chain to which it is connecting. The SSL configuration used is taken from the default SSL repertoire of the application server within which the messaging client is running, or from the relevant configuration file if the messaging client is running within the J2EE client container.
BootstrapTunneledMessaging	This chain can be used to tunnel a bootstrap request through the Hypertext Transfer Protocol (HTTP). Before this transport can be used, a corresponding inbound transport chain must be configured on the bootstrap server.

Archive

Transport chain	Description
BootstrapTunneledSecureMessaging	This chain can be used to tunnel a secure bootstrap request through the Hypertext Transfer Protocol (HTTPS). Success in establishing such a connection is dependent on a suitably compatible set of SSL credentials being associated with both this bootstrap outbound transport chain and also the inbound transport chain to which it is connecting. The SSL configuration used is taken from the default SSL repertoire of the application server within which the messaging client is running, or from the relevant configuration file if the messaging client is running within the J2EE client container. Before this transport can be used, a corresponding inbound transport chain must be configured on the bootstrap server.
OutboundBasicMQLink	This chain is suitable for establishing a connection to a WebSphere MQ queue manager receiver channel using the TCP protocol.
OutboundSecureMQLink	This chain is suitable for establishing a secure connection to a WebSphere MQ queue manager receiver channel that has been configured to accept SSL connections. Success in establishing such a connection is dependent on a suitably compatible set of SSL credentials being associated with both this outbound transport chain and also the WebSphere MQ receiver channel to which it is connecting. The SSL configuration for the outbound transport chain is taken from the default SSL repertoire of the application server that is attempting to contact the WebSphere MQ queue manger receiver channel.

When attempting to establish a network connection, a messaging client must use an outbound transport chain suitable for connecting to the corresponding target. For instance, the BootstrapTunneledMessaging transport chain can only be used to connect to an inbound transport chain that supports bootstrap requests tunneled over the HTTP protocol. Similarly, the OutboundBasicMQLink can only be used to connect to a WebSphere MQ queue manager receiver channel. Refer to section 10.7, “Connecting to a service integration bus” on page 576 for more information regarding bootstrap servers.

Configuring outbound transport chains within an application server used for bootstrap purposes, is considered to be an advanced administrative task. For this reason, these transport chains can only be altered, or new bootstrap transport chains defined, using the wsadmin command line environment.

Outbound transport chains within the J2EE client container environment that are used for bootstrap purpose are not configurable. However, certain attributes of the outbound transport chains that are used to establish SSL connections can be customized.

Secure transport considerations

As discussed previously, additional considerations need to be taken into account when using a transport chain that makes use of the SSL protocol to encrypt the traffic that flows over the connection.

Establishing an SSL or HTTPS connection between messaging engines, or between a messaging engine and a JMS application running within the J2EE client container, requires a set of compatible credentials to be supplied by both the party initiating the connection, and the party accepting the connection.

Within an application server environment, the credentials used by a secure transport chain can be configured by associating the required SSL repertoire with the relevant SSL channel within the chain. For inbound transport chains, this can be performed using the WebSphere administrative console. By default, secure transport chains within an application server environment are associated with the default SSL repertoire for the WebSphere Application Server V6 cell. When configuring secure communications between two messaging engines, the name of the inbound transport chain on both messaging engines must match in order for the connection to be established. These transport chains must also be configured with compatible SSL credentials. This is true when securing both intra-bus messaging engine connections and inter-bus messaging engine connections.

Within the J2EE client container environment, the credentials used by a secure outbound transport chain are specified in the sib.client.ssl.properties file. Every WebSphere Application Server V6 profile has its own copy of this file, contained in the properties subdirectory of the profile. The properties contained within this file specify, among other things, the location of the key store and trust store to be used by the outbound transport chain, when attempting to establish a secure connection to a messaging engine.

Note: Any messaging engine that is active on an application server can be contacted by any enabled inbound transport chain. By default, all application servers are created with both secure and insecure transport chains. In order to ensure that a messaging engine can only be contacted using a secure transport chain, it is necessary to either disable or delete the insecure transport chains that are defined on the corresponding application server.

11.2.3 Data stores

A messaging engine uses a JDBC data source to access the data store with which it is associated. The process of adding an application server as a member of a service integration bus automatically creates a messaging engine on that application server. By default, a data store will also be created for this messaging engine, hosted within an embedded Cloudscape database. A JDBC data source for this database is also defined on the server that has been added to the bus. These defaults allow the messaging engine to run without any further configuration.

However, while adding a bus member, it is possible to specify the JNDI name of a different data source for use by the messaging engine. The sections that follow describe the issues that must be considered when deciding which RDBMS to use as a data store.

Data store location

The data store can be located on the same host as the messaging engine with which it is associated, or it can be located on a remote host. The decision of where to locate the data store might depend on the capabilities of the RDBMS that host the data store. For instance, the embedded Cloudscape database must run within the same application server process on which the messaging engine runs.

Note: Check with your database administrator to ensure that your RDBMS supports remote access from JDBC client applications.

The location chosen for the data store can have an impact on the overall performance, reliability or availability characteristics of the service integration bus components. For instance, a data store located on the same host as the messaging engine with which it is associated, can provide higher persistent message throughput by avoiding flowing data over the network to the data store. However, such a configuration might not provide the availability required, because failure of the host would mean that both the messaging engine and its data store would become unavailable.

Figure 11-19, shows the various options available when deciding where to locate a data store. The messaging engine in application server 1 uses the default Cloudscape data store, running in the same process as the application server. The messaging engine in application server 2 uses a data store hosted by a DB2 instance running on the same host as node 1. The messaging engine in application server 3 uses a data store hosted by a DB2 instance running on a remote host.

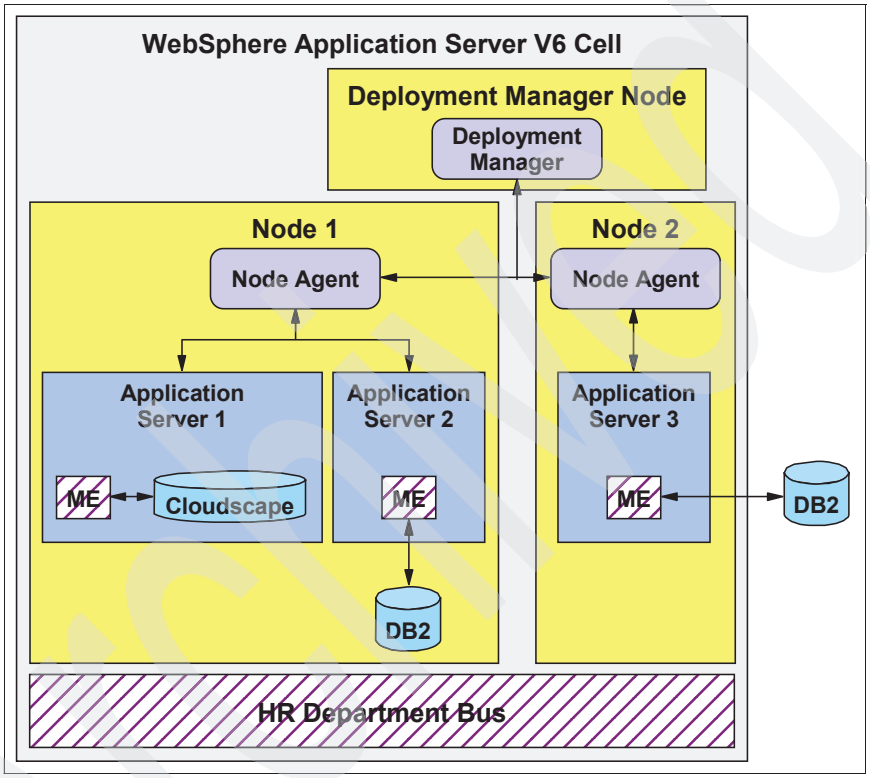


Figure 11-19 Data store locations relative to the associated messaging engine

Data store access

Each messaging engine must have exclusive access to the tables defined within its data store. This can be achieved, either by using a separate database as the data store for each messaging engine, or by partitioning a single, shared, database into multiple data stores using unique schema names for each data store.

Deciding which of these mechanisms to use depends on the capabilities of the RDBMS that will host the data store. For instance, the embedded Cloudscape database does not support concurrent access by multiple processes.

Note: Check with your database administrator to ensure that your RDBMS supports shared access from JDBC client applications and that it allows schema names to be specified on a JDBC connection. DB2 and Network Cloudscape support this functionality.

For databases that do not allow a schema name to be specified on a JDBC connection, multiple messaging engines share database access by each messaging engine using a different user ID when connecting to the database.

Figure 11-20 on page 623 shows the options available when deciding whether to use exclusive access or shared access to a data store. The messaging engine in application server 1 has exclusive access to the database hosting its data store. The messaging engines in application servers 2 and 3 have shared access to the database hosting their data stores. This shared database has been partitioned into separate schemas, with each messaging engine accessing the data store tables within a different schema.

Archive

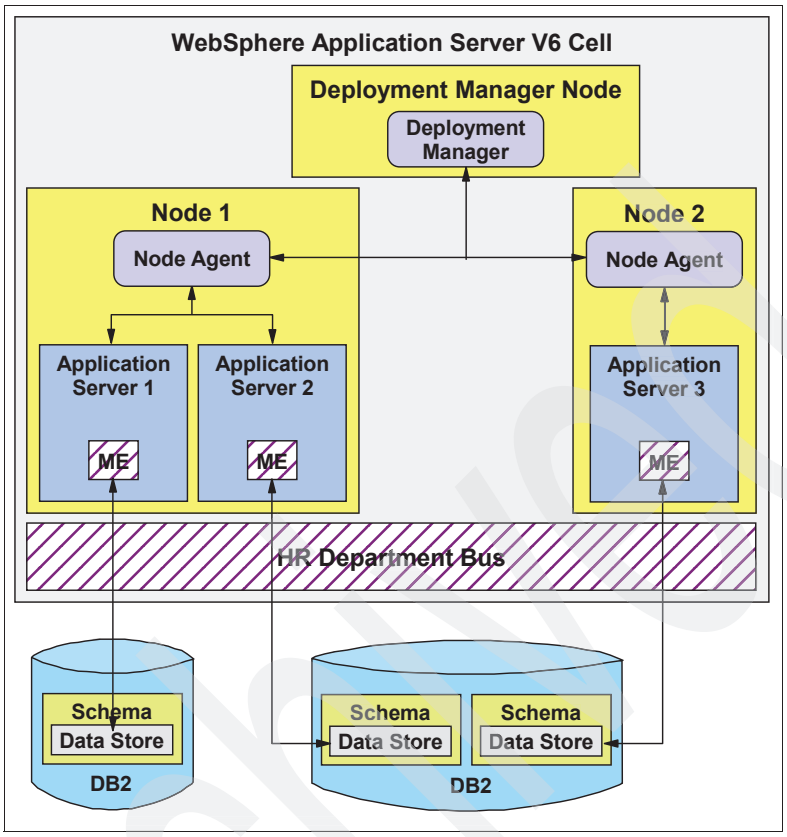


Figure 11-20 Exclusive and shared access to data stores

Data store tables

The messaging engine expects its data store to contain a set of specific tables, each of which has a specific table definition. Each messaging engine can be configured to create the tables within its data store, if they are not already present. During initialization, a messaging engine connects to its data store and checks for the required tables. If the messaging engine has the functionality to create tables, and they do not exist, it attempts to create the tables.

Some organizations allow a database administrator to perform only certain tasks on a database, such as creating tables. In this situation, the database administrator can use the **`sibDDLGenerator`** command to generate the DDL statements required to create these tables. The sibDDLGenerator command is located in the \bin\ subdirectory of the WebSphere installation directory. Refer to the WebSphere Information Center for a full description of the sibDDLGenerator command.

Note: In order for the messaging engine to be able to create the required tables within its data store, the user ID for the database must have sufficient privileges. Please refer to the WebSphere Information Center for a full description of the database privileges required in order for the messaging engine to access the data store.

Table 11-5 describes the tables defined within the data store for a messaging engine.

Table 11-5 Messaging engine data store tables

Table name	Description
SIBOWNER	Ensures exclusive access to the data store by an active messaging engine
SIBCLASSMAP	Catalogs the different object types in the data store
SIBLISTING	Catalogs the SIBnnn tables
SIBXACTS	Maintains the status of active two-phase commit transactions
SIBKEYS	Assigns unique identifiers to objects in the messaging engine
SIBnnn, where nnn is a number	Contains persisted objects such as messages and subscription information. These tables hold both persistent and nonpersistent objects, using separate tables for the different types of data, according to the following convention: ► SIB000 Use this name for the table which contains information about the structure of the data in the other two tables ► SIB001 Use this name for the table which contains persistent objects ► SIB002 Use this name for the table which contains non-persistent objects saved to the data store to reduce the messaging engine memory requirement

Archived

Note: When you remove a messaging engine, WebSphere Application Server V6 does not automatically delete the tables in its data store. To reuse this data store with another messaging engine, delete the tables within the data store manually.

Embedded Cloudscape considerations

As we discussed previously, each messaging engine defaults to using an embedded Cloudscape database to host its data store. However, when adding an application server cluster as a bus member, you do not have this default configuration. This is because the embedded Cloudscape database is not supported for cluster bus members. In certain failover scenarios, multiple processes might attempt to access the embedded Cloudscape database, and this is not supported.

11.2.4 Exception destinations

If a messaging client encounters a problem when attempting to consume a message from a bus destination, message delivery has failed. The message can be placed back on the bus destination for redelivery. Use the maximum failed deliveries property on a bus destination to determine the number of times a message can fail delivery. The default value of this property is five.

An *exception destination* handles undeliverable messages. Both queue and topic space destinations can define an exception destination. If a message cannot be delivered to its intended bus destination, it is rerouted to the specified exception destination. This mechanism prevents the loss of messages that cannot be delivered.

Note: Messages can also be placed on an exception destination for a variety of other reasons, examples of which include:

- When a destination is deleted, any messages on the destination are placed on the exception destination, unless the bus has been configured to discard them.
- When a message is received from a foreign bus, the message is placed on the exception destination if the target destination has reached its high message threshold.

Each messaging engine has a default exception destination of _SYSTEM.Exception.Destinaton.<ME_NAME>. By default, all bus destinations that have message points on a messaging engine use the default exception destination for that messaging engine when rerouting undeliverable messages.

This enables administrators to access all of the undeliverable messages for a messaging engine in one place.

However, an administrator can also configure a bus destination to use a nondefault exception destination. This enables administrators to access all of the undeliverable messages for a specific destination in one place, allowing for more fine-grained management of undeliverable messages.

When configuring a destination to use a non-default exception destination, the exception destination specified can be a local or a remote bus destination. It is also recommended that this destination is a queue destination and that it exists prior to the creation of the bus destination with which it is associated. If the exception destination specified has been deleted when a destination attempts to reroute an undeliverable message, the undeliverable message is rerouted to the default exception destination for the message engine.

Note: It is not possible to delete a default exception destination from a service integration bus. This ensures that there is always a default exception destination available on each messaging engine within the bus.

Note: Errors might occur as a message traverses the bus to its target destination. In this situation, the messaging engine handling the message attempts to redeliver the message. However, if the messaging engine determines that the target destination is unreachable, it can place the message on its default exception destination. For this reason, all exception destinations on the bus must be monitored to ensure that problem messages are processed appropriately.

When message order is important, it might be necessary to configure a bus destination not to use an exception destination. In this case, any messages that cannot be delivered to the target destination are not rerouted, and will be redelivered repeatedly. This has the effect of blocking the delivery of subsequent messages to the bus destination in question. For this reason, such a configuration should be used with caution.

Note: Publication messages arriving at a topic space destination for which there are no subscribers are not considered to be undeliverable. Such messages are discarded.

11.2.5 Service integration bus links

As discussed in section 11.1.7, “Foreign buses” on page 606, defining a foreign bus on a service integration bus simply defines a link between the two buses at an architectural level. When the foreign bus in question represents another service integration bus, the link is implemented at runtime by establishing a connection between a messaging engine from each of the buses. This link is configured on a messaging engine by defining a service integration bus link. A *service integration bus link* encapsulates the information required to communicate with a specific messaging engine, within a specific foreign bus.

When configuring a service integration bus link, it must be associated with the target foreign bus definition. The foreign bus definition with which it is associated enables the service integration bus link to determine the name of the target service integration bus. This is shown in Figure 11-21.

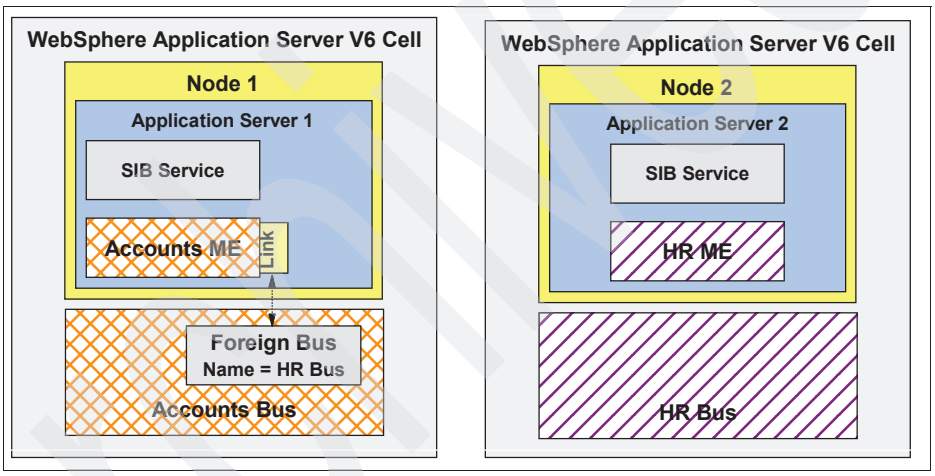


Figure 11-21 Association between a service integration bus link and a foreign bus

This requirement also determines the order in which these objects must be defined. The foreign bus must be defined within a service integration bus before a corresponding service integration bus link can be configured on a messaging engine.

Note: The name specified for the foreign bus must exactly match the real name of the target service integration bus.

The names of each of the buses involved in the link must also be unique. For this reason, if two service integration buses within separate WebSphere Application Server V6 cells need to be linked, care must be taken when naming each of the buses.

When attempting to establish the connection, the messaging engine within the local bus always attempt to connect to the foreign bus as though it were a remote client, even if the foreign bus is defined within the same WebSphere Application Server V6 cell. For this reason, a list of provider endpoints must also be specified when configuring the service integration bus link. These provider endpoints are used by the messaging engine in the local bus to connect to a bootstrap server in the foreign bus. For more information about the bootstrap process, refer to section 10.7, “Connecting to a service integration bus” on page 576.

The service integration bus link is also required to specify the name of the messaging engine on the target bus with which to connect. The messaging engine in the local bus uses the bootstrap server to locate the target messaging engine in the foreign bus. Figure 11-22 shows this process.

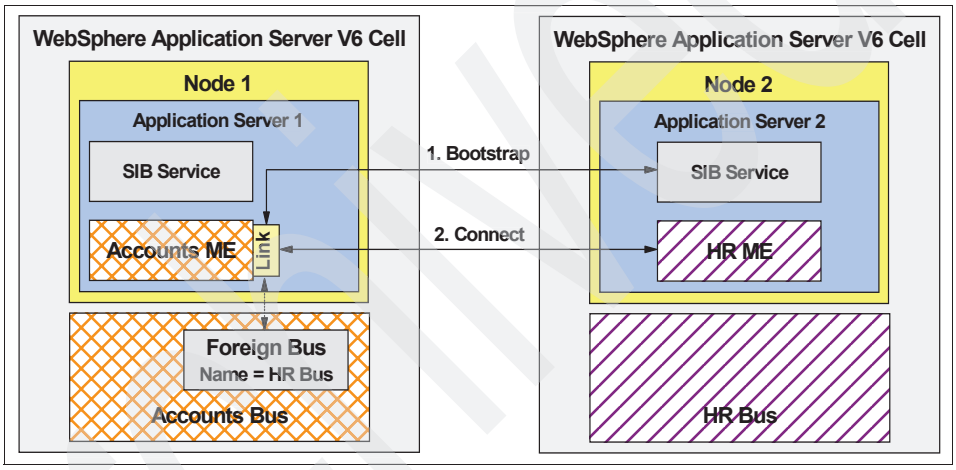


Figure 11-22 Bootstrapping during service integration bus link initialization

Once again, this requirement imposes an order in which the various configuration tasks must be performed. Each of the buses involved in the link must have at least one bus member defined before a service integration bus link can be configured.

The final requirement when configuring a service integration bus link, is that the link must be configured in both directions in order for the two buses to communicate at runtime. This is shown in Figure 11-23 on page 629.

Note: The name specified for the service integration bus link within both buses must be the same.

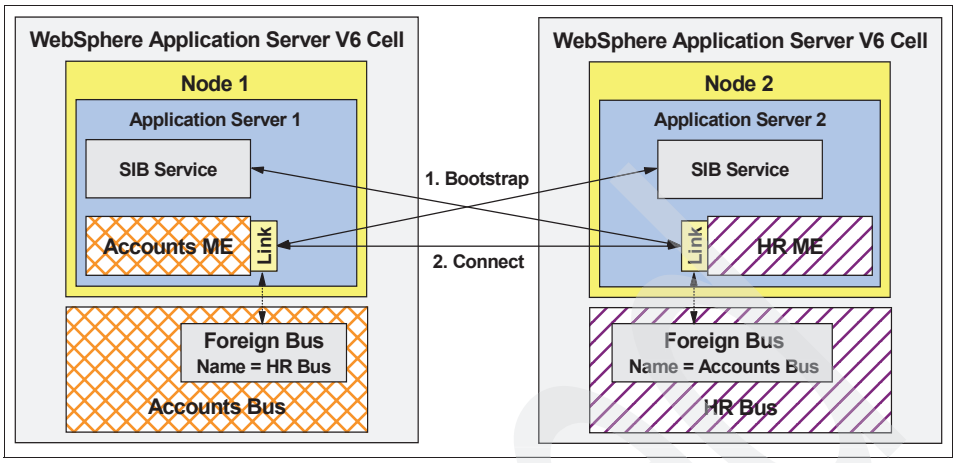


Figure 11-23 Defining a service integration bus link in both directions

Note: If the transport chain used by the service integration bus link encrypts its traffic using SSL, the names of the target inbound transport chain on each link must be the same. The transport chain specified must also be configured identically on each bus to ensure that compatible SSL credentials are used when establishing the link.

Topic space mappings

By default, a service integration bus link only flows messages across the link that are addressed to a queue destination on the foreign bus. In order to flow publication messages across the service integration bus link, topic space mappings need to be configured on the foreign bus definition.

These mappings define the topic space destination within the local bus for which publication messages are passed over the link. They also define the topic space destination on the foreign bus to which these publication messages are addressed. Refer to the WebSphere Information Center for more information regarding the definition of topic space mappings.

11.2.6 WebSphere MQ links

As discussed in section 11.1.7, “Foreign buses” on page 606, defining a foreign bus on a service integration bus simply defines a link between the two buses at an architectural level. When the foreign bus in question represents a WebSphere MQ network, the link is implemented at runtime by establishing sender and receiver channels between a specific messaging engine and a WebSphere MQ

queue manager. These channels are configured on a messaging engine by defining a *WebSphere MQ link*.

To a messaging engine configured with a WebSphere MQ link, the WebSphere MQ queue manager appears to be a foreign bus. To the WebSphere MQ queue manager, the messaging engine appears to be another WebSphere MQ queue manager. When configuring a WebSphere MQ link, an administrator must specify a virtual queue manager name. This is the queue manager name by which the messaging engine will be known to the remote WebSphere MQ queue manager. The WebSphere MQ queue manager is completely unaware that it is communicating with a messaging engine.

When you configure a WebSphere MQ link, you must be associate it with the target foreign bus definition. The name specified for the foreign bus does not need to match the name of the target WebSphere MQ queue manager. However, specifying a name for the foreign bus that matches the target WebSphere MQ queue manager, simplifies the routing of messages across the link.

Figure 11-24 shows a high level view of a WebSphere MQ link. Notice that the name of the foreign bus with which the WebSphere MQ link is associated, matches the name of the target WebSphere MQ queue manager.

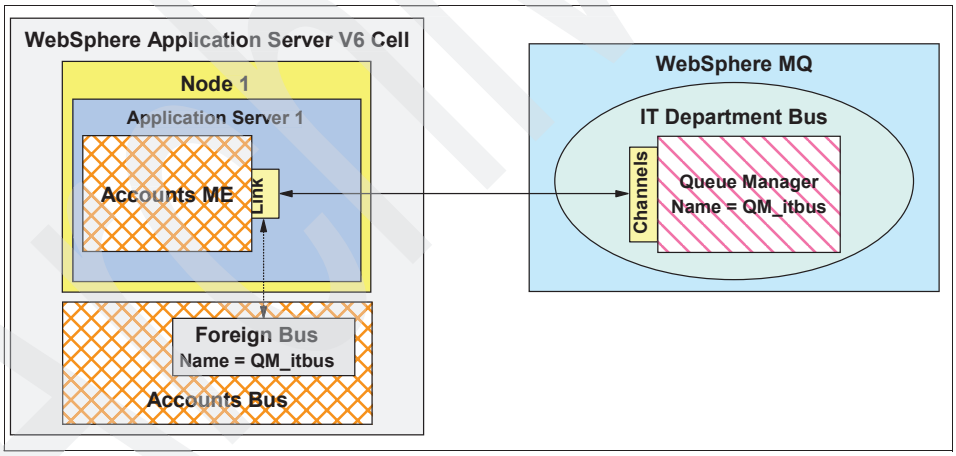


Figure 11-24 Overview of a WebSphere MQ link

WebSphere MQ link sender channel

The WebSphere MQ link sender channel establishes a connection to a receiver channel on the target WebSphere MQ queue manager. It converts messages from the format used within the service integration bus, to the format used by WebSphere MQ, and then sends these messages to the receiver channel on the target WebSphere MQ queue manager. For a full description of how messages

are converted as they traverse the WebSphere MQ link, refer to the WebSphere Information Center. The WebSphere MQ link sender channel emulates the behavior of a sender channel in WebSphere MQ. This is shown in Figure 11-25.

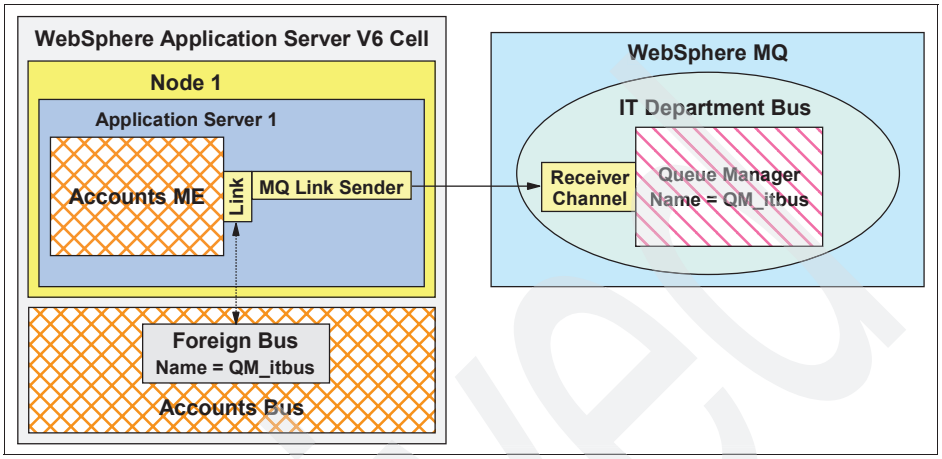


Figure 11-25 WebSphere MQ link sender channel

Note: It is only necessary to define a WebSphere MQ link sender channel if messages are required to be sent from the service integration bus to the WebSphere MQ network.

When you configure a WebSphere MQ link sender channel, you are required to specify the following information:

- A name for the channel, which must exactly match, including case, the name of the receiver channel defined on the target WebSphere MQ queue manager.
- The host name or IP address of the machine hosting the target WebSphere MQ queue manager
- The port number on which the target WebSphere MQ queue manager is listening for inbound communication requests
- An outbound transport chain

Note: If the receiver channel on the target WebSphere MQ queue manager accepts only SSL connections, you must associate the transport chain with a suitably compatible set of SSL credentials.

WebSphere MQ link receiver channel

The WebSphere MQ link receiver channel allows a sender channel within a WebSphere MQ queue manager to establish a connection to a messaging engine within the service integration bus. It converts messages from the format used within WebSphere MQ, to the format used by the service integration bus. For a full description of how messages are converted as they traverse the WebSphere MQ link, refer to the WebSphere Information Center. The WebSphere MQ link receiver channel emulates the behavior of a receiver channel in WebSphere MQ. This is shown in Figure 11-26.

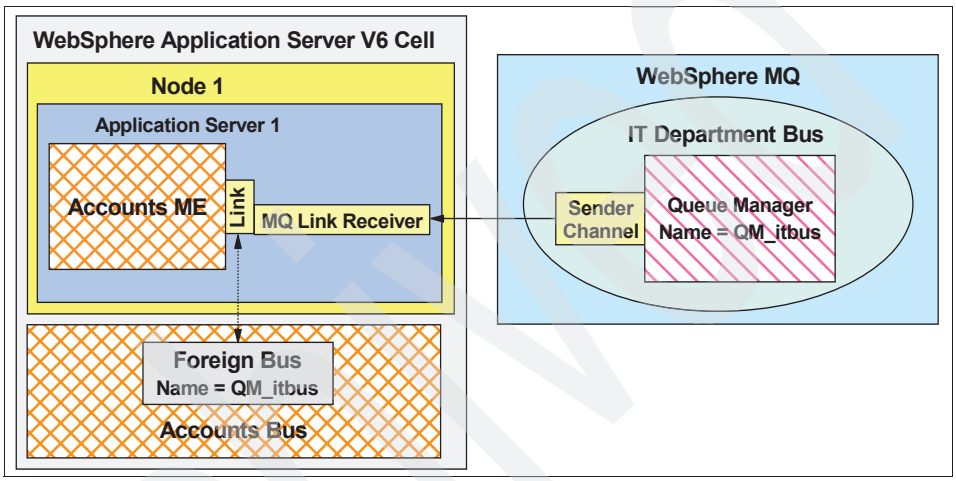


Figure 11-26 WebSphere MQ link receiver channel

Note: It is only necessary to define a WebSphere MQ link receiver channel if messages are required to be sent from the WebSphere MQ network to the service integration bus.

When configuring a WebSphere MQ link receiver channel, the following information is required:

- A Name for the channel, which must exactly match, including case, the name of the sender channel defined on the target WebSphere MQ queue manager

The inbound transport chain with which the sender channel on the WebSphere MQ queue manager communicates, is dependent on the configuration of the WebSphere MQ sender channel. The WebSphere MQ administrator should be consulted to ensure that the sender channel is configured appropriately. As discussed in “Inbound transport chains” on page 614, the InboundBasicMQLink

transport chain defaults to listening on port 5558 for connections from WebSphere MQ, and the InboundSecureMQLink transport chain defaults to listening on port 5578 for connections from WebSphere MQ.

MQ Publish/Subscribe broker profile

By default, a WebSphere MQ link only flows messages across the link that are addressed to a queue destination on the WebSphere MQ network. To flow publication messages across the WebSphere MQ link, configure a *publish/subscribe broker profile* for the WebSphere MQ link. A Publish/Subscribe broker profile allows *topic mappings* to be defined. These topic mappings define the topic names for which publication messages will be flowed across the WebSphere MQ link. Please refer to the WebSphere Information Center for more information about the definition of topic mappings within a publish/subscribe broker profile.

Addressing destinations across the WebSphere MQ link

There are several issues that must be considered when addressing a message to a destination that will flow across a WebSphere MQ link. These issues exist because of the differences in naming structure between the service integration bus and WebSphere MQ.

WebSphere MQ has a two-level addressing structure, as follows:

- Queue manager name
- Queue name

Each of these elements within WebSphere MQ is limited in length to 48 characters. Within the service integration bus, a destination can be uniquely identified using the following elements:

- Service integration bus name
- Destination name

The service integration bus places no length restrictions on these elements.

The difference in the allowable lengths of the various naming elements causes problems when a messaging application running in one environment attempts to address a message to a destination defined in the other environment, across the WebSphere MQ link. These issues are discussed in the sections that follow.

WebSphere MQ to service integration bus addressing

Messages that are sent from a WebSphere MQ application to a bus destination which has a name greater than 48 characters in length, must have some means of using the shorter name used in WebSphere MQ to address the long name used in the service integration bus.

The service integration bus uses an alias destination to map between the shorter name used by WebSphere MQ, and the longer name used by the service integration bus. A WebSphere MQ client application can address a message to an alias destination within a service integration bus that is defined with a short name of less than 48 characters. The alias destination then maps this message onto the destination defined with a long name of greater than 48 characters.

Service integration bus to WebSphere MQ addressing

Another problem can happen when a messaging client is required to address a message to a queue defined on an arbitrary queue manager within the WebSphere MQ network. For example, when defining JMS destinations for use by JMS client applications, it is only possible to specify the name of the bus on which the target destination is defined, and the name of the destination. If the destination exists within the WebSphere MQ network, the name of the foreign bus is specified as the bus name. However, if the target queue is not defined on the queue manager to which the WebSphere MQ link connects, additional information is required in order to address messages to the correct queue.

To solve this problem, when defining a JMS queue or an alias destination that represents a queue on a WebSphere MQ network, use a special format for the target queue name, of the form: <queue>@<queue manager>. These destination names are only parsed by the WebSphere MQ link, which uses the information to determine which values to place in the target queue and queue manager fields of the message header.

In the most simple case, the name specified for the foreign bus matches the name of the queue manager on which the target queue is defined. When this is the case, only the name of the target queue needs to be specified. If no queue manager name is applied as a suffix, then the foreign bus name will be added as the queue manager name by default. This is shown in Figure 11-27 on page 635.

Archive

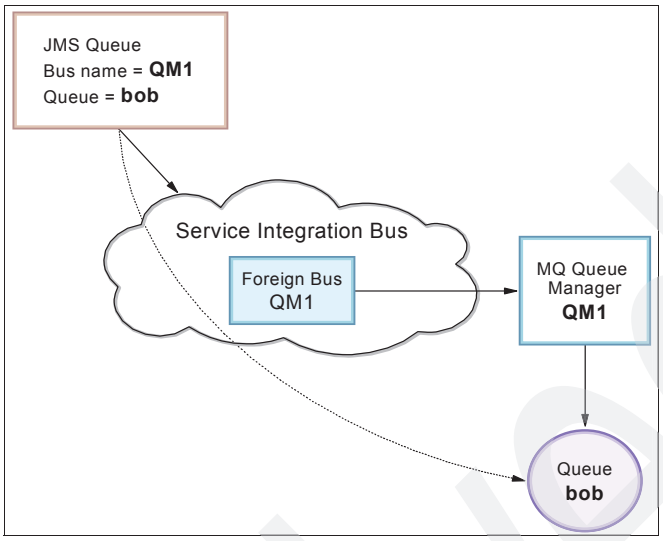


Figure 11-27 Simple WebSphere MQ addressing

This is still the case, even if the WebSphere MQ queue manager on which the target queue is defined, is not the same queue manager to which the WebSphere MQ link connects. This is shown in Figure 11-28.

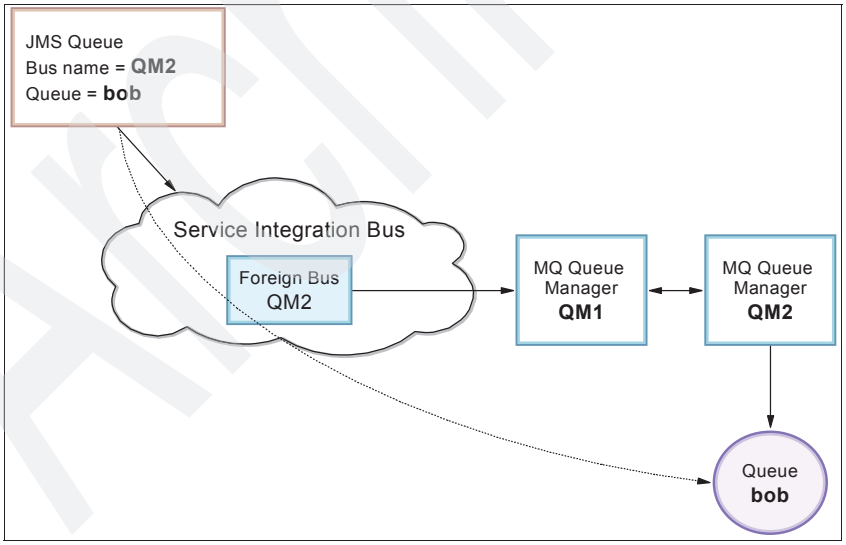


Figure 11-28 Simple WebSphere MQ addressing

When the name specified for the foreign bus does not match the name of the queue manager on which target queue is defined, the queue manager name

must be included as part of the queue name using the format described previously. This allows the message to be appropriately routed by WebSphere MQ once the message has left the service integration bus. This is shown in Figure 11-29.

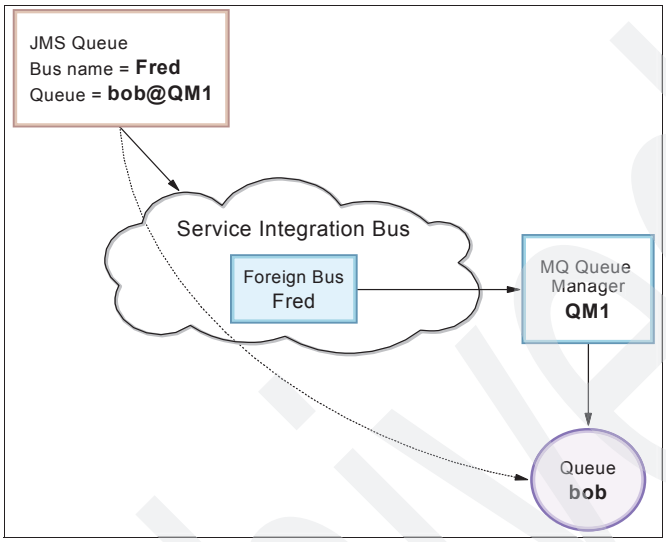


Figure 11-29 Advanced WebSphere MQ addressing

This mechanism enables a messaging client to address a message to a queue that is defined on any queue manager within the WebSphere MQ network. This is shown in Figure 11-30 on page 637.

Archived

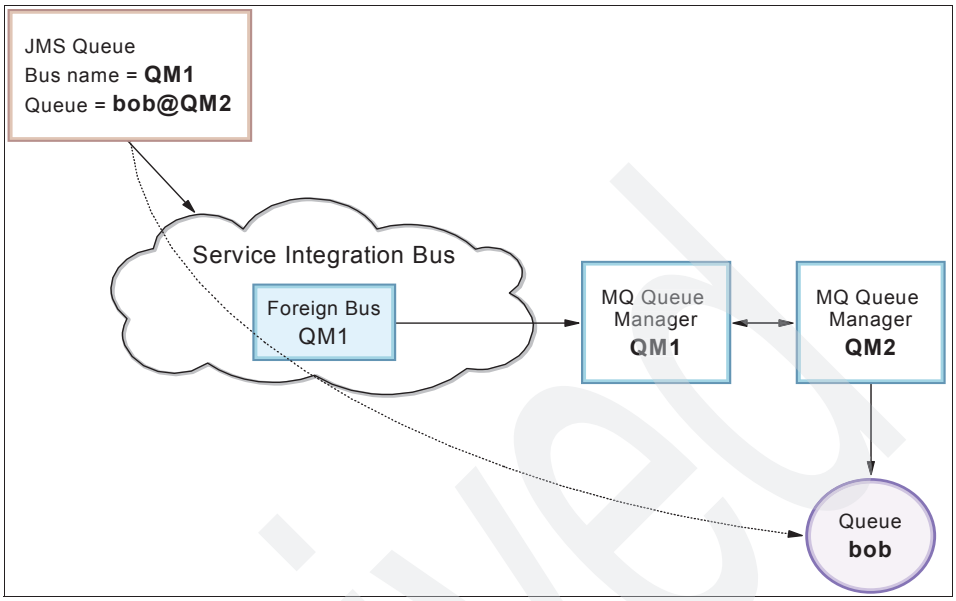


Figure 11-30 Advanced WebSphere MQ addressing

Note: The naming mechanism described within this section can only be used to address messages to destinations defined within WebSphere MQ. It must not be used to attempt to address messages to destinations defined on another service integration bus. An *indirect* foreign bus must be used for that purpose.

WebSphere MQ client links

A WebSphere MQ client link enables a messaging engine to act as a WebSphere Application Server V5.x embedded JMS Server. This function is provided as an aid to migration of V5.x to V6 and should not be used for any other purpose.

A WebSphere MQ client link enables any applications that are installed and configured on V5.x, using V5.x JMS resources, to continue to function as normal after the V5.x JMS server has been migrated to V6.

The process of migrating a V5.x node that contains an embedded JMS server will remove that JMS server and create a service integration bus with a WebSphere MQ client link. Queues previously defined on the V5.x embedded JMS server will be created automatically on the service integration bus.

See the Information Center topic *Migrating from version 5 embedded messaging* for more information.

You should not need to create a WebSphere MQ client link manually. Use the one created automatically for you by the migration process.

Important: It is recommended that you replace all v5.x JMS resources with v6.0 default messaging provider JMS resources as soon as possible. Once all resources have been changed, it is possible to delete the WebSphere MQ client link as all applications will be using the v6.0 default messaging provider directly.

11.3 High availability and workload management

Note: This section introduces you to the high availability and workload management capabilities when using the service integration bus. Before configuring your system, consult the following:

- *WebSphere Application Server V6 Scalability and Performance Handbook*, SG24-6392.
- *WebSphere Application Server V6: High Availability Solutions,* REDP-3971

High availability and workload management can be achieved using clusters as service integration bus members. It is worth noting however, that service integration bus messaging engines do not follow the same clustering model that J2EE applications do in clusters.

11.3.1 Cluster bus members for high availability

When You add a cluster to a service integration bus, a single messaging engine is created. The messaging engine is active on only one server within the cluster. In the event of an application server or messaging engine failure, the messaging engine becomes active on another server in the cluster if one is available.

By default, the messaging engine starts on the first available server in a cluster. If you want to ensure that the messaging engine runs on a particular server, for example if you have one primary server and one backup server, or if you want the messaging engine to only run on a small group of servers within the cluster, then you must specifically configure this. See 11.8.9, “Setting up preferred servers” on page 672 for details on configuring preferred servers.

11.3.2 Cluster bus members for workload management

Because a single messaging engine for the cluster is active, there is no workload management by default. To achieve greater throughput of messages, it is

beneficial to spread messaging load across multiple servers and, optionally, across multiple hosts. You can achieve this, while maintaining a simple destination model, by creating additional messaging engines for the cluster, each of which has a preference to run on a separate server in the cluster.

You can configure these messaging engines with a preference to run on particular servers within the cluster. This enables a messaging engine to run in every server in the cluster, thus providing every application in the cluster with a messaging engine for local access to the bus. Local access to the bus is always better for messaging performance, especially in the case of queues where the queue is assigned to the bus member from which it is being accessed.

When a queue is assigned to a cluster bus member the queue will be partitioned across all messaging engines in the cluster.

11.3.3 Partitioned queues

A queue is partitioned automatically for you when a queue destination is assigned to a cluster bus member. Every messaging engine within the cluster owns a partition of that queue and is responsible for managing messages assigned to the partition. Every message sent to the queue is assigned to exactly one of the partitions.

Local partitions

When a JMS client attempting to access a partitioned queue is connected to a messaging engine hosting one of those partitions (a messaging engine in the cluster), then the client is able to access only that local partition of the queue for both consuming and producing messages.

Note: The only instance where messages are not sent to the local partition is when that local partition is full and other partitions of the queue are not. In this case, messages are routed to an available remote partition.

Clients attempt to consume only from the local partition, even if there are no messages on the local partition and there are messages available on other partitions.

Remote partitions

If the JMS client connects to a messaging engine not hosting a destination partition, a messaging engine in the same bus but not in the cluster, then each client-created consumer connects to one remote partition to consume messages. Each session created is workload managed with respect to which remote partition it connects for consuming messages.

Messages sent to a remote partitioned destination are workload-managed across the individual partitions on an individual message basis, regardless of the session.

.

Important: Cluster bus members and partitioned queues alone do not give better message throughput. The applications producing and consuming the messages must be configured to use the service integration bus.

- Message producers must be configured to ensure that their messages will be workload-managed onto the different partitions of a partitioned queue. Following are examples of workload management:
  - Message producers, JMS clients, connect directly to the cluster. This has some restrictions in version 6.0. See 11.3.4, "JMS clients connecting into a cluster of messaging engines" on page 640. We anticipate removing these restrictions in the near future with a Fix Pack.
  - Message producers connect to messaging engines that are not part of the cluster. This requires servers outside of the cluster to be available and added to the bus, and for the message producers to make their JMS connections to those messaging engines. Once a messaging engine outside of the cluster accepts a message, the engine becomes responsible for routing the message through the bus to a queue point for the destination. Workload management selects a particular queue point of the partitioned destination so messages are spread evenly across all partitions of the queue.
  - An EJB or servlet in a cluster produces messages. Because the calls to the EJB or servlet are workload-managed across the cluster, and assuming that messages are produced to a local queue partition, it follows that the messages produced will be workload managed across the partitions of the queue.
- Message consumers must be configured to connect to each partition of a partitioned queue to consume messages. If any partitions do not have consumers, then the messages sent to that partition might never be consumed.

  The simplest, and recommended way of configuring consumers to every partiton of a partitioned queue is by installing a message-driven bean on the cluster.

11.3.4 JMS clients connecting into a cluster of messaging engines

JMS clients outside of a cluster can connect directly into a workload-managed cluster of messaging engines. In this case, *workload-managed* means the cluster

is a bus member and one messaging engine has been added for every server in the cluster. Each messaging engine has been configured to prefer a different server in the cluster. JMS clients connect to the messaging engines using the connection rules described in 10.7, “Connecting to a service integration bus” on page 576.

In this scenario, there is an undesirable side effect of the rules when the servers in the cluster are used as the provider endpoints for the connection factory. Consider the following example:

A JMS client connects into a cluster of servers A,B and C. The connection factory is configured with provider endpoints of A,B and C. This allows the client to bootstrap to any of the three servers in the cluster. Following the connection rules, the connection factory bootstraps to the first server in the provider endpoints list, A. Server A has a local messaging engine, therefore the messaging engine on Server A is chosen as the preferred connection point for the client.

Because the connection always tries the first entry in the provider endpoints list first, every client connecting directly into the cluster connects to the messaging engine in server A. All messages produced for a destination partitioned across the cluster are assigned to the partiton of the destination associated with the messaging engine. This is obviously not very good for workload management of messages. There are two methods that can overcome this.

- Enable a SIB service on a server outside of the cluster. Configure the provider endpoints on the connection factory to point to this SIB service. If there is no messaging engine local to this SIB service, then the client connections will be workload-managed around all of the messaging engines in the bus.

  If you only have messaging engines in the cluster, no further configuration is required. If there are other non-cluster bus members, and you only want the clients to connect directly to the messaging engines in the cluster, then you must configure a target group on your connection factory. See “Target groups” on page 585.
- Provide different clients with differently configured connection factories, each of which has a different provider endpoint in the first position in the list.

11.3.5 Preferred servers and core group policies

To configure a messaging engine to prefer a server or group of servers you must configure a core group policy. A core group policy is used to identify server components, and define how they will behave within a cell or cluster. This section discusses these components.

Policy type

For service integration bus messaging engines, use a policy type of `One of N`. This means that, while the messaging engine cam be defined on every server in the cluster, WebSphere's HA Manager ensures that it is only active on one of the servers in the group, and will always be active on one of the servers, if one is available.

Match criteria

The match criteria of a core group policy enables the HA Manager to decide what server components match the policy and so should be managed according to the policy. There are two match criteria that you must use to match a messaging engine:

- `type=WSAF_SIB`

  This criterion matches any messaging engine

- `WSAF_SIB_MESSAGING_ENGINE=<messaging_engine_name>`

  This criterion matches the messaging engine of the name provided.

Preferred servers

The preferred servers defined in a policy allow you to list a group of servers on which the messaging engine will prefer to run. The higher up in the list of preferred servers a particular server is, the more preferred it is. For a messaging engine that is part of a cluster bus member, select only preferred servers that are part of the cluster. The messaging engines are defined only in the cluster and cannot be run on any servers outside of the cluster.

Fail back and preferred servers only

These two options have a large effect on how a particular policy will make a messaging engine behave in a cluster.

If you select **fail back**, when a more preferred server becomes available then the messaging engine will be deactivated where it currently runs and activated on the more preferred server.

Enabling fail back ensures that a messaging engine always run on the most preferred server that is available. This is usually desirable as there should be a good reason for configuring a preferred server in the first place.

If you do not enable fail back, then once a messaging engine has started it will not move to a more preferred server if one becomes available.

If you select **preferred servers only,** then the messaging engine will only be allowed to be active on servers in the policy's preferred servers list. If you do no

select Preferred servers only, all servers in the cluster that are not in the list will be able to have the messaging engine active on them, but they will be selected only if none of the preferred servers are available.

Be very careful when selecting preferred servers only because it is possible to reduce or remove the high availability of a messaging engine and of the queue partitions that the messaging engine owns.

If none of the preferred servers are available, then the messaging engine will not be active anywhere. This means any queue partitions owned by that messaging engine will also be unavailable. Any messages currently on those partitions will be trapped and cannot be consumed until one of the preferred servers has become available and the messaging engine has been activated.

Large clusters

If you have a medium or large cluster of servers, five or more, configured with messaging engines, then we recommend a slightly special configuration of preferred servers.

With a large number of messaging engines defined on a cluster, it would be undesirable to have all of the messaging engines starting up on the first server in the cluster to start. We suggest the following configuration.

Configure each messaging engine with a group of preferred servers consisting of a subset of the cluster with fail back and preferred servers only enabled. The set of preferred servers should be large enough to support your availability requirements by providing sufficient failover capabilities for the messaging engine. For example, you might decide that the messaging engine must be able to run on two or three servers. Configure each messaging engine with a different subset of servers, with each messaging engine having a unique, most-preferred server, as in Figure 11-31 on page 644.

In Figure 11-31 on page 644, the shading indicates the preference order of the servers.

Archive

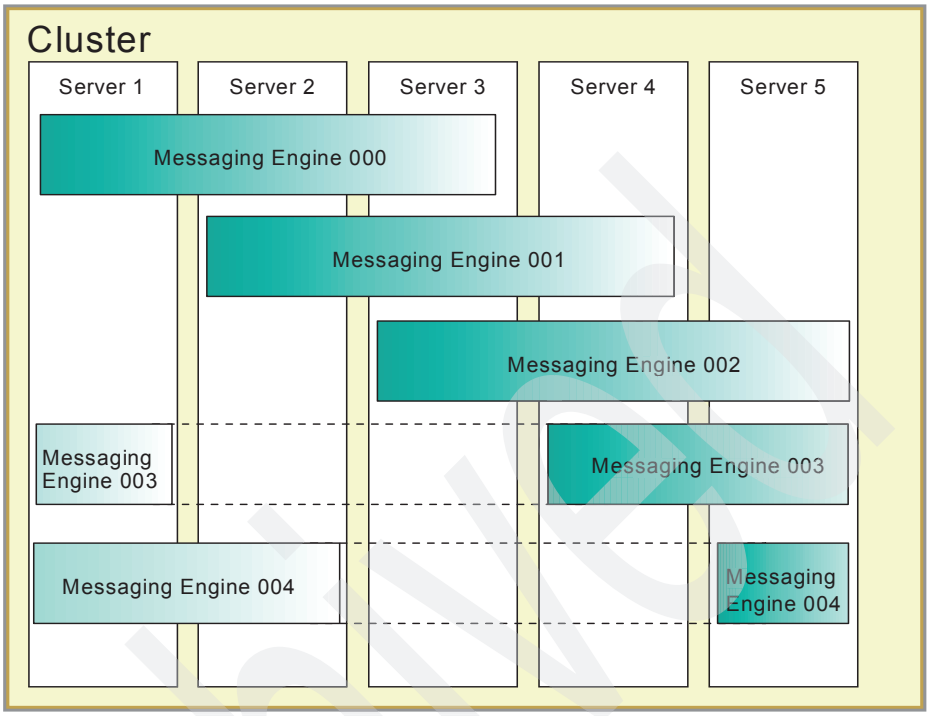


Figure 11-31 Configuring large clusters of messaging engines

11.3.6 Best practices

For the greatest throughput of messages, do the following:

1. Create a cluster bus member with messaging engines running on every server in the cluster.
2. Define the queue or queues being used on the cluster bus member.
3. Ensure that message production to the queue is workload-managed across the cluster:
   - Install an EJB or servlet application on the cluster and have that application produce the messages. Workload management of the client calls to the application workload manages the message production across the cluster.
   - Produce messages from clients connected to messaging engines outside of the cluster. The service integration bus can then workload manage the messages across the cluster.
4. Install an MDB application on the cluster to consume the queue messages.

Archive

11.4 Service integration bus topologies

This section discusses briefly some messaging topologies, working up from the simplest to more complex configurations.

11.4.1 One server in the cell is a member of one bus

In this topology, there is only one service integration bus. There might be multiple application servers in the cell, but only one is a member of the bus. This is roughly equivalent to the typical v5.x JMS server topology.

The pros of this topology are:

- It is very simple to set up and manage.
- It can be expanded later by adding more servers to the bus.

The cons are:

- Clients running on other application servers in the cell have to connect remotely to the bus rather than connecting locally. This can affect messaging performance.
- Clients running outside application servers have to connect to the bus member to do messaging. The connection factory you use needs to have provider endpoints configured with the details of the bus member server.

  If the SIB service is enabled on other application servers in the cell, then connection factories can be configured with provider endpoints that point to a list of bootstrap servers. See 10.7, “Connecting to a service integration bus” on page 576 for more information about using a bootstrap server and defining a list of provider endpoints.

  In either case, all messaging connections go to the bus member server and might affect messaging performance.
- Message consumers might not be on the same server as the queue points they are consuming from. This could have a performance impact.
- This topology cannot be upgraded easily to support high availability or workload management. High availability and workload management require clustering application servers. You can create a new cluster and include the bus member as the first application server in the cluster. However, this will not automatically give you the messaging high availability features that are normally associated with adding a cluster as a bus member.
  - Using the bus member server as the template for a cluster server is *not* equivalent to adding a cluster to the bus. No bus information is copied as part of the template process. The SIB service will be enabled on the new cluster server as a server property, not part of any particular bus.

Archived

– Using the bus member as the first server in cluster server is *not* equivalent to adding a cluster to the bus. Only the original server is part of the bus.

It is possible to add a cluster to the bus, delete all of the queues you want to be highly available or workload-managed and recreate queues of the same name that have their queue points located on the new cluster bus member. Any messages on the queues are lost when they are deleted.

11.4.2 Every server in the cell is a member of the same bus

In this topology there are multiple application servers, but no clusters. There is one service integration bus and each application server is a member of that bus.

The pros of this topology are:

- Clients in application servers can connect locally to the bus, improving performance. If only some servers in the cell are members of the bus, then install any messaging applications on those servers.
- Clients running outside application servers can connect to any cell server to perform messaging, providing some degree of high availability for those clients.
- It is possible to have queue points in the same servers as applications that consume messages from them, improving performance.

The cons are:

- This topology is not easily upgradable to support high availability or workload management. See the list of cons in 11.4.1, “One server in the cell is a member of one bus” on page 645.

11.4.3 A single cluster bus member and one messaging engine

This scenario assumes that all the application servers in the cell belong to one cluster and that cluster is a member of the service integration bus.

The benefit of this scenario is:

- The messaging engine is highly available. If the messaging engine or the server on which it runs fails, then the messaging engine starts up on another available server in the cluster. See “High availability and workload management” on page 638 for more details.

Archived

The drawback of this scenario is:

- If you want to ensure that the messaging engine runs on one preferred server, for example, if you have one primary server and one backup server, then you must specifically configure this. See “Setting up preferred servers” on page 672.

> **Note:** Be aware that some configurations of preferred servers for a messaging engine can make that messaging engine not highly available.
>
> If preferred servers are set up for the messaging engine with the preferred servers only option, then it is possible, if none of the preferred servers is available, that the messaging engine will not have another server on which to start even if other servers are available in the cluster. See “High availability and workload management” on page 638.

11.4.4 A cluster bus member with multiple messaging engines

This scenario assumes that all the application servers in the cell belong to one cluster. Multiple messaging engines have been defined for the cluster.

The pros of this topology are:

- The messaging engines in the cluster are highly available.
- The cluster bus member is capable of messaging workload management. A queue point assigned to the cluster bus member is partitioned onto every messaging engine in the cluster and messages delivered into the cluster are distributed between the partitions. See “High availability and workload management” on page 638 for more details.

The drawback is:

- There are some restrictions on the workload management of client connections directly into a cluster. See 11.3.4, “JMS clients connecting into a cluster of messaging engines” on page 640 for details.

11.4.5 Mixture of cluster and server bus members

The cell has some application server clusters and other non-clustered servers. Both non-clustered servers and server clusters have been added as bus members. Complex configurations such as these can be completely tailored to best suit your application and server topologies.

The benefits of this topology are:

- Cluster bus members can be configured with partitioned destinations to support workload-managed, message-consuming applications such as message-driven beans.
- Cluster bus members can be used to make system-critical destinations highly available.
- To overcome the workload management restrictions of clients connecting to a cluster, clients outside the cell can connect to server bus members. Clients can then put messages to destinations with partitioned queue points. Messages are workload-managed between the partitions.
- To overcome the workload management restrictions of clients connecting to a cluster (see 11.3.4, “JMS clients connecting into a cluster of messaging engines” on page 640), clients outside the cell can connect to server bus members outside of the cluster. Clients can then put messages to partitioned destinations and the messages will be workload-managed across the partitions.
- Clients bootstrapping to servers (with a SIB service) outside the cluster can get workload management of their connections to the messaging engines within the cluster bus member.

The drawback is:

- These more complex messaging topologies take a little more planning and configuration than simpler topologies.

11.4.6 Multiple buses in a cell

It is possible to have many service integration buses within a cell. This topology can be desirable under in the following situations:

- Separation of concerns

  Applications that do not need to share messages can be isolated from each other by using their own bus.

- Test configuration

  A test configuration with identical destination names can be created on separate bus that is not used by the production system. Changing the name of the bus in the connection factories can then redirect the test application to the production bus without changing any other configuration.

Archive

11.5 Service integration bus and message-driven beans

Message-driven beans (MDBs) attached to destinations in the service integration bus are attached by means of the SIB JMS Resource Adapter, an activation specification and a JMS destination. The resource adapter is responsible for connecting to the service integration bus and delivering messages to the MDB.

Note: For performance reasons, we recommend that MDBs are always installed on a server that has an active local messaging engine and a queue point on that local messaging engine.

11.5.1 Message-driven beans connecting to the bus

The resource adapter always attempts to connect a message-driven bean to a messaging engine in the same server, if one is defined there. If there is no messaging engine in the same server, then a messaging engine is selected from the bus using the standard connection selection process, see 10.7, “Connecting to a service integration bus” on page 576.

There are three scenarios where an MDB will start but not connect to the destination for which it is configured to listen. The resource adapter will allow the MDB application to start under these circumstances and will attempt to connect the MDB to its configured destination as soon as possible.

Local messaging engine defined but unavailable

If a messaging engine is defined locally, but is unavailable when the MDB application starts, the MDB application starts sucessfully and the resource adapter connects it to the messaging engine when it activates. Situations when this happens include:

- If the messaging engine has not started by the time the MDB application is started
- The MDB is installed on a cluster bus member which has been configured for high availability, and is on a server other than the one with the active messaging engine

When an MDB application is started but the locally defined messaging engine is unavailable the following warning message will appear in SystemOut.log:

Example 11-1 Message: local messaging engine not available

```
CWSIV0759W: During activation of a message-driven bean, no suitable active
messaging engines were found in the local server on the bus MyBus
```

When the messaging engine activates, the message in Example 11-2 is displayed when the MDB is connected to its destination:

Example 11-2 MDB connected to messaging engine

```
CWSIV0764I: A consumer has been created for a message-driven bean against
destination MyQueue on bus MyBus following the activation of messaging engine
cluster1.000-MyBus.
```

Note: Messaging engines are frequently the last component of an application server to complete their startup, often even after the open for e-business message is issued for the server. As a result, it is not unusual for MDB applications to cause the above warning message.

Remote destination unavailable

If there is an active locally defined messaging engine, but the MDB is configured to listen to a queue currently unavailable (for example if the messaging engine that hosts the queue point is not active), then the warning message Example 11-3 is displayed:

Example 11-3 Message: remote destination unavailable

```
CWSIV0769W: The creation of a consumer for remote destination MyQueue on bus
MyBus for endpoint activation ...<section removed>... failed with exception
javax.resource.ResourceException: CWSIP0517E: Cannot attach to queue message
point for destination MyQueue.
```

The resource adapter tries to connect the MDB to the configured destination every 30 seconds until it succeeds. Each failure to connect results in the message in Example 11-3.

Remote messaging engine unavailable

If there is no locally defined messaging engine, then a messaging engine is selected from the bus. If there are no currently available messaging engines in the bus, then the resource adapter allows the MDB application to start anyway and attempt to connect the MDB to a messaging engine every 30 seconds. The message in Example 11-4 appears on the first failed attempt to connect to a messaging engine. Subsequent failures are silent:

Example 11-4 Message: remote messaging engine unavailable

```
CWSIV0775W: The creation of a connection for destination MyQueue on bus MyBus
for endpoint activation ...<section removed>... failed with exception
com.ibm.websphere.sib.exception.SIResourceException: CWSIT0019E: No suitable
messaging engine is available in bus MyBus.
```

No messages are delivered to the MDB until the resource adapter has been able to start a connection to an active messaging engine. The message in Example 11-5 is displayed with a connection is made:

Example 11-5 Message: connection made to remote messaging engine

```
CWSIV0777I: A connection to remote messaging engine myNode.server1-MyBus for
destination MyQueue on bus MyBus for endpoint activation ...<section
removed>... is successfully created.
```

11.5.2 MDBs and clusters

The behavior of message-driven beans installed on clusters that use the service integration bus is directly related to the service integration bus configuration.

Clusters that are not part of a bus

When an MDB is installed on a cluster that is not part of a bus, the MDBs on each server connect independently to the bus to consume messages.

Note: You should not configure an MDB on a cluster with no local messaging engine to listen to a partitioned queue in another cluster. There is no guarantee that every partition of the queue in the other cluster will have at least one MDB listening to it. This could lead to a partition without any consumers.

Clusters configured for highly available messaging

When a cluster is configured for highly available messaging, a messaging engine is active on one of the servers in the cluster. An MDB application installed on that cluster will start on all servers in the cluster, but only the MDB on the server with the active messaging engine will receive messages. Should the active messaging engine fail, or the server on which it is active fail or be stopped, then the messaging engine will start on another server in the cluster. The MDB on that server will be connected to the messaging engine and start receiving messages.

In this scenario, the service integration bus has been configured to have one active messaging engine in the cluster, and, effectively, the MDB mirrors that configuration.

Clusters configured for messaging workload management

When a cluster is configured for messaging workload management, a messaging engine will most likely be active on each server in the cluster.

For a MDB installed on the cluster and listening to a topic with a non-durable subscription, each message on the topic will be received once on each server with an active messaging engine. If more than one messaging engine be active on a server, a publish topic message will still be received only once by the MDB on that server.

If the MDB installed on the cluster is listening to a topic with a shared, durable subscription, then one MDB in the cluster receives each message published on the topic only once.

If the MDB installed on the cluster is listening to a queue partitioned on the cluster, then the MDB is attached to each partition active on the server. Should more than one messaging engine be active on a server then the MDB will receive messages from each messaging engine's partition of the queue.

For a MDB installed on the cluster and listening to a queue with its queue point on a messaging engine outside of the cluster, the MDB on each server is attached to the queue. An MDB on a server with more than one active messaging engine will not receive a greater proportion of the messages than an MDB on a server with only a single active messaging engine.

11.6 Service integration bus security

When security is enabled on WebSphere Application Server V6, certain steps must be taken for JMS applications using the service integration bus to authenticate themselves to the bus, allowing them to continue to use the messaging resources.

- All JMS connection factory connections must be authenticated. This can be done in two ways:
  - The connection factory can have a valid authentication alias defined on it.
  - The JMS application can pass a valid user name and password on the call to ConnectionFactory.createConnection(). An ID passed in this way overrides any ID specified in an authentication alias on the connection factory.
- All activation specifications must have a valid authentication alias defined on them.

Archived

Note: If a connection factory is looked up in the server JNDI from outside of the server environment, as from the client container, any authentication alias defined on the connection factory will be unavailable. This prevents unauthorized use of an authenticated connection factory.

JMS clients outside of the server can provide user name and password on the call to create connection. If the client is a J2EE client application running in the WebSphere application client environment, it is possible to define an authenticated connection factory resource in the .ear file.

Any user that authenticates as a valid user to WebSphere has, by default, full access to the bus and all destinations on it. It is possible to configure destination and even topic-specific authentication requirement, if you want.

Every bus has an optional inter-engine authentication alias which can be specified. If this property is left unset, then it will be ignored. However, if an alias is specified and security enabled, then the ID will be checked when each messaging engine starts communications to other messaging engines in the bus. This provides additional security to prevent hackers pretending to be another messaging engine in the bus.

Details on WebSphere security can be found in *WebSphere Application Security V6 Security Handbook,* SG24-6316

11.7 Problem determination

The following information is presented to help you become familiar with successful messaging engine startup, and some common problems.

No problems

Example 11-6 shows an example of what you can expect to see in systemOut.log on server start up for a messaging engine that starts sucessfully:

Example 11-6 Successful messaging engine start

```
...
CWSID0016I: Messaging engine Node1.server1-ITSOBus is in state Joined.
...
CWSID0016I: Messaging engine Node1.server1-ITSOBus is in state Starting.
...
CWSID0016I: Messaging engine Node1.server1-ITSOBus is in state Started.
...
```

Note: When you start a server that is part of a cluster bus member, then the messaging engine will not always be started. Only one server in the cluster will have a specific messaging engine activated on it and this messaging engine might already be started.

If this is the case, then you will see the messaging engine in the state `Joined`, but not `Starting` or `Started`. This is perfectly normal and means that the messaging engine is in a stand-by state, waiting to be activated should the currently active instance of the messaging engine become unavailable.

When you have more than one messaging engine in a bus, you will also see the messaging engines communicate with each other. Every messaging engine in the bus connects to every other messaging engine in the bus, as shown in Example 11-7.

Example 11-7 Messaging engine connections

```
...
CWSITO028I: The connection for messaging engine Node1.server1-ITSOBus in bus
ITSOBus to messaging engine Node2.server2-ITSOBus started.
...
CWSIPO382I: messaging engine B68588EF698F4527 responded to subscription
request, Publish Subscribe topology now consistent.
...
```

CWSIS1535E: Messaging engine's unique id does not match...

If you see the error shown in Example 11-8, the database that the messaging engine points to contains the unique ID of a different messaging engine. The most likely cause of this is when you create a bus, add a server to that bus using the default Cloudscape database and start the server. Later, you delete and recreate a bus of the same name. The newly created messaging engine will use a default data source that points to the same database used by the old messaging engine, and this database will contain the ID of the old messaging engine.

This error can also be caused by configuring any messaging engine with the same data store as another messaging engine.

Example 11-8 Messaging engine unique id doesn't match

```
CWSIS9999E: Attempting to obtain an exclusive lock on the data store.
CWSIS1535E: The messaging engine's unique id does not match that found in the
data store. ME_UUID=1C8O283E64EAB2CA, ME_UUID(DB)=B1C4OF1182BOAO45
WSIS1519E: Messaging engine Node1.server1-ITSOBus cannot obtain the lock on its
data store, which ensures it has exclusive access to the data.
```

```
CWSID0027I: Messaging engine Node1.server1-ITSOBus cannot be restarted because
a serious error has been reported.
CWSID0016I: Messaging engine Node1.server1-ITSOBus is in state Stopped.
```

The simplest solution is to drop the tables in the database, or delete and recreate the database then restart the server. Another solution is to change the messaging engine's data store by changing the schema, user, and database configured for the messaging engine. See , "Adding the bus member" on page 664 for more details.

CWSIT0019E: No suitable Messaging Engine...

This exception shown in Example 11-9 can be thrown to a JMS client on a createConnection call. Causes of this exception include:

- The JMS connection factory cannot contact an SIB service, for out of cell JMS clients only. Check that the provider endpoints listed in the connection factory match the host and port for the SIB services on the servers. Ensure that the SIB services are enabled and the servers are started.
- The bus name defined in the JMS connection factory does not match the name of a bus defined in WebSphere.
- No messaging engines on the named bus are active.

Example 11-9 Exception on createConnection call

```
javax.jms.JMSException: CWSIA0241E: An exception was received during the call
to the method JmsManagedConnectionFactoryImpl.createConnection:
com.ibm.websphere.sib.exception.SIResourceException: CWSIT0019E: No suitable
messaging engine is available in bus ITSOBus.
```

11.8 Configuration and management

This section discusses how to set up and configure a service integration bus using the administrative console.

Note: In the following instructions, we frequently suggest saving the changes. You do not have to do this and can make several changes before saving.

11.8.1 SIB service configuration

SIB service is an application server service enabling the server for service integration activities. When a server is added to a bus, it automatically has its SIB service enabled. Having the SIB service allows an application server to have

active messaging engines and to be used as a provider endpoint for default messaging connection factories. The port on which the SIB service listens can be looked up on the servers configuration panel.

1. Select **Servers** → **Application Servers.**
2. Select any server.
3. Under **Communications**, expand the **Ports** heading. `SIB_ENDPOINT_ADDRESS` is the port used by SIB Service for that server. See Figure 11-32.

Note: SIB service listens on a number of ports, not just the port for `SIB_ENDPOINT_ADDRESS`. `SIB_ENDPOINT_SECURE_ADDRESS` is also available, and is used for secure communications. Tunnelled and tunnelled secure endpoints are also provided: `jfap/http/tcp` and `jfap/http/ssl/tcp`. Refer to the Information Center for more details.

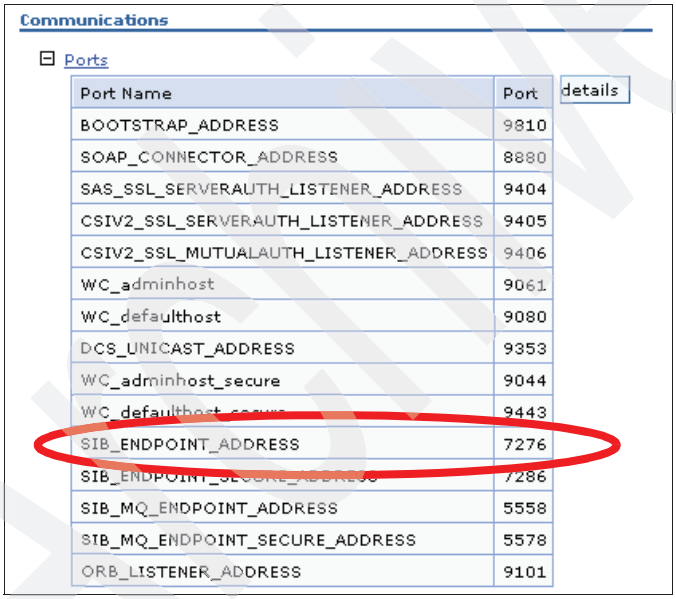


Figure 11-32 Port numbers used by a server

The settings for the SIB service of an application server can be found on the administrative console:

1. Select **Servers** → **Application Servers.**
2. Select any server.
3. Under **Server messaging** select **SIB service.** See Figure 11-33 on page 657.

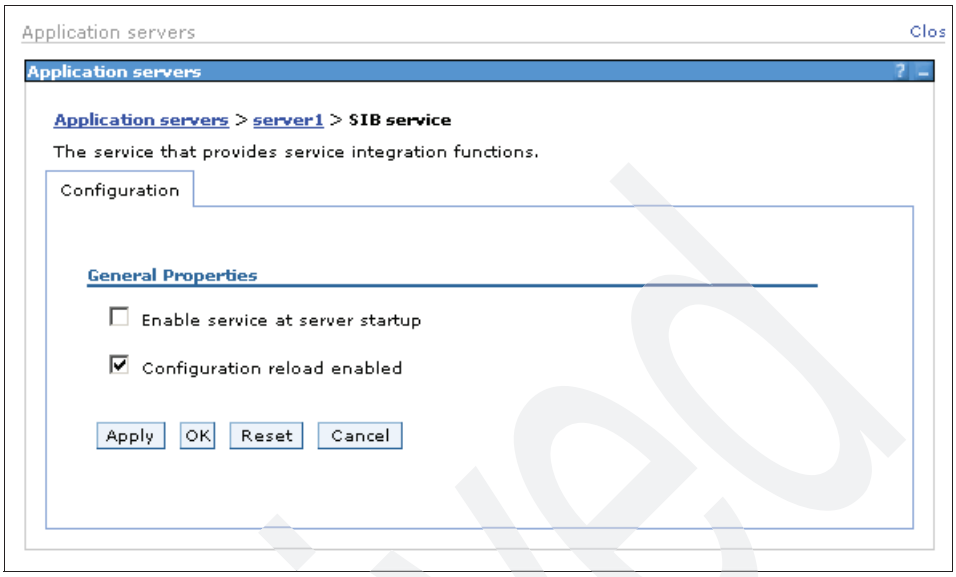


Figure 11-33 SIB Service panel

The panel for SIB service has two options.

– **Enable service at server startup**

  This option is not enabled on a server by default. However, it is automatically enabled if you add a server to a bus. If you disable the SIB service, then any messaging engines defined on the server will not be started.

– **Configuration reload enabled**

  This option allows the SIB service to activate dynamically select changes to a bus configuration during runtime. Creation, deletion or modification of a destination or mediation takes effect almost immediately on a running system. If a new destination is created, it becomes available for use without having to restart servers or messaging engines. Some configuration changes do require the affected server or messaging engine to be restarted before the changes become effective, such as the creation of a new bus, messaging engine, foreign bus link or MQ link.

  A matching flag must also be enabled on each bus on which you want to enable configuration reload. This flag is enabled by default on every bus, but can be disabled if you want. To modify the flag either way, do the following:

  i. Select **Service integration** → **Buses.**
  ii. Select a bus.

Archived

iii. Modify the **Configuration reload enabled** flag as appropriate.
iv. Save the changes.

11.8.2 Creating a bus

No service integration buses are defined by default. To create a bus, do the following:

1. Select **Service integration** → **Buses.**
2. Click **New.** See Figure 11-34.

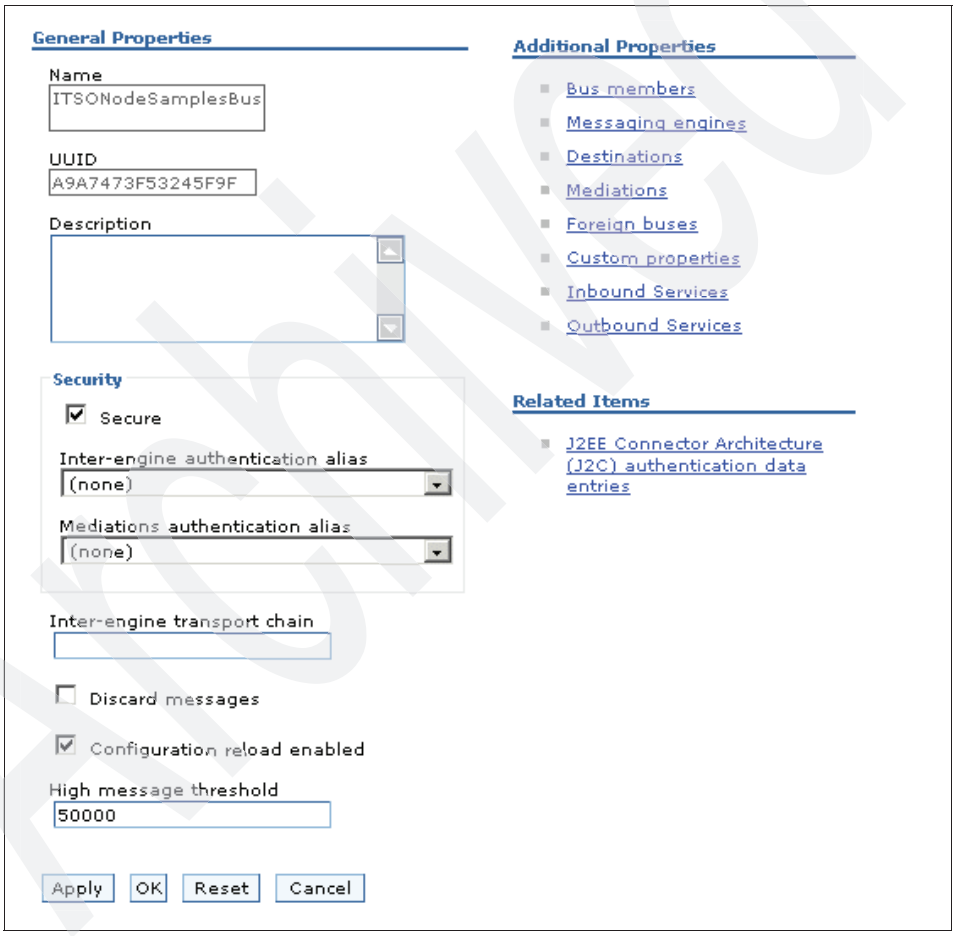


Figure 11-34 Bus creation panel

Archived

The only required property is **Name**. You cannot change the name of a bus after it has been created, but you can create any number of buses in a cell and delete old ones. Make your bus name unique and meaningful.

The following properties can also be set:

- **Description**

  This field is an optional description for the bus, for administrative purposes.

- **Secure**

  Select this option to inherit the Global Security setting of the cell. Deselect this option if you always want to disable bus security. When the Secure property is selected and global security for the cell is also selected, access to the bus itself and to all destinations on the bus must be authorized.

- **Inter-engine authentication alias**

  This field contains the name of the authentication alias used to authorize communication between messaging engines on the bus. This field is optional. If a value is specified, and bus security is enabled, incoming connections to the bus are controlled to prevent unauthorized clients or messaging engines from establishing a connection.

- **Mediations authentication alias**

  Enter the name of the authentication alias used to authorize mediations to access the bus.

- **Inter-engine transport chain**

  The transport chain used for communication between messaging engines in this bus. It must correspond to one of the transport chains defined in the Messaging engine inbound transports settings for the server. When you specify the name of a transport chain, that chain must be defined to all servers hosting messaging engines in the bus. Otherwise, some messaging engines might not be able to communicate with their peers in the bus. The default transport chain is InboundBasicMessaging.

- **Discard messages**

  Use this field to specify whether messages on a deleted message point should be retained at a system exception destination or can be discarded.

- **Configuration reload enabled**

  Select this option to enable certain changes to the bus configuration to be applied without requiring the messaging engines to be restarted. If you select this option, make sure the matching flag on the SIB service is also enabled. See 11.8.1, “SIB service configuration” on page 655.

Archived

– **High message threshold**

  Enter a threshold above which the messaging system will take action to limit the addition of more messages to a message point. When a messaging engine is created on this bus, the value of this property sets the default high message threshold for the messaging engine.

3. Click **Apply** or **OK** and save your changes.

11.8.3 Adding a bus member using a default data store

Every messaging engine has a data store associated with it. If you elect to use the default data store, a Cloudscape database will be created automatically and initialized with the messaging engine tables. To create create a bus member which automatically creates a messaging engine and uses the default Cloudscape database, do the following.

1. Select **Service integration** → **Buses.**
2. Select the bus to which you want to add a member.
3. Select **Bus members** in the Additional Properties section.
4. On the Bus members panel click **Add.** See Figure 11-35 on page 661.

Archived

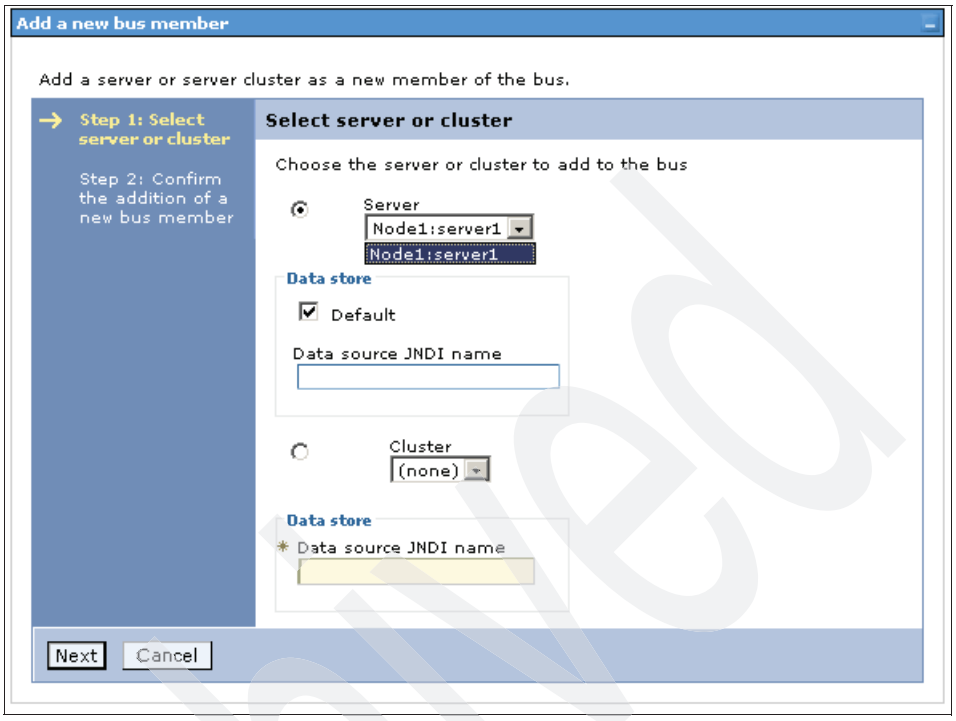


Figure 11-35 Adding a bus member

- Click the appropriate radio button to add a server or cluster, then select the server or cluster from the list.
- Check the **Default** box to use the default Cloudscape data store. You do not need to enter the data source JNDI name. It will default to the proper value when the server or cluster is added to the bus.

5. Click **Next** and then **Finish** and save your changes.

11.8.4 Adding a bus member with a different data store

This section discusses the steps required to create a bus member using a different data source from the default. In this section, we use DB2 as an example.

Creating a database

The first step is to create the new database and define the user IDs allowed to access the database. The privileges required are outlined in the Information Center. Refer to the *Data Stores* topic under the service integration bus administration topics for further information.

Archived

For example, The user ID for a DB2 database must have the following privileges:

- SELECT, INSERT, UPDATE, and DELETE privileges on the tables
- CREATETAB authority on the database
- USE privilege on the tablespace
- CREATEIN privilege on the schema

Use the **sibDDLGenerator** command to generate the DDL statements needed to create the data store for the messaging engine, including the proper privileges. For information about using this command, see the *sibDDLGenerator command* topic in the Information Center.

Creating a J2C authentication alias

To define access to the new database, define a J2C authentication alias containing the user ID and password defined in "Creating a database" on page 661.

1. Select **Security** → **Global security**.
2. Under **Authentication** expand the **JAAS Configuration** section and select **J2C Authentication data.**
3. Click **New**.
   a. Provide a name for this **Alias**. The alias name will be used later to identify this name as the one to access the database.
   b. Provide a **User ID** and **Password** that have permission to access the resource you will be using.
   c. Click **Apply** or **OK** and save your changes.

Creating a JDBC provider and data source

With this step, you define the database to the application server. First, a JDBC provider is defined to tell the application server how to find the libraries required to access the database. Information about defining JDBC providers and data sources can be found in 7.2, "JDBC resources" on page 321.

1. Select **Resources** → **JDBC Providers.**
2. Select the appropriate scope for the JDBC Provider. If you are adding a cluster as a bus member, then select that cluster as the scope. If you are adding a server as a bus member, then select the server as the scope.
3. Click **New.**
   a. Select a database type. In this example, we use **DB2.**
   b. Select the provider type. This is dependent on the database type. For a DB2 database, select **DB2 Universal JDBC Driver Provider**.

Archived

c. Select the implementation type. For DB2, use **Connection pool data source.**

4. Click **Next** and then click **Apply**.
5. The default values for the DB2 provider use a variable to designate the directory path where the JDBC drivers are found. Ensure that the DB2UNIVERSAL_JDBC_DRIVER_PATH environment variable is correctly set:
   a. Select **Environment** →**WebSphere Variables**
   b. Select an appropriate scope for the variable, usually node.
   c. Set the value for the DB2UNIVERSAL_JDBC_DRIVER_PATH variable to be the path to the Java folder in the DB2 installation on the host appropriate to the scope selected.
   d. Save your changes.

Note: When the data source is being created at cluster scope, each node that has a server in the cluster must have the DB2 JAR files available on it. The DB2UNIVERSAL_JDBC_DRIVER_PATH variable must be set appropriately for every node.

6. Create a data source for the bus member. Select **Resources** → **JDBC Providers** and then select your DB2 JDBC Provider.
7. Select **Data sources.**
8. Click **New** to create a new data source.
   a. Provide a **JNDI Name** for the data source. Remember this name because you will need to provide it when adding your cluster or server to the bus.
   b. Provide the Database name, Driver type and, optionally, the Server name. Get this information from your database administrator.
      i. The database name must be the name of an existing DB2 Database.
      ii. The driver type is 2 if the DB2 database exists locally or is catalogued locally. If the database is only available on a remote host, then the driver type is 4 and you must enter the **Server name**.

Note: There is no need to provide a component-managed authentication alias at this stage. That will be specified later in the data store of the messaging engine. Specifying the alias in either location is supported, but for tighter security control, we recommend that you specify it in the messaging engine's data store.

   c. Click **Apply** or **OK** and save your changes.

Adding the bus member

Once the database and supporting definitions are in place, the bus member can be added. To add the bus member, do the following:

1. Select **Service integration** → **Buses**. Select the bus you want.
2. Select **Bus members** in the Additional Properties section.
3. Click **Add**. See Figure 11-35.
4. To add a server to the bus, do the following:

   a. Select **Server** on the radio button.
   b. Select the server you want to add from the drop-down list.
   c. Provide the data source JNDI name for the data store.

   To add a cluster to the bus, do the following:

   a. Select **Cluster** on the radio button.
   b. Select the cluster you want to add from the drop down list.
   c. Enter the data source JNDI name for the data store.

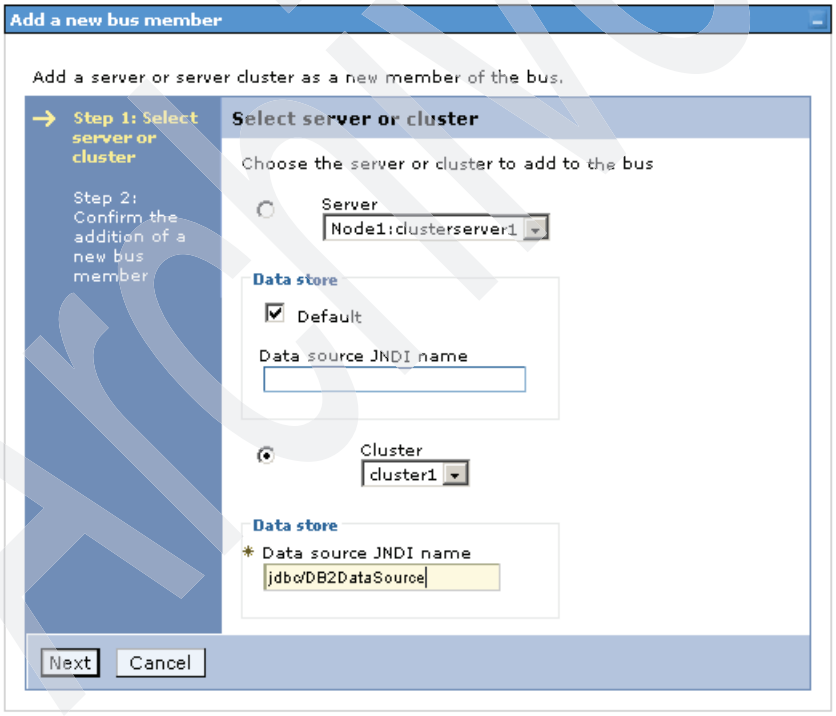


Figure 11-36 Adding a cluster to a bus.

5. Click **Next** and then click **Finish**.

Archived

6. You now have to set the J2C authentication alias on the newly created messaging engines data store. Select **Service integration** → **Buses**. Select the appropriate bus.
7. Select **Messaging engines.**
8. Select the messaging engine created when you added the bus member.
9. Select **Data store** in the Additional Properties section. See Figure 11-37.
10. From the menu, select the authentication alias, allowing the messaging engine to access the database.

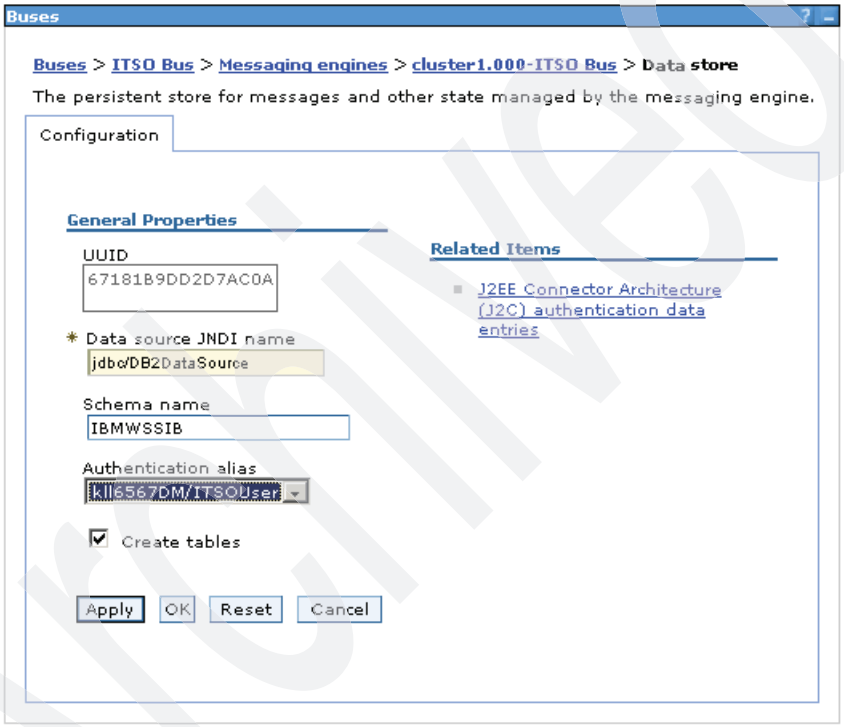


Figure 11-37 Defining the authentication alias for the data store

11. Ensure that the **Create tables** box is checked. The messaging engine will create all of the tables it needs in the database when it starts for the first time.

Archived

Important: The user ID in the authentication alias must have sufficient authority to be able to create tables in the database. Check with your database administrator.

If you do not want the data store to use an ID with authority to create and drop tables, then your database administrator must create the tables for you before you start the messaging engine. See the Information Center section on *Enabling your database administrator to create the data store tables*.

11.8.5 Creating a queue destination

Queue destinations are destinations that you can configure for point-to-point messaging.

1. Select **Service integration** → **Buses.**
2. Select the bus on which you want to create a queue.
3. Select **Destinations** in the Additional Properties section. See Figure 11-38.

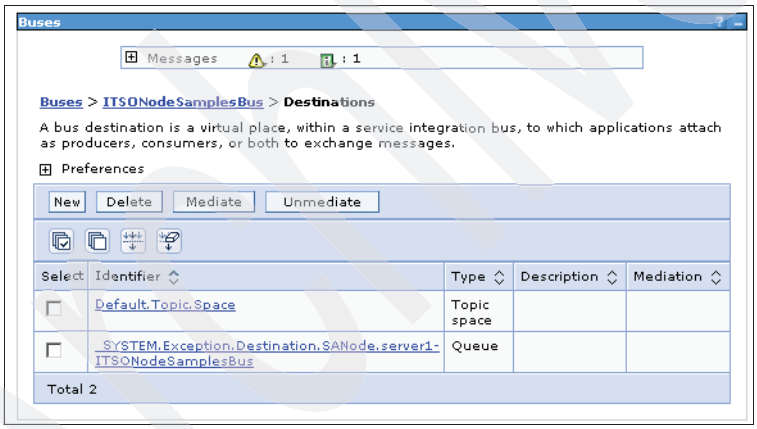


Figure 11-38 Default destinations

The Destinations panel shows two destinations that are created automatically for you. The Default.Topic.Space is a default topic space which can be used for publish/subscribe messaging. It can be deleted. The _SYSTEM.Exception.Destination is a built in queue which cannot be deleted.

4. Click **New.** See Figure 11-39 on page 667.

Archive

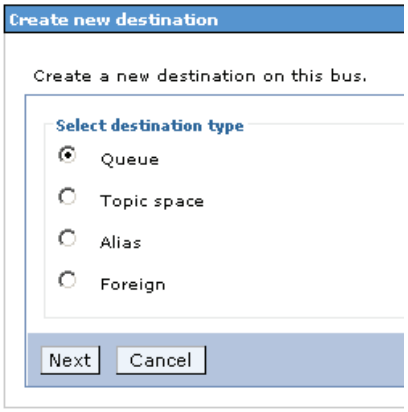


Figure 11-39 Options when creating a new destination

Select **Queue** from the radio button list and click **Next.** See Figure 11-40.

5. Provide an identifier and optional description for the queue.

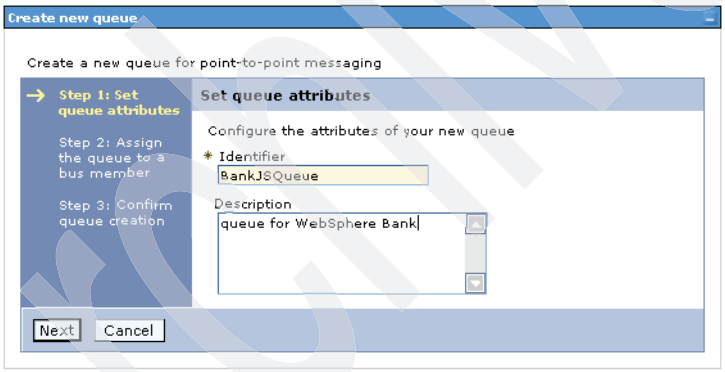


Figure 11-40 Provide an identifier for your destination

If your application uses the JMS interface, it is not sufficient to create a destination on the service integration bus. A JMS destination referencing the service integration bus destination must also be created. The identifier value specified here must match the Queue name property of the JMS queue definition (see “JMS queue configuration” on page 539).

Click **Next**. See Figure 11-41 on page 668.

6. Select a bus member for the queue point for this queue from the list for the queue. Click **Next**.

Archived

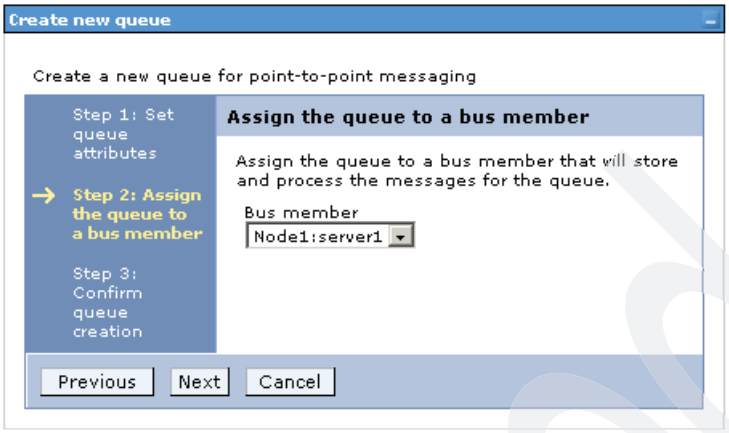


Figure 11-41 Select a bus member for the queue

7. Click **Finish** and then save your changes.

11.8.6 Creating a topic space destination

Topic space destinations are destinations that can be configured for publish/subscribe messaging.

1. Select **Service integration** → **Buses.**
2. Select the bus on which you want to create a topic space on.
3. Select **Destinations** in the Additional Properties section.
4. Click **New**.
5. Select **Topic space** from the list and click **Next**.
6. Provide an identifier and optional description for your topic space.
7. Click **Next**.
8. Click **Finished**.
9. Save your changes.

11.8.7 Creating an alias destination

Alias destinations refer to another destination, potentially on a foreign bus, providing an extra level of indirection for messaging applications. An alias destination can also be used to override some of the values specified on the target destination, such as default reliability and maximum reliability. Foreign buses are discussed in 11.1.7, “Foreign buses” on page 606.

1. Select **Service integration** → **Buses.**
2. Select the bus on which you want to create a topic space.
3. Select **Destinations**. in the Additional Properties section.

4. Click **New**.
5. Select **Alias** from the list and click **Next**. See Figure 11-42.

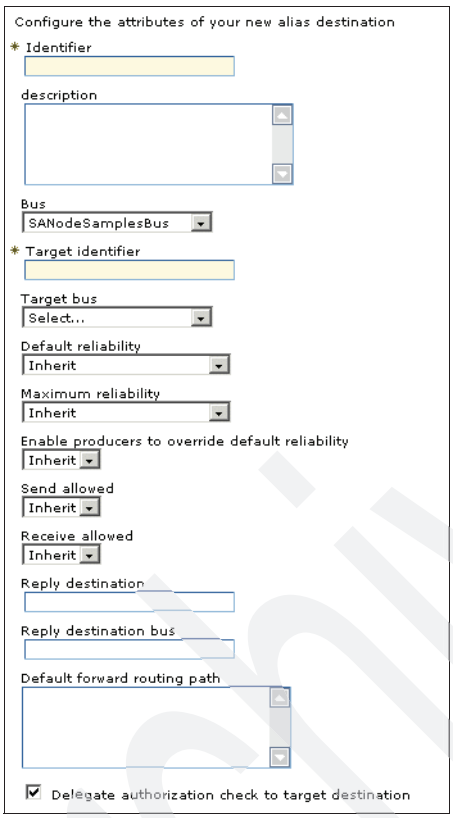


Figure 11-42 Alias destination properties

The properties to note are:

– **Identifier**

This field is the destination name as known by applications.

– **Bus**

Enter the name of the bus used by applications when referring to the alias destination.

If the destination that clients will attempt to access is known to them to be on a foreign bus, then select that bus from the menu. An example of this is if a foreign destination is configured in the JMS layer and you want to redirect client requests for that destination.

If the bus does not appear in the list, select **other**, specify from the list and enter the name of the bus in the text box.

Archived

If you leave the Bus field empty, the alias destination is created on the local bus.

- **Target identifier**

  Enter the identifier of the target destination to which you want this alias destination to route messages. If the alias destination is targeting a queue provided by WebSphere MQ, type the value as a concatenation of the queue name and the queue manager name, for example:

  `queue_name@qmanager_name; for example: Queue1@Qmgr2.`

- **Target bus**

  Enter the name of the service integration bus or foreign bus hosting the target destination. This can be the name of a foreign bus representing a WebSphere MQ network. The default is the name specified for the Bus property.

Override any of the other values on the panel that you want to override for the destination.

6. Click **Next**.
7. Click **Finished**.
8. Save your changes.

11.8.8 Adding messaging engines to a cluster

When you add a cluster to a bus, you get one messaging engine. To define additional messaging engines, do the following:

1. Ensure that you have defined a data source that the new messaging engine will use for its data store before starting this section, see “Creating a JDBC provider and data source” on page 662.
2. Select **Service integration** → **Buses**. Select the bus you want to use.
3. Select **Bus members** in the Additional Properties section.
4. Select the cluster bus member to which you want to add an additional messaging engine. This will display the list of messaging engines that are defined for the cluster bus member. See Figure 11-43 on page 671.

Archived

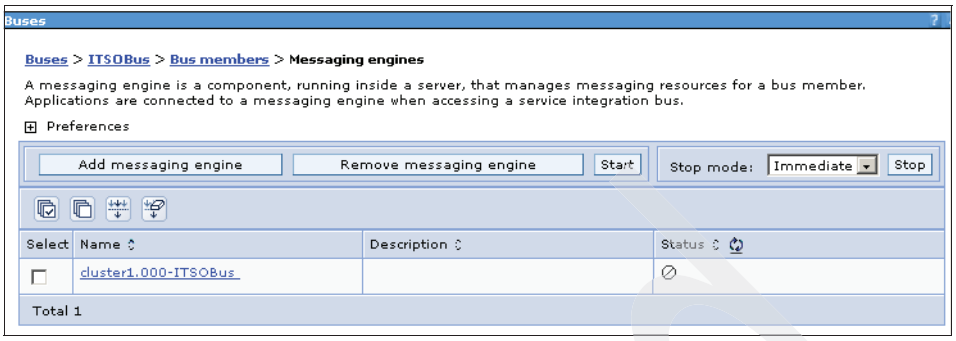


Figure 11-43 Messaging engines as part of a cluster bus member

5. Click **Add messaging engine**. Enter data store information for the new messaging engine as in Figure 11-44.

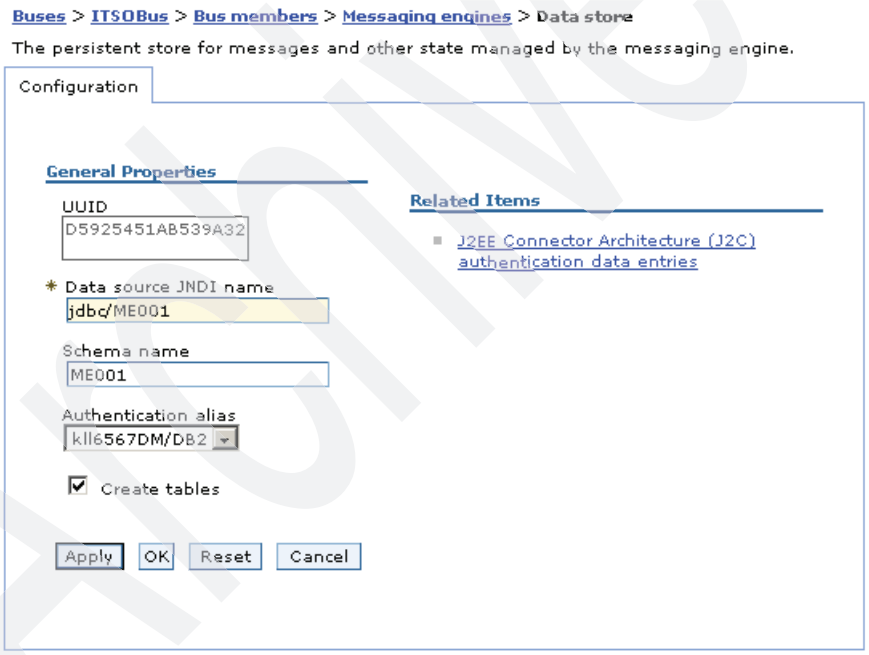


Figure 11-44 Provide information for the messaging engines data store.

6. Enter the required information for the database. For information about using multiple data stores, see 11.2.3, “Data stores” on page 620.
7. Click **Apply** or **OK** and save your changes.

Archived

11.8.9 Setting up preferred servers

Configure a messaging engine to prefer to run on one server or a group of servers in a cluster using a core groups policy. The use of policies is required if you want to workload-manage your messaging with the service integration bus.

Note: Before attempting to configure a system for workload management and high availability, consult the following:

- 11.3, “High availability and workload management” on page 638
- The *Configuring high availability and workload sharing of service integration* topic in the Information Center
- *WebSphere Application Server V6 Scalability and Performance Handbook*, SG24-6392.
- *WebSphere Application Server V6: High Availability Solutions,* REDP-3971

Setting up a policy with the appropriate values can give many different behaviors, including the following:

- A messaging engine will have an affinity for one particular server in the cluster. If that server fails, then the messaging engine will run on other servers, but will move back to the preferred server as soon as it becomes available. This is set up by having a One-of-N Policy defined with one preferred server configured, **preferred servers only** set to **false** and **fail back** set to **true**.
- A messaging engine will run on only one server inside the cluster. This means that the messaging engine cannot fail over to another server in the cluster and will only ever run on the defined server. This can be set up by having a One-of-N Policy with one preferred server and **preferred servers only** set to **true**.

Important: If you have more than one messaging engine defined on a cluster bus member and do not define additional core group policies to set up preferred servers, then all messaging engines will start and run on the first server to become available.

To create a core group policy for a messaging engine, do the following:

1. Select **Servers** → **Core groups** → **Core group settings**.
2. Select the **DefaultCoreGroup**. This will show the properties for the default core group. See Figure 11-45 on page 673.

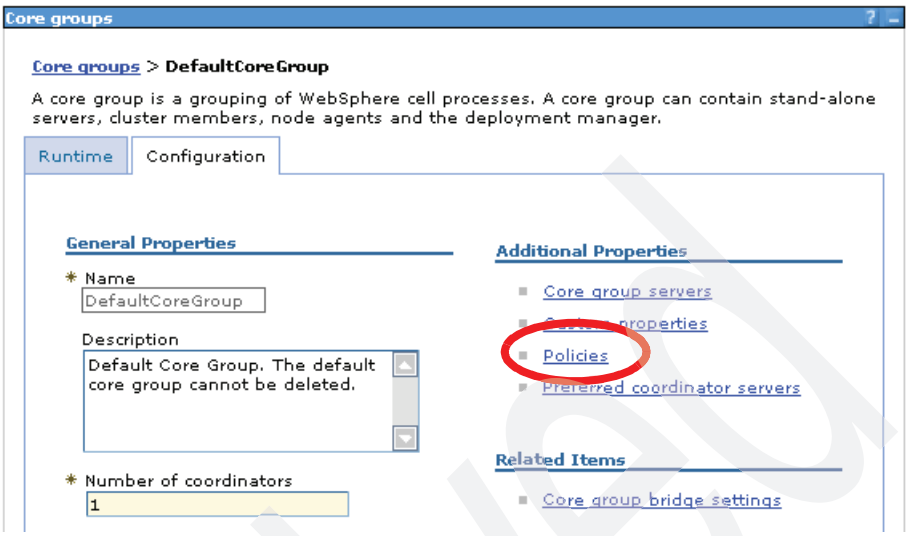


Figure 11-45 Default Core Group

3. Select **Policies** in the Additional Properties section. This will show you the list of policies defined for the core group. Two policies are created by default. Do not delete or modify these policies. See Figure 11-46.

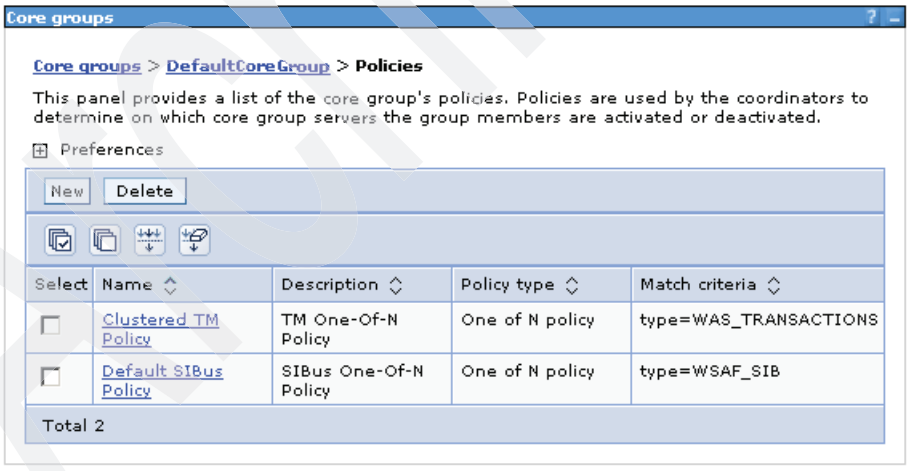


Figure 11-46 Predefined core group policies in the default core group.

4. Click **New**.
5. From the drop-down list select the **One of N Policy**. Click **Next.** See Figure 11-47 on page 674.

6. Enter a name for the new policy. It might be helpful if the name includes the name of the messaging engine for which you are creating this policy.

   Enable **fail back** and **preferred servers only** as desired. These settings can be changed later.

   Click **Apply**.

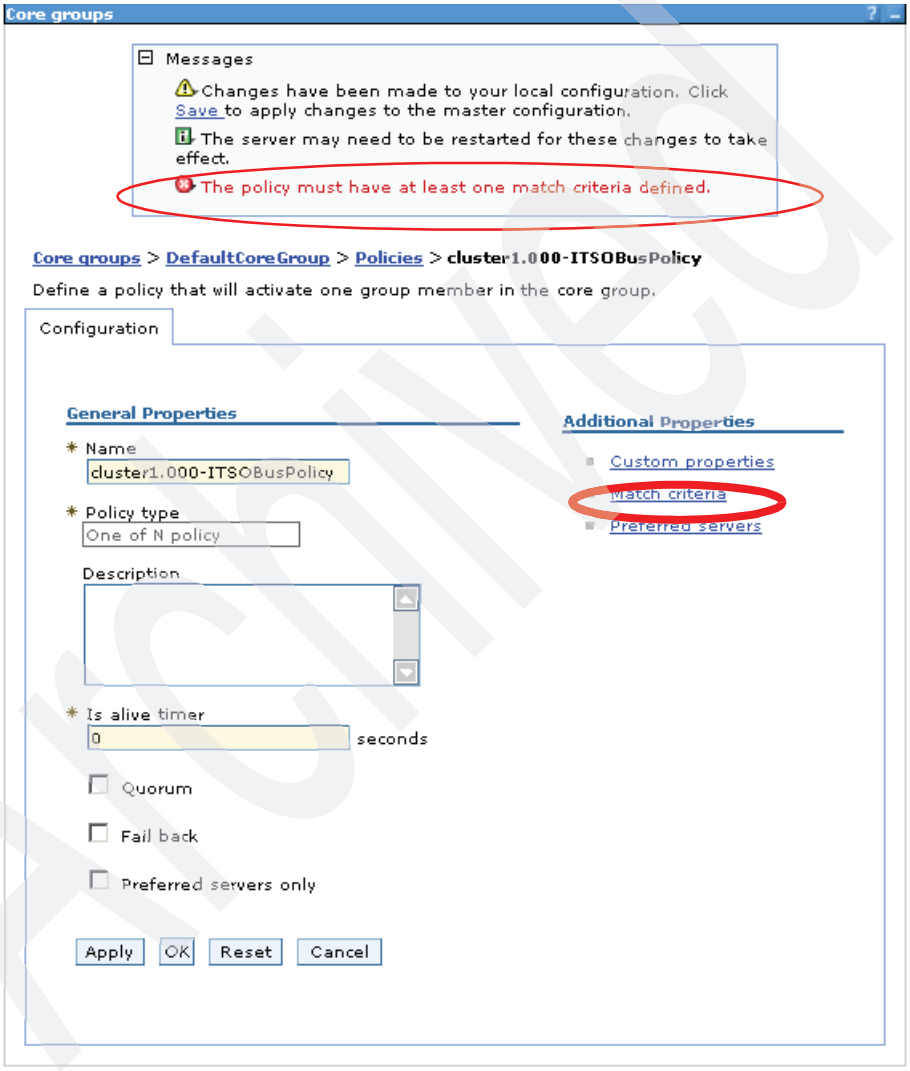


Figure 11-47 Defining a new policy

Archived

Important: Be aware that if you set **preferred servers only** that this can prevent the messaging engine from being highly available. If the messaging engine or the server it runs on fails or stops and no other servers that are preferred are available, then the messaging engine cannot be started on other servers that are available in the cluster. They are not preferred and only preferred servers can be used.

7. A warning will show that you must define at least one match criteria. Match criteria are name and value pairs used to match server components such as messaging engines.
8. Select **Match criteria** in the Additional Properties section.
9. Click **New**. See Figure 11-48.

   Enter `type` for the name and `WSAF_SIB` for the value. This match criteria will match any messaging engine.

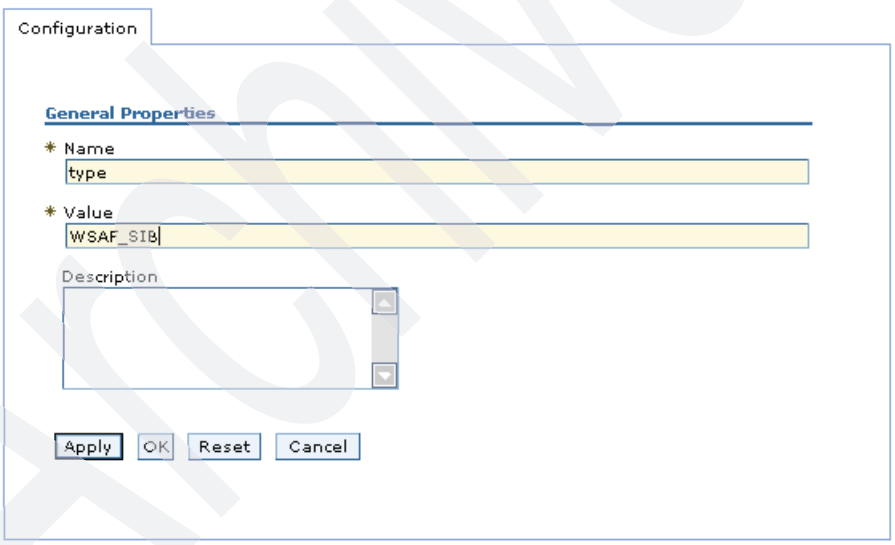


Figure 11-48 Defining match criteria for any messaging engine

   Click OK.

10. Click **New** to define another set of match criteria. See Figure 11-49 on page 676.

    Enter `WSAF_SIB_MESSAGING_ENGINE` for the name and the messaging engine name for the value.

Archived

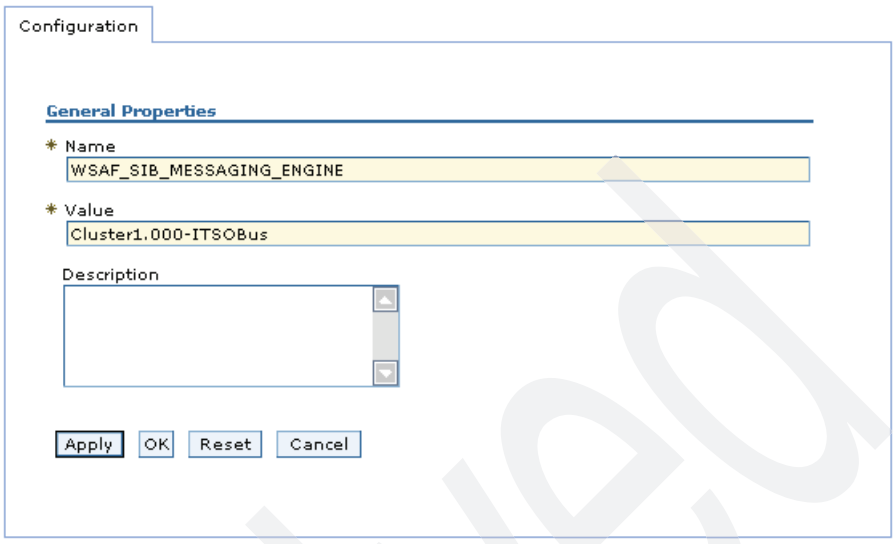


Figure 11-49 Defining a match criteria for a specific messaging engine

Click **OK**.

11. Return to your policy by clicking the policy name in the navigation trail. See Figure 11-50 on page 677.

Archived

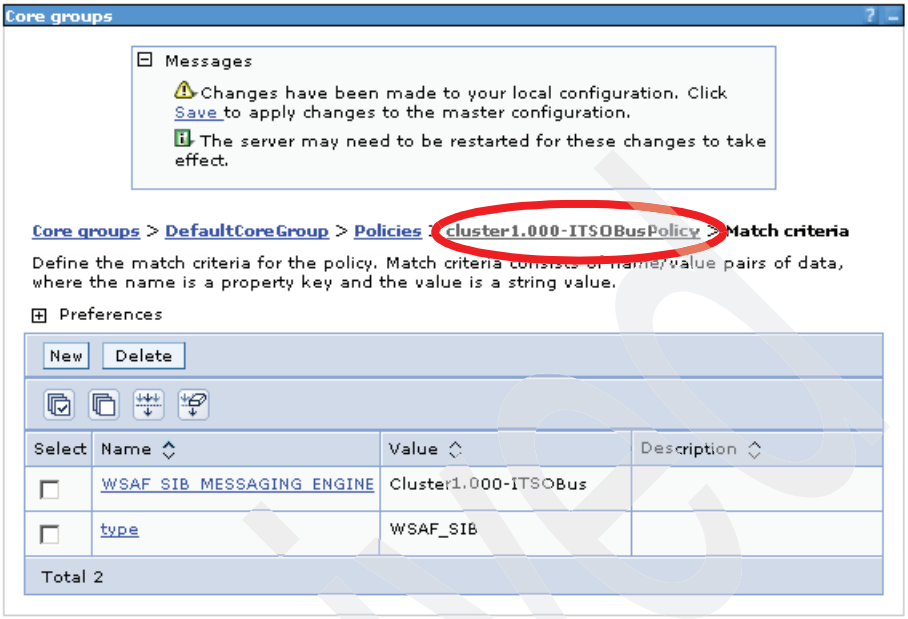


Figure 11-50 Match criteria for a messaging engine

12. Click **Preferred servers** in the Additional Properties section.
13. Select the servers you want to configure as preferred and click **Add>>**.

    You can select as many preferred servers as you want. All preferred servers must be servers that are in the cluster on which the messaging engine is defined. Do not select a node agent or deployment manager.

    See Figure 11-51 on page 678. Preferred servers have an order of preference. The higher up the list of preferred servers, the more preferred the server will be. To move a server up or down the list select the server and click **Move up** or **Move down**. If **Fail back** is enabled, then a messaging engine will fail over to the highest available server in the list.

Archive

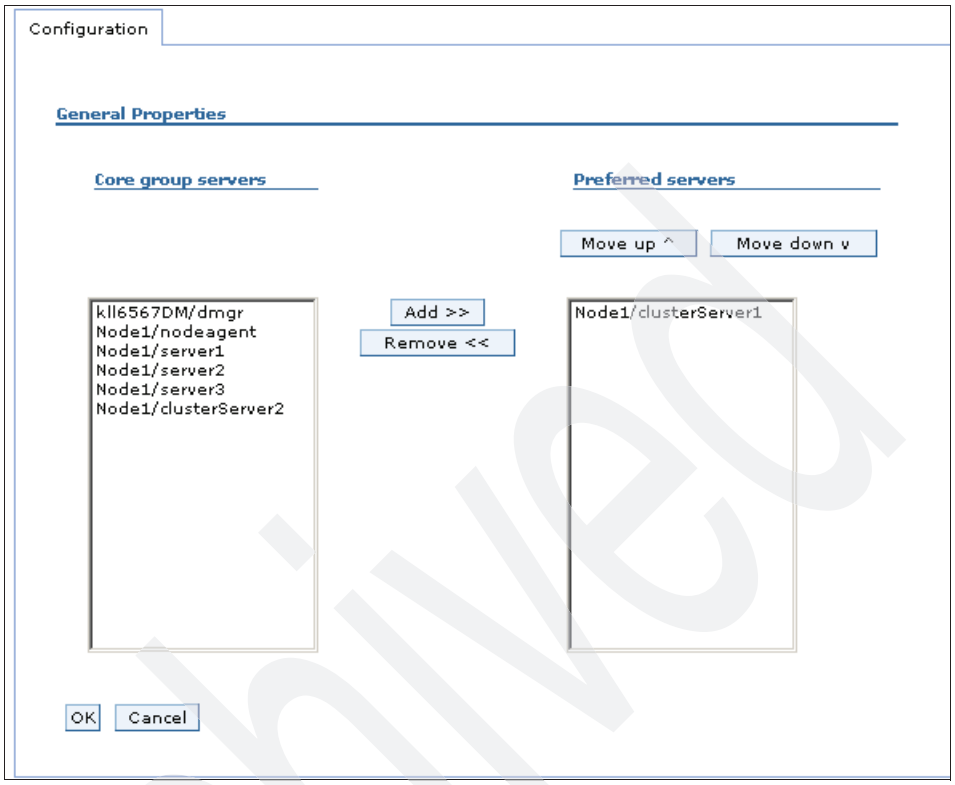


Figure 11-51 Selecting preferred servers for a core group policy

14. Click **OK** and save your changes.

11.8.10 Setting up a foreign bus link to a service integration bus

To define a foreign bus to the bus from which you want to access it, do the following:

1. Select **Service integration** → **Buses**.
2. Select the bus from which you want to access the foreign bus.
3. Select **Foreign buses** in the Additional Properties section.
4. Click **New**. See Figure 11-52 on page 679.
5. Provide the **Name** of the foreign bus.

> **Important:** When your foreign bus is a service integration bus, then this name must match exactly the name of that bus.

Archived

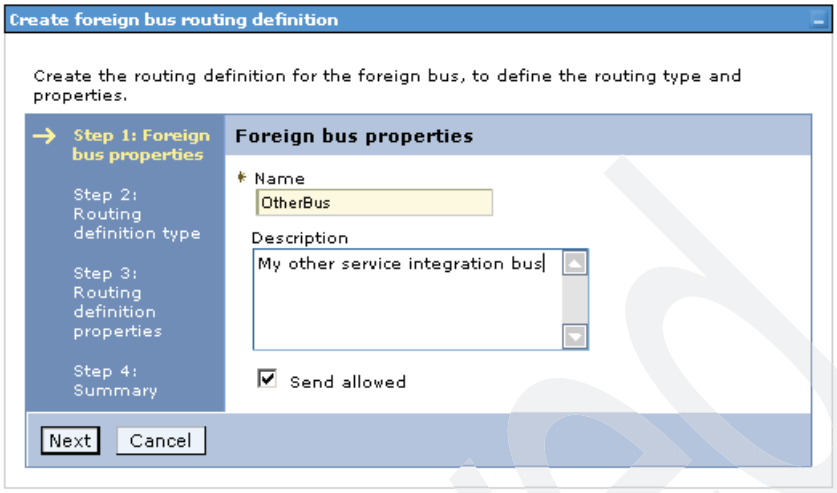


Figure 11-52 Creating a new foreign bus

Checking the **Send allowed** box allows this bus to send messages to destinations on the foreign bus. This is the default.

You can change this setting at any time. This can be useful if you want to disable a foreign bus for a short time, for example while configuration changes are being made.

Click **Next**. See Figure 11-53.

6. Select the appropriate value for the routing type.

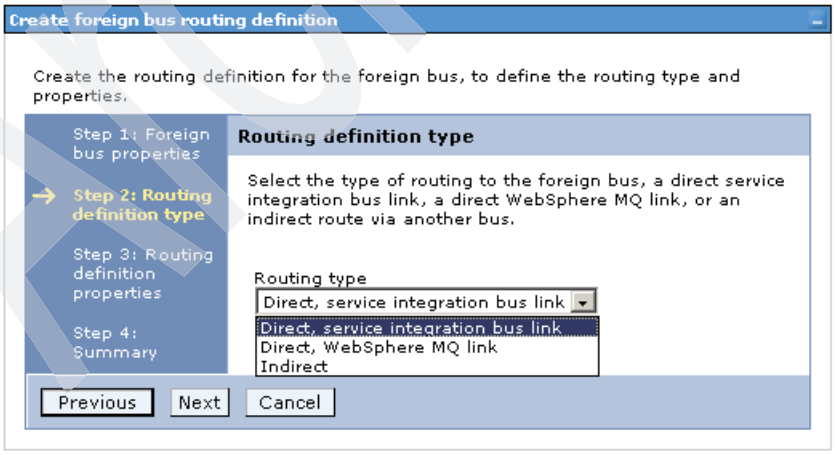


Figure 11-53 Selecting the type of foreign bus

Archived

To define another service integration bus, select **Direct, service integration bus link** from the menu. Click **Next**. See Figure 11-54.

7. Optionally, define outbound and inbound user IDs.

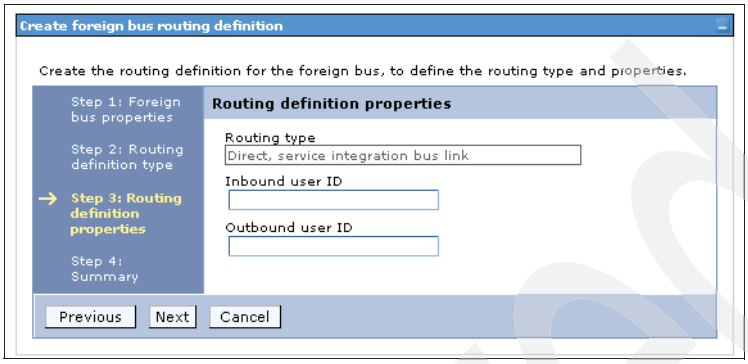


Figure 11-54 Define inbound and outbound user IDs

The inbound user ID authorizes individual messages arriving from the foreign bus to destinations in this bus. When set, this property replaces the user ID in messages entering this bus from the foreign bus. If this is not a secure bus, this property does not affect on messages.

The outbound user ID replaces the user ID that identifies the source of a message in all messages being sent to the foreign bus. When set, this property replaces the user ID in messages leaving this bus for the foreign bus. The foreign bus also uses this ID to authorize the message to its destination if both buses are secure buses and the foreign bus has not overridden the user ID with its own inbound user ID.

8. Click **Next**.
9. Click **Finish** and save your changes.

Define the link to the service integration bus

Now that your bus knows about the foreign bus, you will have to set up the link to that bus. This link will be managed by a particular messaging engine on your bus. A link must be created on each bus and it is important that the link has the same name on each bus.

1. Select **Service integration** → **Buses**. Select the bus you want to use.
2. Select **Messaging engines** in the Additional Properties section and select the messaging engine you want to host the link.
3. Select **Service integration bus link** in the Additional Properties section
4. Click **New**. See Figure 11-55 on page 681 and fill in the following properties:

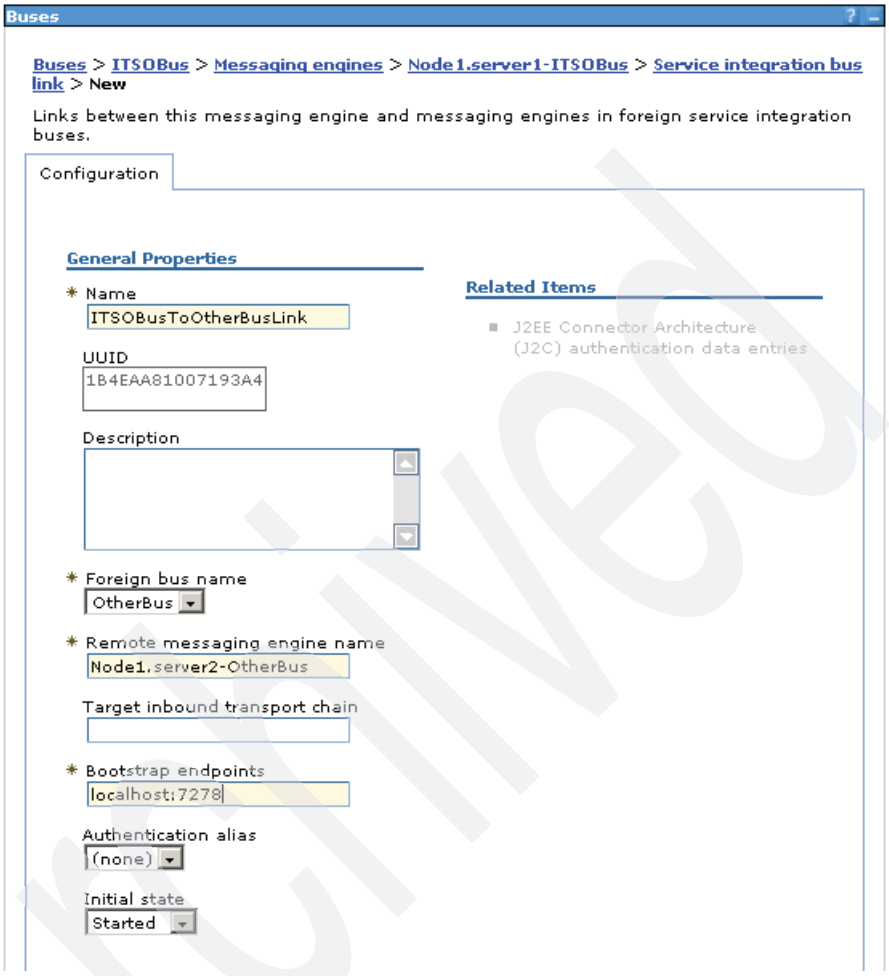


Figure 11-55 Defining a service integration bus link

– **Name**

Enter a name for the link. It might be helpful if this name includes the names of the buses you are linking.

> **Important:** This link name must be the *exactly the same* as the link name on the other bus.

– **Foreign bus name**

Archived

Enter the name of the messaging engine in the foreign bus to which you are linking. This name must also match exactly the name of the messaging engine in the foreign bus hosting the link and is required to prevent configuration changes on the other bus from causing problems with the link.

– **Bootstrap endpoints**

Provide bootstrap endpoints to allow your bus to connect to the foreign bus. This field is equivalent to the **Provider endpoints** field for a default messaging provider connection factory. Both provide a list of endpoints to be used to connect to a SIB service.

See 10.7.1, “JMS client runtime environment” on page 576.

– **Authentication alias**

If the foreign bus is secure, then you need to provide authentication data for the link.

5. Click **Apply** or **OK** and save your changes.

Important: You must configure a corresponding foreign bus and service integration bus link on the other bus to complete the link. Ensure that the name of the link is the same in both buses.

Configuring topic space mappings to a foreign bus

This section discusses the steps required to create a topic space mapping between two service integration buses. Before starting this section, you must have defined a foreign bus that is a service integration bus.

- Select **Service integration** → **Buses**. Select the local bus.
- Select **Foreign buses** in the Additional Properties section.
- Select the foreign service integration bus you to which you want to create a topic mapping.
- Select **Service integration bus link routing properties** in the Additional Properties section.
- Select **Topic space mappings** in the Additional Properties section.
- Optionally, enter a description.

  Click **Apply**.

  Note: You have to click **Apply** even if you do not enter a description.

- Select **Topic space map entries** in the Additional Properties section.

- Click **New**.
  - Enter the name of the **Local topic space** from which you want to receive published messages.
  - Enter the name of the **Remote Topic space** from which you want to receive published messages.

  Click **Apply** or **OK** and save your changes.

11.8.11 Setting up a foreign bus link to an MQ queue manager

A WebSphere MQ link allows your service integration bus to exchange messages with a WebSphere MQ queue manager.

Note: Before creating these definitions, review the information in 11.2.6, “WebSphere MQ links” on page 629.

First, you must define a foreign bus and define it in your bus. From there, enter information in the following fields.

1. Select **Service integration** → **Buses**. Select the bus you want to use.
2. Select **Foreign buses** in the Additional Properties section.
3. Click **New**. See Figure 11-56. Enter information in the following fields.

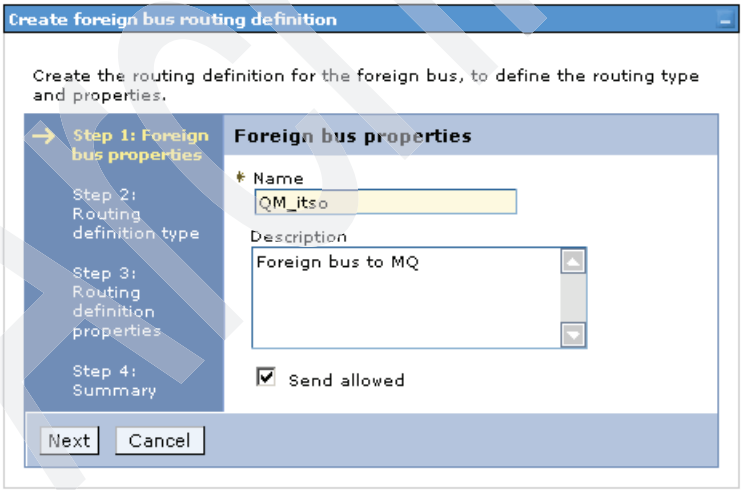


Figure 11-56 creating a new foreign bus

– **Name**

  Enter the name of the foreign bus.

– **Send allowed**

  Checking the **Send allowed** box allows this bus to send messages to destinations on the foreign bus. This is the default. You can change this setting at any time. This can be useful if you want to disable a foreign bus for a short time, for example while configuration changes are being made.

Click **Next**. See Figure 11-57.

4. Select **Direct, WebSphere MQ link** from the menu.

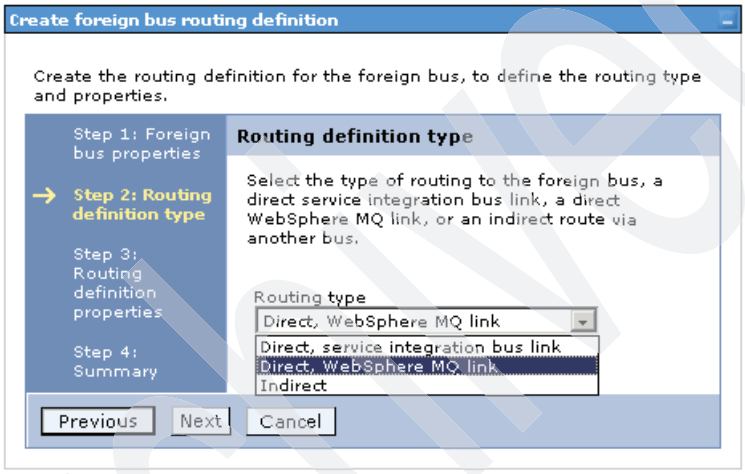


Figure 11-57 Selecting the type of foreign bus

5. Define inbound and outbound user IDs (optional)

   The inbound user ID authorizes individual messages arriving from the foreign bus to destinations in this bus. When set, this property replaces the user ID in messages entering this bus from the foreign bus. If this is not a secure bus, this property does not affect messages.

   The outbound user ID replaces the user ID that identifies the source of a message in all messages being sent to the foreign bus. When set, this property replaces the user ID in messages leaving this bus for the foreign bus. The foreign bus also uses this user ID to authorize the message to its destination if both buses are secure buses and the foreign bus has not overridden the user ID with its own inbound user ID.

6. Click **Next**.
7. Click **Finish** and save your changes.

Define the WebSphere MQ link

Now that your bus knows about the foreign bus, set up the link to that MQ queue manager. This link is managed by a particular messaging engine on your bus.

Important: If you are unsure of any of the correct MQ values to supply for the MQ link, then refer to your MQ administrator or documentation for more information.

1. Select **Service integration** → **Buses**.
2. Select the bus you want to use.
3. Click **Messaging engines** and select the messaging engine you want to host the link.
4. Select **WebSphere MQ Links** in the Additional Properties section.
5. Click **New**. See Figure 11-58 on page 686 and enter information in the following fields.

Archived

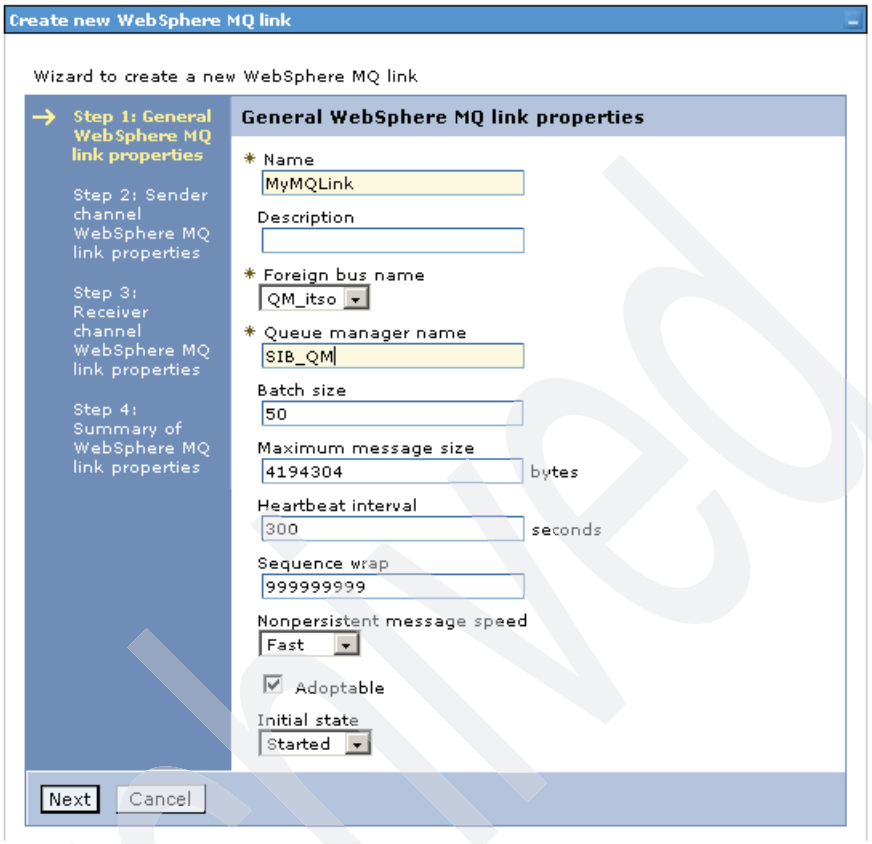


Figure 11-58 Defining properties for a new MQ link

- **Name**

  Enter a name for the link. It might be helpful if this name includes the name of the foreign bus for which you are creating the link.

- **Foreign bus name**

  From the menu select the name of the foreign bus to which this link will connect. This should be the name of the queue manager that is participating in the link.

- **Queue manager name**

  This is the queue manager name by which the MQ queue manager will know this bus. You must ensure that the MQ queue manager participating in this link is configured to know about this bus as another queue manager using this queue manager name.

Archived

– **Nonpersistent message speed**

This field define whether the channel to MQ will have MQ's NPMSPEED channel attribute set to `fast` or `normal`.

– Adoptable

This option, enabled by default, provides function similar to MQ's ADOPTMCA function. If selected, the receiver channel can be reused when the sender channel fails or has to be restarted.

Click **Next**.

6. You must now provide details on how the link will send information to the MQ queue manager. See Figure 11-59. Enter the following:

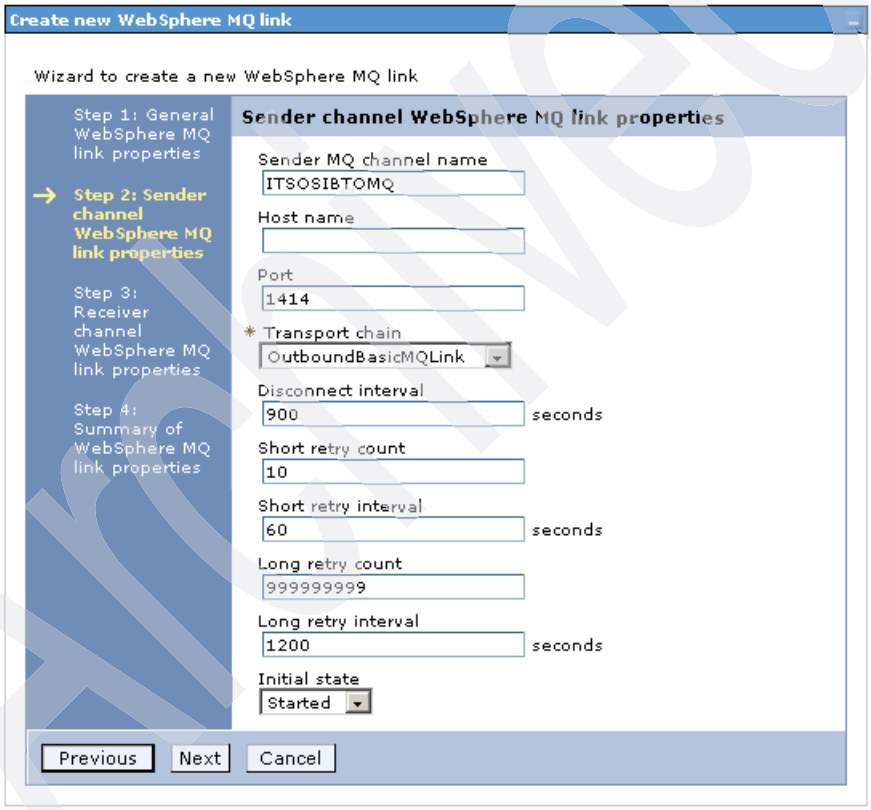


Figure 11-59 Providing the link with details on how to send messages to MQ

– **Sender MQ channel name**

This is the name of the receiver channel that the link will send messages to in the MQ queue manager.

– **Host name**

  Enter the host name or IP address of the server hosting the MQ queue manager.

– **Port**

  If the MQ queue manager is using a port other than the default port of 1414, then enter that information.

– **Transport chain**

  Select the appropriate transport chain from the menu. See 11.2.2, “Service integration bus transport chains” on page 614 for further information.

Click **Next**. See Figure 11-60.

7. Enter information about the virtual queue manager. Remember, this link performs as a virtual queue manager for WebSphere MQ. Enter the following information:

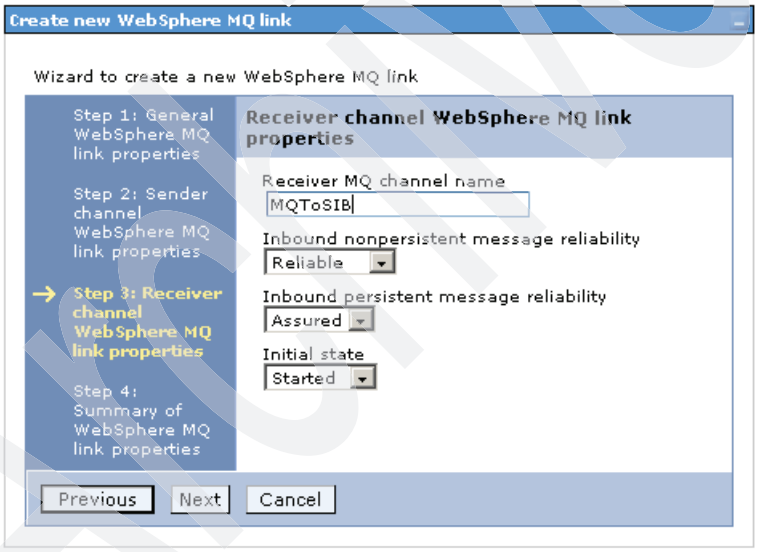


Figure 11-60 Providing the link with details on how MQ will send messages to it

– **Receiver MQ channel name**

  The MQ queue manager participating in the link must be configured with a sender channel of this name.

Archived

– Provide default information for mapping incoming persistent and nonpersistent MQ messages into service integration messages, see “Reliability” on page 604 for more information about service integration message reliability.

Click **Next**.

8. Click **Finish** and save your changes.

WebSphere MQ considerations: Ensure that the WebSphere MQ queue manager participating in the link has the appropriate sender and receiver channels defined. Consult your MQ administrator or documentation for details on how to perform this configuration.

- **Sender channel**

  This channel must have the same name as the name defined in the MQLink’s receiver channel.

  The connection name is the IP address or host name for the server hosting the messaging engine on which the link is defined.

  The port used should match the value of the `SIB_MQ_ENDPOINT_ADDRESS` port defined for the server hosting the messaging engine on which the link is defined. The default is 5559. To find this value through the administrative console, do the following:

  a. Select **Servers** → **Application Servers.**

  b. Select the server hosting the messaging engine.

  c. Under Communications expand the Ports heading. Find the port number for `SIB_MQ_ENDPOINT_ADDRESS`.

- **Receiver channel**

  This channel must have the same name as the name defined in the MQLink’s sender channel.

Configuring topic space mappings to WebSphere MQ

To configure an MQ publish/subscribe profile, define a WebSphere MQ link to the WebSphere MQ network. The link does not need to be directly to the broker’s queue manager. However, it must be to a queue manager that is able to route to the broker’s queue manager.

1. Select **Service integration** → **Buses**. Select the bus you want to use.
2. Click **Messaging engines** and select the messaging engine that hosts the MQ link.
3. Select **WebSphere MQ links** in the Additional Properties section.

4. Select the MQ link on which you want to define a MQ publish/subscribe profile.
5. Select **Publish/Subscribe broker profiles** in the Additional Properties section.Click **New**.
   - Enter a name for the profile.
   - Enter the name of the queue manager associated with the broker.

   Click **Apply**. Define some topic mappings to link MQ topics to service integration topics.
6. Select **Topic mappings** in the Additional Properties section.
7. Click **New**. See Figure 11-61 and enter information in the following fields:

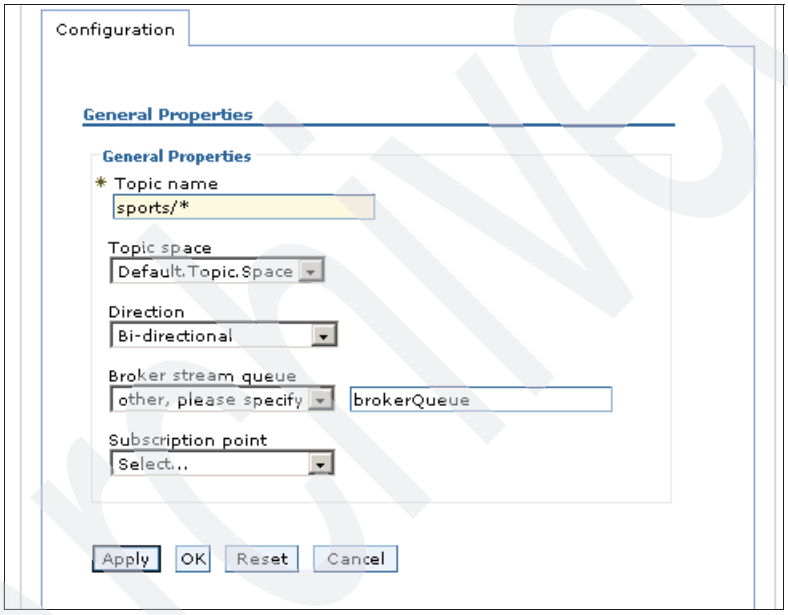


Figure 11-61 Defining a new Topic mapping

- **Topic name**

  Enter the name of the topic that you want to map. This name is the topic name that will be linked in the service integration bus and MQ. You can use service integration bus wildcards in this topic name to map a group of topics. For more information about the use of wildcard characters when specifying topic names, refer to “Topic specific connection properties” on page 537.

Archive

– **Topic space**

  Select the Topic space in which this topic will be published on the service integration bus.

– **Direction**

  Select the desired direction of the mapping.

  • **Bi-directional**

    Messages published in either WebSphere MQ or the service integration bus will be published in both.

  • **To WebSphere MQ**

    Messages published in the service integration bus will be published in WebSphere MQ, but messages published in WebSphere MQ will not be published in the service integration bus.

  • **From WebSphere MQ**

    Messages published in WebSphere MQ will be published in the service integration bus, but messages published in the service integration bus will not be published in WebSphere MQ.

– **Broker stream queue**

  Select the appropriate broker stream queue from the menu, if required**.** The broker stream queue is the queue in MQ to which the message broker is connected. If the queue does not appear in the list then, select **other, please specify**. A text entry box will appear to the right of the drop-down menu. Enter the name of the queue there.

  Note: Broker stream queue is required if you want to send messages to WebSphere MQ. If your topic mapping is **Bi-directional, To WebSphere MQ** or if it is **From WebSphere MQ** and your applications need to be able to send reply messages to publications received, then a broker stream queue must be specified.

– **Subscription point**

  Select an appropriate subscription point from the menu, if required. If the subscription point does not appear in the list then select **other, please specify**. A text entry box will appear to the right of the dropdown menu. Enter the name of the subscription point there.

  Ask your WebSphere MQ administrator if a subscription point should be specified and what it should be.

Archived

11.8.12 Creating a foreign destination

To create a destination on a foreign bus, do the following:

1. Select **Service integration** → **Buses.**
2. Select the bus on which you want to create a queue.
3. Select **Destinations** in the Additional Properties section.
4. Click **New**.
5. Select **Foreign** from the list and click **Next**. Enter information as in Figure 11-62.

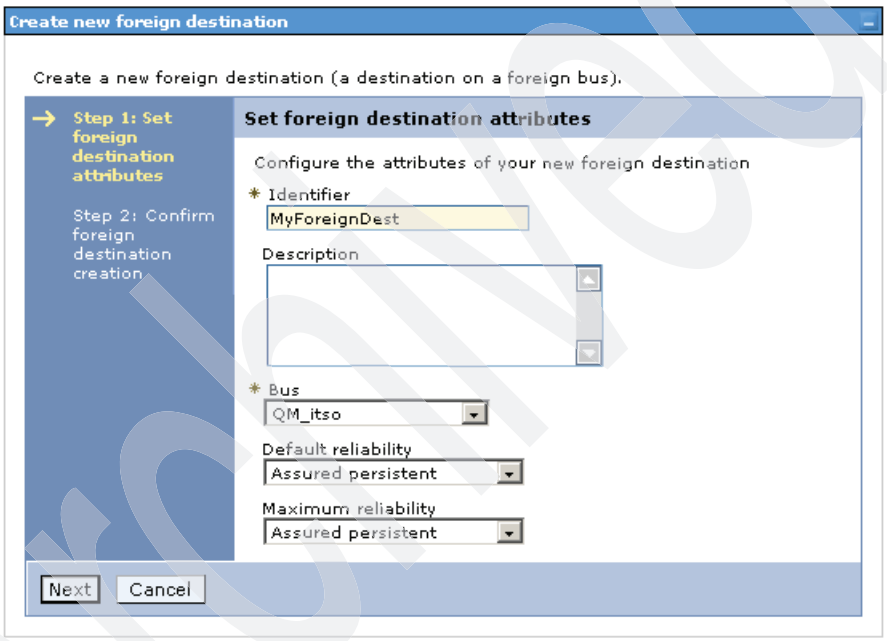


Figure 11-62 Creating a new foreign destination

– **Identifier**

Enter the name of the foreign destination for which you want to provide defaults. This must match the name of the destination that exists on the foreign bus.

Archived

– **Bus**

From the drop-down menu, select the foreign bus on which this destination exists. If the foreign bus is not in the list then select **other, please specify** and enter the name of the foreign bus in the box.

– **Default reliability and Maximum reliability**

Select the desired default and maximum reliabilities from the drop-down menus. Consult “Reliability” on page 604.

Click **Next**.

6. Click **Finished**.

Archived

Archived

Part 3

Working with applications

This part takes you through the process of packaging and deploying applications. In addition, it contains information about concepts that you need to understand to successfully develop and package applications for the WebSphere Application Server V6 runtime environment.

This part includes the following chapters:

- Chapter 12, “Session management” on page 697
- Chapter 13, “WebSphere naming implementation” on page 769
- Chapter 14, “Understanding class loaders” on page 821
- Chapter 15, “Packaging applications” on page 847
- Chapter 16, “Deploying applications” on page 907
- Chapter 17, “WebSphere Rapid Deployment” on page 957

© Copyright IBM Corp. 2005. All rights reserved.

Archived

12

Session management

Session support allows a Web application developer to maintain state information across multiple user visits to the application. In this chapter, we discuss HTTP session support in WebSphere Application Server V6 and how to configure it. We also discuss the new support for stateful session bean failover. The topics include:

- 12.1, “What is new?” on page 698
- 12.2, “HTTP session management” on page 698
- 12.3, “Session manager configuration” on page 699
- 12.4, “Session scope” on page 700
- 12.5, “Session identifiers” on page 702
- 12.6, “Local sessions” on page 710
- 12.7, “General properties for session management” on page 711
- 12.8, “Session affinity” on page 715
- 12.9, “Persistent session management” on page 719
- 12.10, “Invalidating sessions” on page 749
- 12.11, “Session security” on page 751
- 12.12, “Session performance considerations” on page 752
- 12.13, “Stateful session bean failover” on page 760
- 12.11, “Session security” on page 751

© Copyright IBM Corp. 2005. All rights reserved.

12.1 What is new?

The following improvements have been made to session management in WebSphere Application Server V6:

- The Data Replication Service (DRS) that provides session persistence using memory-to-memory replication has the following improvements:
  - Simplified topology
  - Simplified configuration
  - Improved performance with faster underlying transport
  - Integration with workload management (WLM) to provide hot failover in peer-to-peer mode
- Stateful session bean failover support

  WebSphere Application Server V6 utilizes the functions of DRS and WLM to enable stateful session bean failover.

> **Note:** The above features are only available for Network Deployment distributed server environments.

12.2 HTTP session management

In many Web applications, users collect data dynamically as they move through the site based on a series of selections on pages they visit. Where the user goes next, and what the application displays as the user's next page, or next choice, depends on what the user has chosen previously from the site. For example, if the user clicks the checkout button on a site, the next page must contain the user's shopping selections.

In order to do this, a Web application needs a mechanism to hold the user's state information over a period of time. However, HTTP alone does not recognize or maintain a user's state. HTTP treats each user request as a discrete, independent interaction.

The Java servlet specification provides a mechanism for servlet applications to maintain a user's state information. This mechanism, known as a *session*, addresses some of the problems of more traditional strategies such as a pure cookie solution. It allows a Web application developer to maintain all user state information at the host, while passing minimal information back to the user through cookies, or another technique known as *URL rewriting*.

12.3 Session manager configuration

Similar to WebSphere Application Server V5, session management in WebSphere Application Server V6 can be defined at the following levels:

- Application server

  This is the default level. Configuration at this level is applied to all Web modules within the server.

- Application

  Configuration at this level is applied to all Web modules within the application.

- Web module

  Configuration at this level is applied only to that Web module.

12.3.1 Session management properties

With one exception, the session management properties you can set are the same at each configuration level:

- *Session tracking mechanism* lets you select from cookies, URL rewriting, and SSL ID tracking. Selecting cookies will lead you to a second configuration page containing further configuration options.
- Select *Maximum in-memory session count* and whether to allow this number to be exceeded, or overflow.
- *Session timeout* specifies the amount of time to allow a session to remain idle before invalidation.
- *Security integration* specifies that the user ID be associated with the HTTP session.
- *Serialize session access* determines if concurrent session access in a given server is allowed.
- *Overwrite session management,* for enterprise application and Web module level only, determines whether these session management settings are used for the current module, or if the settings are used from the parent object.
- *Distributed environment settings* select how to persist sessions (memory-to-memory replication or a database) and set tuning properties. Memory-to-memory persistence is only available in a Network Deployment distributed server environment.

Archived

12.3.2 Accessing session management properties

You can access all configuration settings using the administrative console.

Application server session management properties

To access session management properties at the application server level, from the administrative console, do the following:

1. Click **Servers** →**Application servers**.
2. Click the application server.
3. In the Container Settings section of the Configuration tab, click **Web Container Settings**.
4. In the Additional Properties section, click **Session management**.

Application session management properties

To access session management properties at the application level, from the administrative console, do the following:

1. Click **Applications** → **Enterprise Applications**.
2. Click the application.
3. In the Additional Properties section of the Configuration tab, click **Session management**.

Web module session management properties

To access session management properties at the Web module level, from administrative console, do the following:

1. Click **Applications** →**Enterprise Applications**.
2. Click the application.
3. In the Related Items section of the Configuration tab, click **Web modules**.
4. Click the Web module.
5. In the Additional Properties section, click **Session Management**.

12.4 Session scope

The Servlet 2.4 specification defines session scope at the Web application level. Session information can be accessed only by a single Web application. However, there can be times when there is a logical reason for multiple Web applications to share information, for example, a user name.

Archived

WebSphere Application Server provides an IBM extension to the specification allowing session information to be shared among Web applications within an enterprise application. This option is offered as an extension to the application deployment descriptor. No code change is necessary to enable this option. This option is specified during application assembling.

Note: Because session information is shared within the enterprise application, you cannot use the Overwrite Session Management property at the Web module level when the IBM option for shared session context is selected.

Sharing session context

The WebSphere extension for sharing session context is set in the META-INF/ibm-application-ext.xmi file in the enterprise project. You can set this using the Application Server Toolkit or from Rational Application Developer:

1. Start the Application Server Toolkit or Rational Application Developer and switch to the J2EE perspective.
2. Double-click the **EAR file** in the J2EE Hierarchy view. This will open the application deployment descriptor.
3. Click the **Overview** tab.
4. Select **Shared session context**. See Figure 12-1 on page 702.

Archived

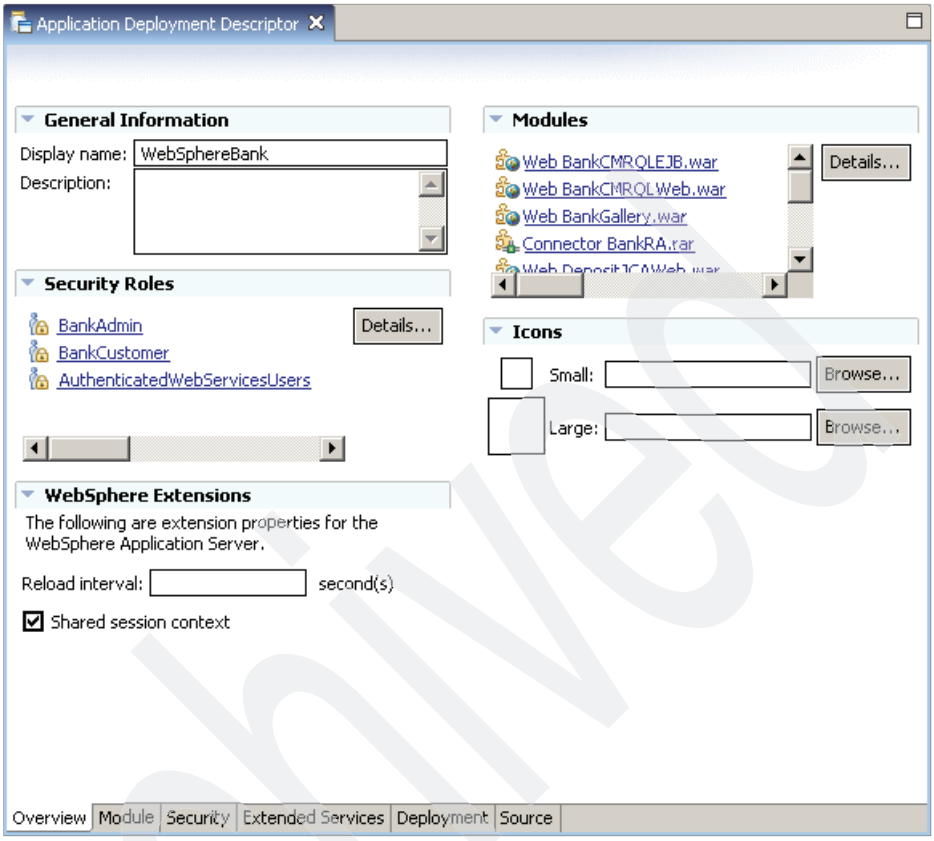


Figure 12-1 Shared HTTP session context using the Application Server Toolkit

5. Save and close the deployment descriptor.

12.5 Session identifiers

WebSphere session support keeps the user's session information about the server. WebSphere passes the user an identifier known as a session ID, which correlates an incoming user request with a session object maintained on the server.

Note: The example session IDs provided in this chapter are for illustrative purposes only and are *not* guaranteed to be absolutely consistent in value, format and length.

12.5.1 Choosing a session tracking mechanism

WebSphere supports three approaches to tracking sessions:

- SSL session identifiers
- Cookies
- URL rewriting

It is possible to select all three options for a Web application. If you do this:

- SSL session identifiers are used in preference to cookie and URL rewriting.
- Cookies are used in preference to URL rewriting.

Note: If SSL session ID tracking is selected, we recommended that you also select cookies or URL rewriting so that session affinity can be maintained. The cookie or rewritten URL contains session affinity information enabling the Web server to properly route a session back to the same server for each request.

To set or change the session mechanism type, do the following:

1. Open the session management properties for the application server, enterprise application, or Web module.
2. Select the session tracking mechanism that you require. See Figure 12-2 on page 704.

Archived

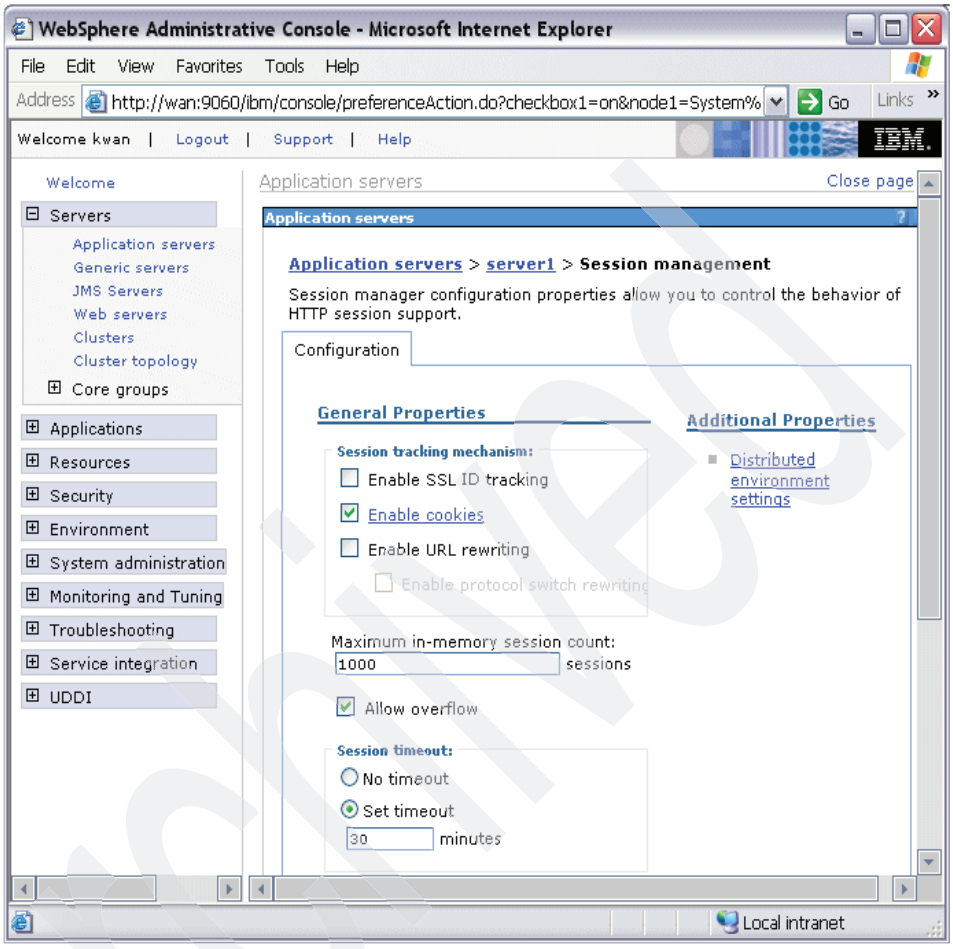


Figure 12-2 Selecting a session tracking mechanism window

3. Click **OK**.
4. Save and synchronize the configuration changes.
5. Restart the application server or the cluster.

12.5.2 SSL ID tracking

When SSL ID tracking is enabled for requests over SSL. SSL session information is used to track the HTTP session ID.

Arnied

Because the SSL session ID is negotiated between the Web browser and HTTP server, it cannot survive an HTTP server failure. However, the failure of an application server does not affect the SSL session ID and if the distributed session is not configured, the session itself is lost. In environments that use WebSphere Edge Server with multiple HTTP servers, you must use an affinity mechanism when SSL session ID is used as the session tracking mechanism.

Note: SSL tracking is supported only for the IBM HTTP Server and SUN ONE Web Server.

The lifetime of an SSL session ID can be controlled by configuration options in the Web server. For example, in the IBM HTTP Server, the configuration variable SSLV3TIMEOUT must be set to allow for an adequate lifetime for the SSL session ID. Too short an interval could result in premature termination of a session. Also, some Web browsers might have their own timers that affect the lifetime of the SSL session ID. These Web browsers might not leave the SSL session ID active long enough to be useful as a mechanism for session tracking.

When the SSL session ID is to be used as the session tracking mechanism in a clustered environment, either cookies or URL rewriting must be used to maintain session affinity. The cookie or rewritten URL contains session affinity information that enables the Web server to properly route requests back to the same server once the HTTP session has been created on a server. The SSL ID is not sent in the cookie or rewritten URL but is derived from the SSL information.

Disadvantages of SSL ID tracking

The main disadvantage of using SSL ID tracking is the performance hit of using SSL. If you have a business requirement to use SSL, then this would be a good choice. If you do not have such a requirement, it is probably a good idea to consider using cookies instead.

As discussed previously, Web server and Web browser SSL session timeout settings can also limit the usefulness of SSL ID tracking.

12.5.3 Cookies

Many sites choose cookie support to pass the user's identifier between WebSphere and the user. WebSphere Application Server session support generates a unique session ID for each user, and returns this ID to the user's browser with a cookie. The default name for the session management cookie is JSESSIONID. See Figure 12-3 on page 706.

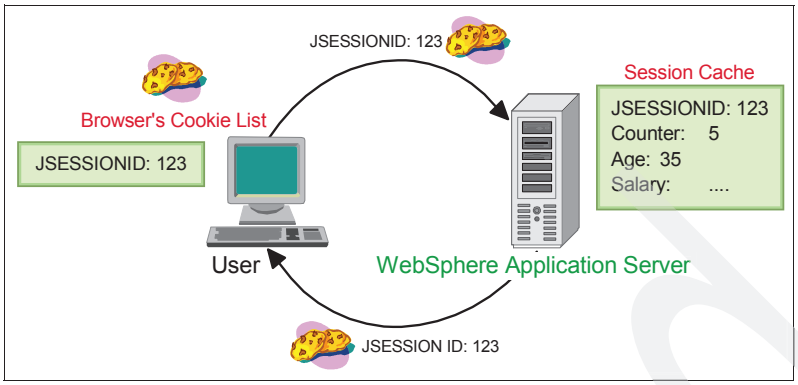


Figure 12-3 Cookie overview

A cookie consists of information embedded as part of the headers in the HTML stream passed between the server and the browser. The browser holds the cookie and returns it to the server whenever the user makes a subsequent request. By default, WebSphere defines its cookies so they are destroyed if the browser is closed. This cookie holds a session identifier. The remainder of the user's session information resides at the server.

The Web application developer uses the HTTP request object's standard interface to obtain the session:

```
HttpSession session = request.getSession(true);
```

WebSphere places the user's session identifier in the outbound cookie whenever the servlet completes its execution, and the HTML response stream returns to the end user. Again, neither the cookie nor the session ID within it require any direct manipulation by the Web application. The Web application only sees the contents of the session.

Cookie disadvantages

The main disadvantage with cookies is that some users, either by choice or mandate, disable them from within their browser.

Cookie settings

To configure session management using cookies, do the following from administrative console:

1. Open the Session Manager window at your preferred level.
2. Click the box for **Enable Cookies** as the session tracking mechanism. See Figure 12-4 on page 707.

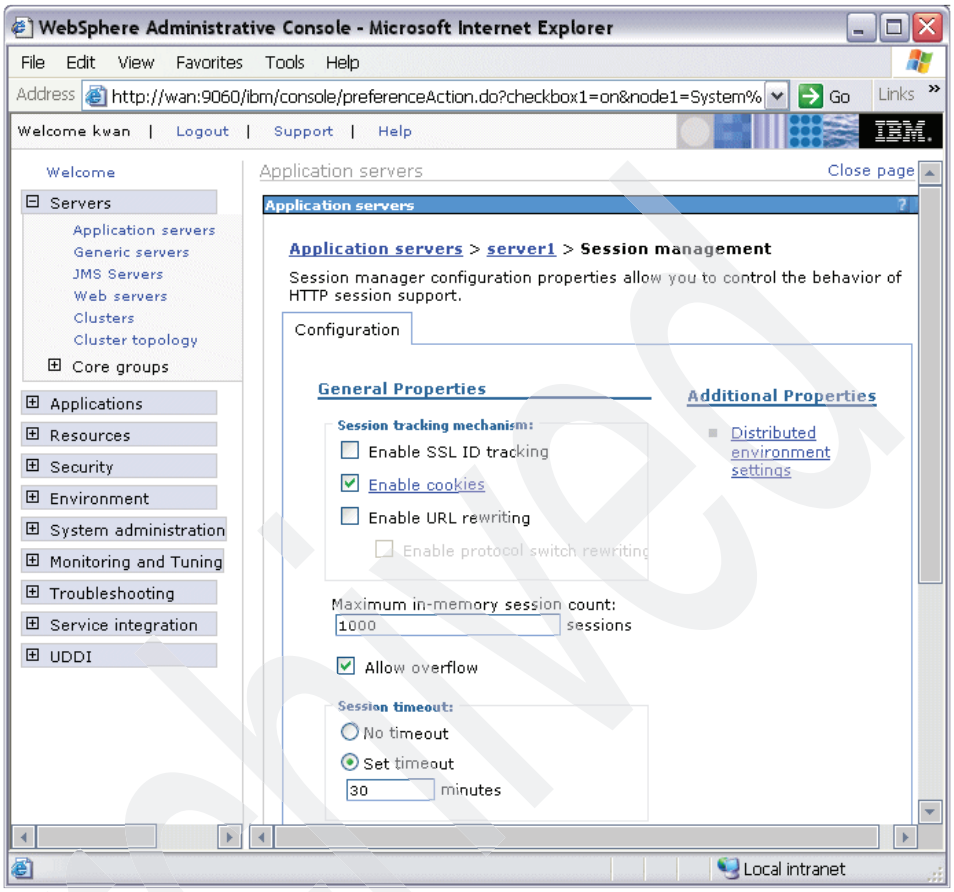


Figure 12-4 Session tracking mechanism

3. If you would like to view or change the cookies settings, select the **Enable Cookies** hot link. The following cookie settings are available:

   – **Cookie name**

     The cookie name for session management should be unique. The default cookie name is JSESSIONID, which is required by the Servlet 2.4 specification for all cookie-based session IDs. However, this value can be configured for flexibility.

   – **Restrict cookies to HTTPS sessions**

     Enabling this feature restricts the exchange of cookies only to HTTPS sessions. If it is enabled, the session cookie's body includes the secure indicator field.

Archived

– **Cookie domain**

This value dictates to the browser whether or not to send a cookie to particular servers. For example, if you specify a particular domain, the browser will only send back session cookies to hosts in that domain. The default value in the session manager restricts cookies to the host that sent them.

Note: The LTPA token/cookie that is sent back to the browser is scoped by a single DNS domain specified when global security is configured. This means that *all* application servers in an *entire* WebSphere Application Server domain must share the same DNS domain for security purposes.

– **Cookie path**

The paths on the server to which the browser will send the session tracking cookie. Specify any string representing a path on the server. Use the slash (/) to indicate the root directory.

Specifying a value restricts the paths to which the cookie will be sent. By restricting paths, you can keep the cookie from being sent to certain URLs on the server. If you specify the root directory, the cookie will be sent no matter which path on the given server is accessed.

– **Cookie maximum age**

The amount of time that the cookie will live in the client browser. There are two choices:

- Expire at the end of the current browser session
- Expire at a configurable maximum age

If you choose the maximum age option, specify the age in seconds.

4. Click **OK** to exit the page and change your settings.
5. Click **OK** to exit the session management settings.
6. Save and synchronize your configuration changes.
7. Restart the application server or the cluster.

For more information about cookie properties, see *Persistent Client State HTTP Cookies* at :

`http://home.netscape.com/newsref/std/cookie_spec.html`

12.5.4 URL rewriting

WebSphere also supports URL rewriting for session ID tracking. While session management using SSL IDs or cookies is transparent to the Web application, URL rewriting requires the developer to use special encoding APIs, and to set up the site page flow to avoid losing the encoded information.

URL rewriting works by storing the session identifier in the page returned to the user. WebSphere encodes the session identifier as a parameter on URLs that have been encoded programmatically by the Web application developer. This is an example of a Web page link with URL encoding:

```
<a href="/store/catalog;$jsessionid=DA32242SSGE2">
```

When the user clicks this link to move to the /store/catalog page, the session identifier passes into the request as a parameter.

URL rewriting requires explicit action by the Web application developer. If the servlet returns HTML directly to the requester, without using a JavaServer Page, the servlet calls the API, as shown in Example 12-1, to encode the returning content.

Example 12-1 URL encoding from a servlet

```
out.println("<a href=\");
out.println(response.encodeURL ("/store/catalog"));
out.println("\>catalog</a>");
```

Even pages using redirection, a common practice, particularly with servlet or JSP combinations, must encode the session ID as part of the redirect, as shown in Example 12-2.

Example 12-2 URL encoding with redirection

```
response.sendRedirect(response.encodeRedirectURL("http://myhost/store/catalog")
);
```

When JavaServer Pages (JSPs) use URL rewriting, the JSP calls a similar interface to encode the session ID:

```
<% response.encodeURL ("/store/catalog"); %>
```

Archive

URL rewriting configuration

URL rewriting is selected in the same way as cookies. The only additional configuration option is:

- **Enable protocol switch rewriting**

  This option defines whether the session ID, added to a URL as part of URL encoding, should be included in the new URL if a switch from HTTP to HTTPS or from HTTPS to HTTP is required. For example, if a servlet is accessed over HTTP and that servlet is doing encoding of HTTPS URLs, URL encoding will be performed only when protocol switch rewriting is enabled, and vice versa.

Disadvantages of using URL rewriting

The fact that the servlet or JSP developer has to write extra code is a major drawback over the other available session tracking mechanisms.

URL rewriting limits the flow of site pages exclusively to dynamically generated pages, such as pages generated by servlets or JSPs. WebSphere inserts the session ID into dynamic pages, but cannot insert the user's session ID into static pages, .htm or .html.

Therefore, after the application creates the user's session data, the user must visit dynamically generated pages exclusively until they finish with the portion of the site requiring sessions. URL rewriting forces the site designer to plan the user's flow in the site to avoid losing their session ID.

12.6 Local sessions

Many Web applications use the simplest form of session management: the in-memory, local session cache. The local session cache keeps session information in memory and local to the machine and WebSphere Application Server where the session information was first created.

Local session management does not share user session information with other clustered machines. Users only obtain their session information if they return to the machine and WebSphere Application Server holds their session information about subsequent accesses to the Web site.

Most importantly, local session management lacks a persistent store for the sessions it manages. A server failure takes down not only the WebSphere instances running on the server, but also destroys any sessions managed by those instances.

Archive

WebSphere allows the administrator to define a limit on the number of sessions held in the in-memory cache from the administrative console settings on the session manager. This prevents the sessions from acquiring too much memory in the Java VM associated with the application server.

The session manager also allows the administrator to permit an unlimited number of sessions in memory. If the administrator enables the **Allow overflow** setting on the session manager, the session manager permits two in-memory caches for session objects. The first cache contains only enough entries to accommodate the session limit defined to the session manager, 1000 by default. The second cache, known as the overflow cache, holds any sessions the first cache cannot accommodate, and is limited in size only by available memory. The session manager builds the first cache for optimized retrieval, while a regular, un-optimized hash table contains the overflow cache.

For best performance, define a primary cache of sufficient size to hold the normal working set of sessions for a given Web application server.

Important: If you enable overflow, the session manager permits an unlimited number of sessions in memory. Without limits, the session caches might consume all available memory in the WebSphere instance's heap, leaving no room to execute Web applications. For example, two scenarios under which this could occur are:

- The site receives greater traffic than anticipated, generating a large number of sessions held in memory.
- A malicious attack occurs against the site where a user deliberately manipulates their browser so the application creates a new session repeatedly for the same user.

If you choose to enable session overflow, the state of the session cache should be monitored closely.

Note: Each Web application will have its own base, or primary, in-memory session cache, and with overflow allowed, its own overflow, or secondary, in-memory session cache.

12.7 General properties for session management

The session management settings allow the administrator to tune a number of parameters that are important for both local or persistent sessions. See Figure 12-5 on page 712/

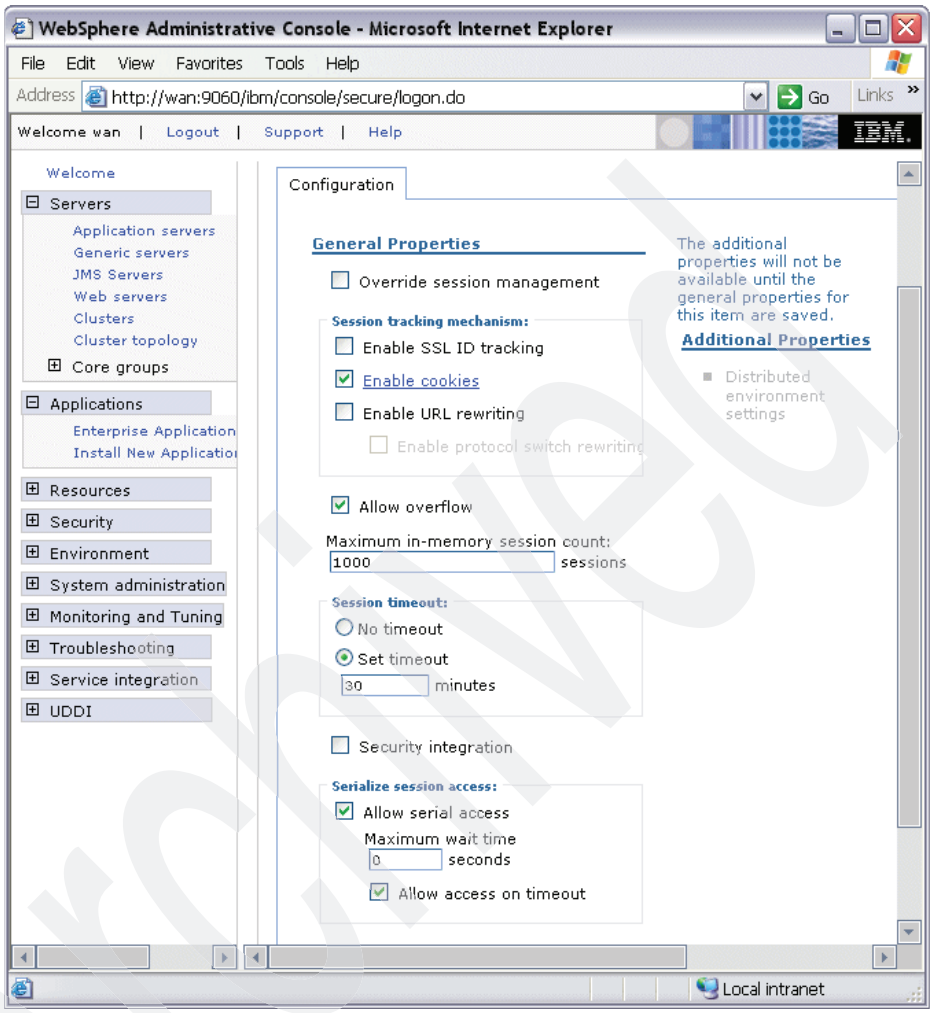


Figure 12-5 Session Management window 2

- Maximum in-memory session count

  This field specifies the maximum number of sessions to maintain in memory. The meaning differs depending on whether you are using local or persistent sessions. For local sessions, this value specifies the number of sessions in the base session table. Select **Overflow** to specify whether to limit sessions to this number for the entire session manager, or to allow additional sessions to be stored in secondary tables. Before setting this value, see 12.6, “Local sessions” on page 710.

Archived

For persistent sessions, this value specifies the size of the general cache. This value determines how many sessions will be cached before the session manager reverts to reading a session from the database automatically. Session manager uses a least recently used (LRU) algorithm to maintain the sessions in the cache.

This value holds when you use local sessions, persistent sessions with caching, or persistent sessions with manual updates. The manual update cache keeps the last n time stamps representing the last access times, where `n` is the maximum in-memory session count value.

- Allow overflow

  Choosing this option specifies whether to allow the number of sessions in memory to exceed the value specified in the maximum in-memory session count field. If **Allow overflow** is not checked, then WebSphere limits the number of sessions held in memory to this value.

  For local sessions, if this maximum is exceeded and Allow overflow is not checked, then sessions created thereafter will be dummy sessions and will not be stored in the session manager. Before setting this value, see 12.6, “Local sessions” on page 710.

  As shown in Example 12-3, the IBM HttpSession extension can be used to react if sessions exceed the maximum number of sessions specified when overflow is disabled.

Example 12-3 Using IBMSession to react to session overflow

```
com.ibm.websphere.servlet.session.IBMSession sess =
   (com.ibm.websphere.servlet.session.IBMSession) req.getSession(true);
if(sess.isOverFlow()) {
   //Direct to a error page…
}
```

Note: Allowing an unlimited amount of sessions can potentially exhaust system memory and even allow for system sabotage. Someone could write a malicious program that continually hits your site, creating sessions, but ignoring any cookies or encoded URLs and never utilizing the same session from one HTTP request to the next.

- Session timeout

  If you select **set timeout**, when a session is not accessed for this many minutes it can be removed from the in-memory cache and, if persistent sessions are used, from the persistent store. This is important for performance tuning. It directly influences the amount of memory consumed by the JVM in order to cache the session information.

Archived

Note: For performance reasons the session manager invalidation process runs at regular intervals to invalidate any invalid sessions. This interval is determined internally based on the Session timeout interval specified in the Session manager properties. For the default timeout value of 30 minutes, the invalidation process interval is around 300 seconds. In this case, it could take up to 5 minutes (300 seconds) beyond the timeout threshold of 30 minutes for a particular session to become invalidated.

The value of this setting is used as a default when the session timeout is not specified in a Web module's deployment descriptor.

If you select **no timeout**, a session will be never removed from the memory unless explicit invalidation has been performed by the servlet. This can cause memory leak when the user closes the window without logging out from the system. This option might be useful when sessions should be kept for a while until explicit invalidation has been done, when an employee leaves the company, for example. To use this option, make sure that enough memory or space in a persistent store is kept to accommodate all sessions.

- Security integration

  When security integration is enabled, the session manager associates the identity of users with their HTTP sessions. See 12.11, "Session security" on page 751 for more information.

  Note: Do not enable this property if the application server contains a Web application that has form-based login configured as the authentication method and the local operating system is the authentication mechanism. It will cause authorization failures when users try to use the Web application.

- Serialize session access

  In WebSphere V4, sessions could be accessed concurrently, meaning multiple threads could access the same session at the same time. It was the programmer's responsibility to serialize access to the session to avoid inconsistencies.

  In WebSphere V5 and WebSphere V6, this option is available to provide serialized access to the session in a given JVM. This ensures thread-safe access when the session is accessed by multiple threads. No special code is necessary for using this option. This option is not recommended when framesets are used heavily because it can affect performance.

  An optional property, **Maximum wait time,** can be set to specify the maximum amount of time a servlet request waits on an HTTP session before continuing execution. The default is two minutes.

If you set the **Allow access on timeout** option, multiple servlet requests that have timed out concurrently will execute normally. If it is false, servlet execution aborts.

12.8 Session affinity

The Servlet 2.4 specification requires that an HTTP session be:

- Accessible only to the Web application that created the session

  The session ID, but not the session data, can be shared across Web applications.
- Handled by a single JVM for that application at any one time

In a clustered environment, any HTTP requests associated with an HTTP session must be routed to the same Web application in the same JVM. This ensures that all of the HTTP requests are processed with a consistent view of the user's HTTP session. The exception to this rule is when the cluster member fails or has to be shut down.

WebSphere is able to assure that session affinity is maintained in the following way: Each server ID is appended to the session ID. When an HTTP session is created, its ID is passed back to the browser as part of a cookie or URL encoding. When the browser makes further requests, the cookie or URL encoding will be sent back to the Web server. The Web server plug-in examines the HTTP session ID in the cookie or URL encoding, extracts the unique ID of the cluster member handling the session, and forwards the request.

This can be seen in Figure 12-6 on page 716, where the session ID from the HTTP header, request.getHeader("Cookie"), is displayed along with the session ID from session.getId(). The application server ID is appended to the session ID from the HTTP header. The first four characters of HTTP header session ID are the cache identifier that determines the validity of cache entries.

Archived

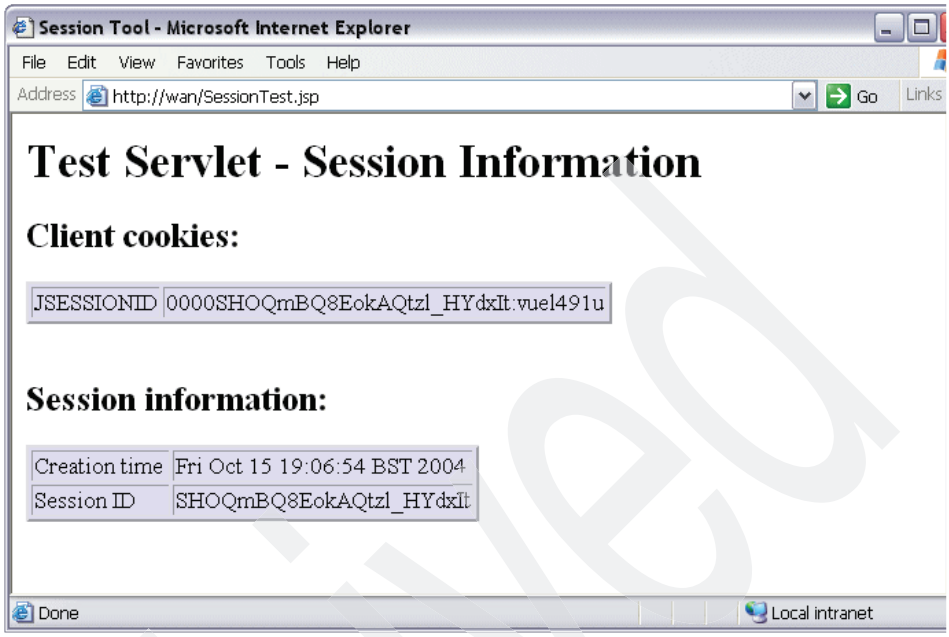


Figure 12-6 Session ID containing the server ID and cache ID

The JSESSIONID cookie can be divided into four parts: cache ID, session ID, separator, clone ID, and partition ID (new in V6). JSESSION ID will include a partition ID instead of a clone ID when memory-to-memory replication in peer-to-peer mode is selected. Typically the partition ID is a long numeric number.

Table 12-1 shows their mappings based on the example in Figure 12-6. A clone ID is an ID of a cluster member.

Table 12-1 Cookie mapping

content	value in the example
Cache ID	0000
Session ID	SHOQmBQ8EokAQtzl_HYdxIt
separator	:
Clone ID	vuel491u

Archived

The application server ID can be seen in the Web server plug-in configuration file, plug-in-cfg.xml file, as shown in Example 12-4.

Example 12-4 Server ID from plugin-cfg.xml file

```
<?xml version="1.0" encoding="ISO-8859-1"?><!--HTTP server plugin config file
for the cell ITSOCell generated on 2004.10.15 at 07:21:03 PM BST-->
<Config>
......
   <ServerCluster Name="MyCluster">
      <Server CloneID="vue1491u" LoadBalanceWeight="2" Name="NodeA_server1">
        <Transport Hostname="wan" Port="9080" Protocol="http"/>
        <Transport Hostname="wan" Port="9443" Protocol="https">
    ......
</Config>
```

Note: Session affinity can still be broken if the cluster member handling the request fails. To avoid losing session data, use persistent session management. In persistent sessions mode, cache ID and server ID will change in the cookie when there is a failover or when the session is read from the persistent store. So don't rely on the value of the session cookie remaining the same for a given session.

12.8.1 Session affinity and failover

Server clusters provide a solution for failure of an application server. Sessions created by cluster members in the server cluster share a common persistent session store. Therefore, any cluster member in the server cluster has the ability to see any user's session saved to persistent storage. If one of the cluster members fail, the user can continue to use session information from another cluster member in the server cluster. This is known as failover. Failover works regardless of whether the nodes reside on the same machine or several machines. See Figure 12-7 on page 718.

Archive

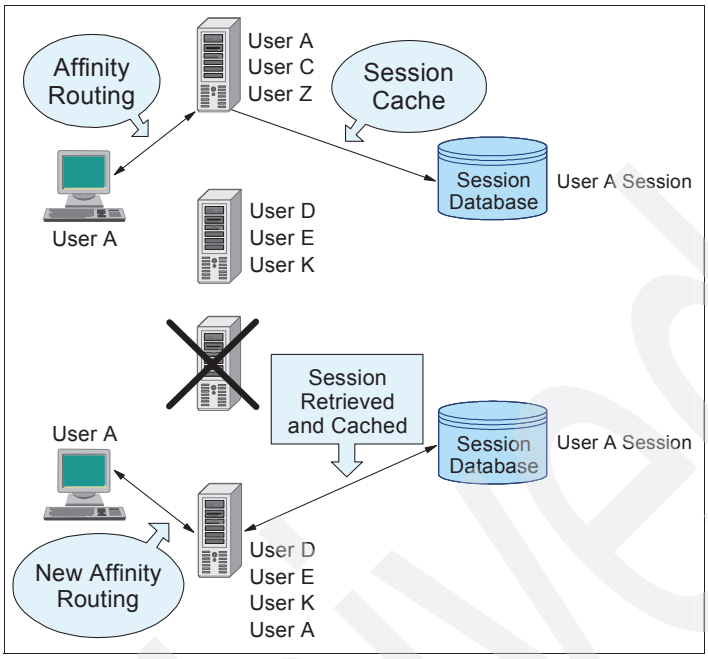


Figure 12-7 Session affinity and failover

Note: According to the Servlet 2.4 specification, only a single cluster member can control and access a given session at a time.

After a failure, WebSphere redirects the user to another cluster member, and the user's session affinity switches to this replacement cluster member. After the initial read from the persistent store, the replacement cluster member places the user's session object in the in-memory cache, assuming the cache has space available for additional entries.

The Web server plug-in maintains the cluster member list in order and picks the cluster member next in its list to avoid the breaking of session affinity. From then on, requests for that session go to the selected cluster member. The requests for the session go back to the failed cluster member when the failed cluster member restarts.

WebSphere provides session affinity on a best-effort basis. There are narrow windows where session affinity fails. These windows are:

- When a cluster member is recovering from a crash, a window exists where concurrent requests for the same session could end up in different cluster members. The reason for this is the Web server is multi-processed and each

process separately maintains its own retry timer value and list of available cluster members. The end result is that requests being processed by different processes might end up being sent to more than one cluster member after at least one process has determined that the failed cluster member is running again.

To avoid or limit exposure in this scenario, if your cluster members are expected to crash very seldom and are expected to recover fairly quickly, consider setting the retry timeout to a small value. This narrowed the window during which multiple requests being handled by different processes get routed to multiple cluster members.

- A server overload can cause requests belonging to the same session to go to different cluster members. This can occur even if all the cluster members are running. For each cluster member, there is a backlog queue where an entry is made for each request sent by the Web server plug-in waiting to be picked up by a worker thread in the servlet engine. If the depth of this queue is exceeded, the Web server plug-in starts receiving responses that the cluster member is not available. This failure is handled in the same way by the Web server plug-in as an actual JVM crash. Examples of when this can happen are:
  - The servlet engine does not have an appropriate number of threads to handle the user load.
  - The servlet engine threads take a long time to process the requests. Reasons for this include: applications taking a long time to execute, resources being used by applications taking a long time, and so on.

12.9 Persistent session management

By default, WebSphere places session objects in memory. However, the administrator has the option of enabling persistent session management, which instructs WebSphere to place session objects in a persistent store. Administrators should enable persistent session management when:

- The user's session data must be recovered by another cluster member after a cluster member in a cluster fails or is shut down.
- The user's session data is too valuable to lose through unexpected failure at the WebSphere node.
- The administrator desires better control of the session cache memory footprint. By sending cache overflow to a persistent session store, the administrator controls the number of sessions allowed in memory at any given time.

There are two ways to configure session persistence in WebSphere Application Server V6, as in Figure 12-8:

- Database persistence
- Memory-to-memory session state replication using the data replication service available in distributed server environments

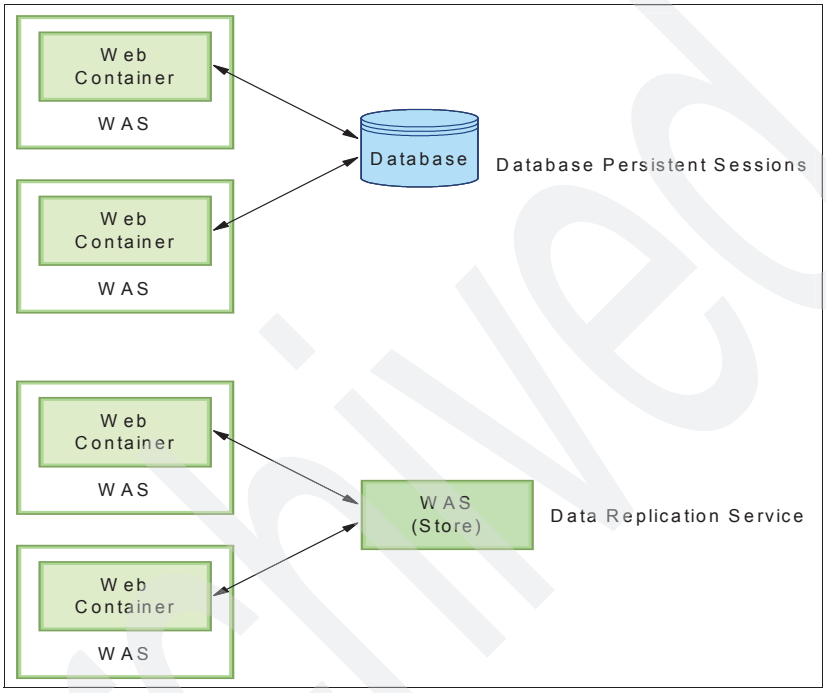


Figure 12-8 Persistent session options

All information stored in a persistent session store must be serialized. As a result, all of the objects held by a session must implement java.io.Serializable if the session needs to be stored in a persistent session store.

In general, consider making all objects held by a session serialized, even if immediate plans do not call for the use of persistent session management. If the Web site grows, and persistent session management becomes necessary, the transition between local and persistent management occurs transparently to the application if the sessions only hold serialized objects. If not, a switch to persistent session management requires coding changes to make the session contents serialized.

Archived

Persistent session management does not impact the session API, and Web applications require no API changes to support persistent session management. However, as mentioned previously, applications storing unserializable objects in their sessions require modification before switching to persistent session management.

If you use database persistence, using multi-row sessions becomes important if the size of the session object exceeds the size for a row, as permitted by the WebSphere session manager. If the administrator requests multi-row session support, the WebSphere session manager breaks the session data across multiple rows as needed. This allows WebSphere to support large session objects. Also, this provides a more efficient mechanism for storing and retrieving session contents under certain circumstances. See 12.9.6, “Single and multi-row schemas (database persistence)” on page 743 for information about this feature.

Using a cache lets the session manager maintain a cache of most recently used sessions in memory. Retrieving a user session from the cache eliminates a more expensive retrieval from the persistent store. The session manager uses a *least recently used* scheme for removing objects from the cache. Session data is stored to the persistent store based on your selections for write frequency and write option.

12.9.1 Enabling database persistence

It is assumed in this section that the following tasks have already completed before enabling database persistence:

1. Create a session database. In this example, it is assumed that the data source JNDI name is `jdbc/Sessions`.
2. Create a JDBC provider and data source. See 7.2, “JDBC resources” on page 321 and 7.2.3, “Creating a data source” on page 326.

Note: The following example illustrates the steps to enable database persistence at the application server level. Session management settings can also be performed at the enterprise application level and the Web application level.

To enable database persistence, repeat the following steps for each application server:

1. Click **Servers** →**Application servers**
2. Select the server.
3. Click **Session management** under **Web container** in the Additional Properties section.

4. Click **Distributed environment settings.**
5. Select **Database** and click **Database**. See Figure 12-9.

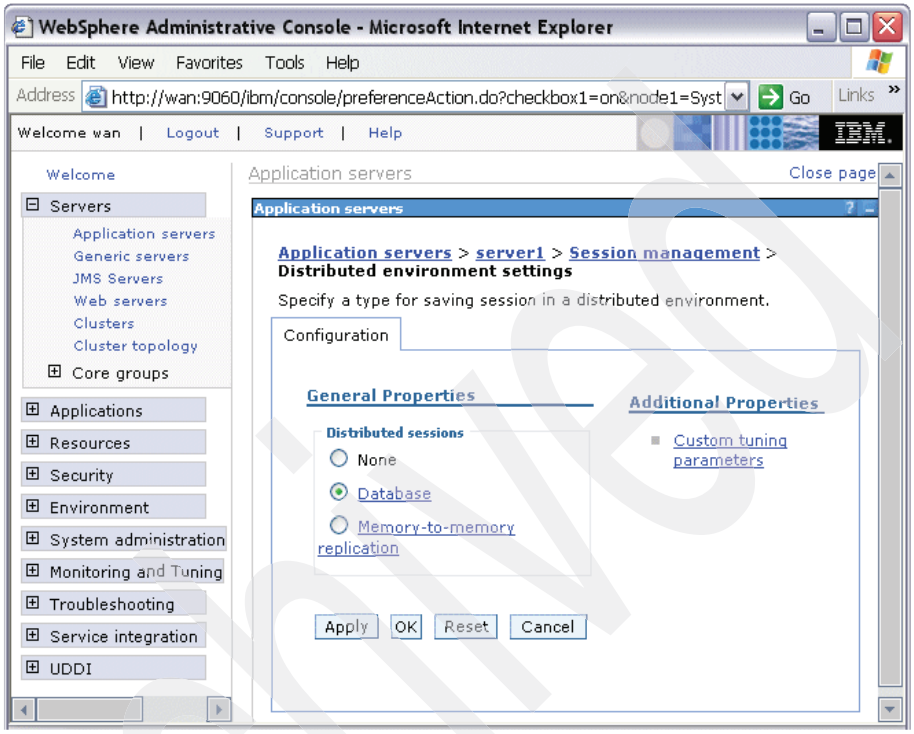


Figure 12-9 Distributed Environment Setting (database)

6. Enter the database information:
   a. Enter the data source JNDI name. The data source must be a non-JTA enabled data source.
   b. Enter the user ID and password to access the database.
   c. If you are using DB2 and you anticipate requiring row sizes greater than 4 KB, select the appropriate value from the DB2 row size menu. See 12.9.5, “Larger DB2 page sizes and database persistence” on page 742 for more information.
   d. If DB2 row size is other than 4 KB, you are required to enter the name of tablespace. See “Larger DB2 page sizes and database persistence” on page 742.
   e. If you intend to use a multi-row schema, select **Use Multi row schema**. See 12.9.6, “Single and multi-row schemas (database persistence)” on page 743 for more information. See Figure 12-10.

Archive

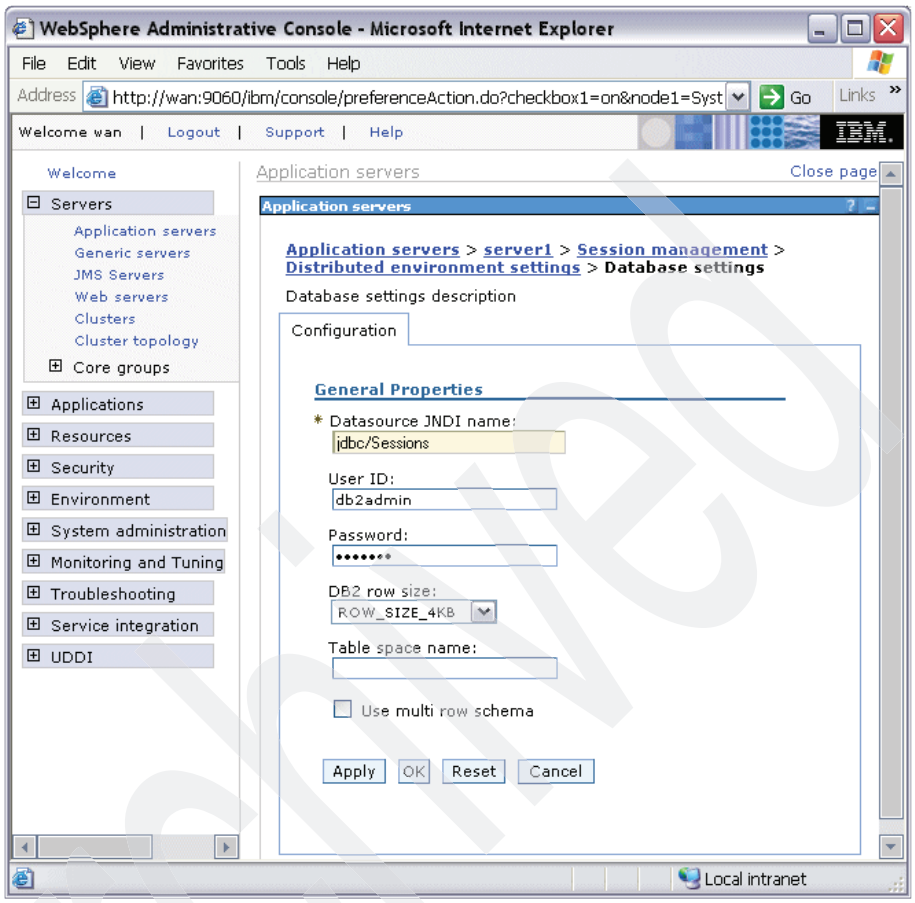


Figure 12-10 Database

7. Click **OK**.

After you have updated each server, save the configuration changes, synchronize them with the servers, and restart the application servers.

12.9.2 Memory-to-memory replication

Memory-to-memory replication uses the data replication service to replicate data across many application servers in a cluster without using a database. Using this method, sessions are stored in the memory of an application server, providing the same functionality as a database for session persistence. Separate threads handle this functionality within an existing application server process.

Archived

The data replication service is an internal WebSphere Application Server component. In addition to its use by the session manager, it is also used to replicate dynamic cache data and stateful session beans across many application servers in a cluster.

The advantages of using this method of session persistence are:

- Flexible configuration options such as peer-peer and client/server
- Elimination of the overhead and cost of setting up and maintaining a real-time production database.
- Elimination of single point of failure that can occur with a database.
- Encrypted session information between application servers.

Version 5.X versus version 6 data replication service

The following changes have been made to the Data Replication Service in WebSphere Application Server V6:

- Simplified configuration

  Many fields from V5.X have been deprecated and the configuration panels are now more intuitive.

- Terminology changes

  Some of the terms from V5.X have been deprecated to reflect the new topologies and configuration needs. The terms *replicas* and *partitions* have been removed. In V6 we have *client* and *servers* in a replication domain.

- Topology changes

  Partitions from V5.X have been deprecated. Replication domain still exists, but defined differently. It is no longer a collection of replicas.You can select the replication mode of server, client, or both when configuring the session management facility for memory-to-memory replication in WebSphere Application Server V6. The default is both.

- Integration with workload management to provide hot failover in peer-to-peer mode

- Ability to collocate stateful session EJB replicas with HTTP session replicas in hot failover

V5.X wsadmin DRS scripts continue to work with V6.

Archived

Data replication service modes

The memory-to-memory replication function is accomplished by the creation of a data replication service instance in an application server that communicates to other data replication service instances in remote application servers.

There are three possible modes you can set up a replication service instance to run in:

- Server mode

  In this mode a server only stores backup copies of other application server sessions. It does not send copies of sessions created in that particular server.

- Client mode

  In this mode, a server only broadcasts or sends copies of the sessions it owns. It does not receive backup copies of sessions from other servers

- Both mode

  In this mode, the server simultaneously sends copies of the sessions it owns, and acts as a backup table for sessions owned by other application servers.

You can select the replication mode of server, client, or both when configuring the session management facility for memory-to-memory replication. The default is both.

With respect to mode, the following are the primary examples of memory-to-memory replication configuration:

- Peer-to-peer replication
- Client/server replication

Although the administrative console allows flexibility and additional possibilities for memory-to-memory replication configuration, only these configurations are officially supported.

There is a single replica in a cluster by default. You can modify the number of replicas through the replication domain.

Peer-to-peer topology

Figure 12-11 on page 726 shows an example of peer-to-peer topology. Each application server stores sessions in its own memory. It also stores sessions to and retrieves sessions from other application servers. Each application server acts as a client by retrieving sessions from other application servers. Each application server acts as a server by providing sessions to other application servers.

Archived

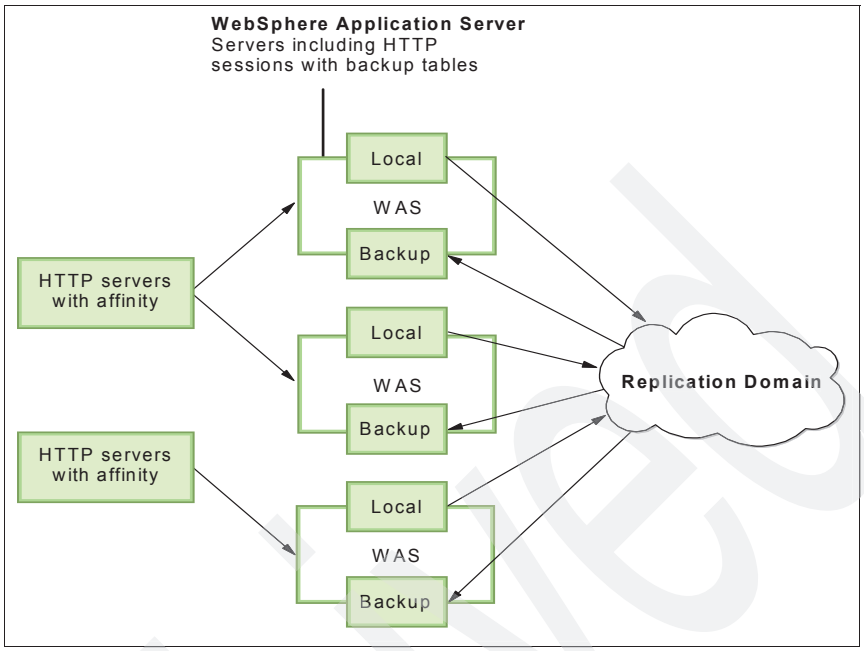


Figure 12-11 Example of peer-to-peer topology

The basic peer-to-peer, both mode, topology is the default configuration and has a single replica. However, you can also add additional replicas by configuring the replication domain.

In this basic peer-to-peer topology, each application server can:

- Host the Web application leveraging the HTTP session
- Send changes to the HTTP session that it owns
- Receive backup copies of the HTTP session from all of the other servers in the cluster

This configuration represents the most consolidated topology, where the various system parts are collocated and requires the fewest server processes. When using this configuration, the most stable implementation is achieved when each node has equal capabilities (CPU, memory, and so on), and each handles the same amount of work.

The advantage of this topology is that no additional processes and products are required to avoid a single point of failure. This reduces the time and cost required to configure and maintain additional processes or products.

One of the disadvantages of this topology is that it can consume large amounts of memory in networks with many users, because each server has a copy of all

sessions. For example, assuming that a single session consumes 10 KB and one million users have logged into the system, each application server consumes 10 GB of memory in order to keep all sessions in its own memory. Another disadvantage is that every change to a session must be replicated to all application servers. This can cause a performance impact.

Client/server topology

Figure 12-12 on page 727 shows an example of client/server topology. In this setup, application servers act as either a replication client or a server. Those that act as replication servers store sessions in their own memory and provide session information to clients. They are dedicated replication servers that just store sessions but do not respond to the users' requests. Client application servers send session information to the replication servers and retrieve sessions from the servers. They respond to user requests and store only the sessions of the users with whom they interact.

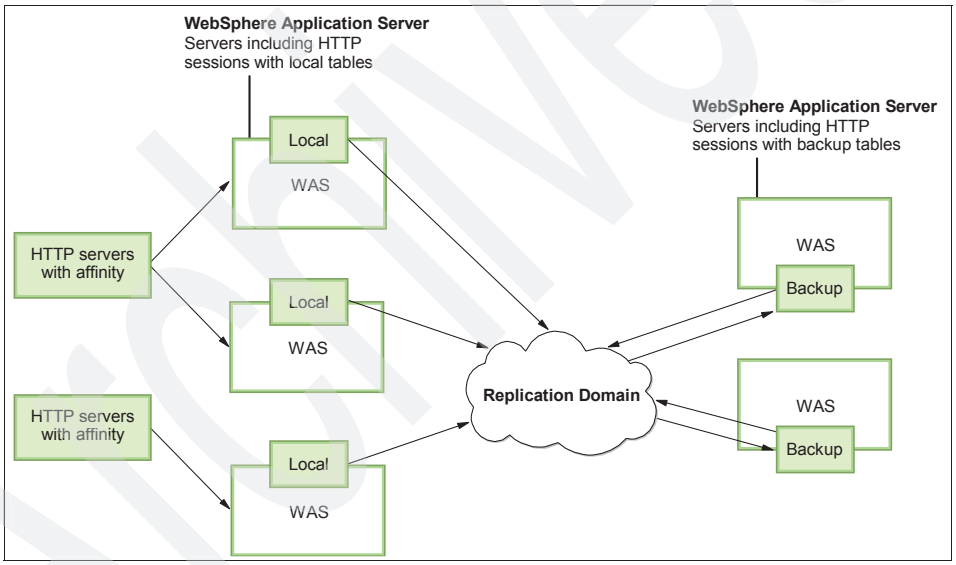


Figure 12-12 Example of client/server topology

The advantage of this topology is that it clearly distinguishes the role of client and server. Only replication servers keep all sessions in their memory and only the clients interact with users. This reduces the consumption of memory on each application server and reduces the performance impact, because session information is only sent to the servers.

Archive

You can recycle a backup server without affecting the servers running the application, When there are two or more backups, failure recovery is possible. Conversely, you can recycle an application server without losing the backup data.

When running Web applications on lower-end hardware, you can choose to have one or two more powerful computers that have the capacity to run a couple of session managers in replication server mode; allowing you to reduce the load on the Web application hardware.

One of the disadvantages of this topology is that additional application servers have to be configured and maintained over and above those that interact with users. We recommended that you have multiple replication servers configured to avoid a single point of failure.

Replication domain

The memory-to-memory replication function is accomplished by the creation of a data replication service instance in an application server that communicates to other data replication service instances in remote application servers. You must configure this data replication service instance as a part of a replication domain.

Data replication service instances on disparate application servers that replicate to one another must be configured as a part of the same domain. You must configure all session managers connected to a replication domain to have the same topology. If one session manager instance in a domain is configured to use the client/server topology, then the rest of the session manager instances in that domain must be a combination of servers configured as Client only and Server only.

If one session manager instance is configured to use the peer-to-peer topology, then all session manager instances must be configured as both client and server. For example, a server only data replication service instance and a both client and server data replication service instance cannot exist in the same replication domain. Multiple data replication service instances that exist on the same application server due to session manager memory-to-memory configuration at various levels that are configured to be part of the same domain must have the same mode.

WebSphere Application Server V6 encourages the creation of a separate replication domain for each consumer. For example, create one replication domain for session manager and another replication domain for dynamic cache.

Archived

The only situation where you should configure one replication domain is when you configure session manager replication and stateful session bean failover. Using one replication domain in this case ensures that the backup state information of HTTP sessions and stateful session beans are on the same application servers.

Note: A replication domain created with WebSphere Application Server V5.X is referred to as a *multi-broker domain*. This type of replication domain consists of replicator entries. This is deprecated in WebSphere Application Server V6 and supported only for backward compatibility. Multi-broker replication domains do not communicate with each other, so migrate any multi-broker replication domains to the new data replication domains. You cannot create a multi-broker domain or replicator entries in the administrative console of WebSphere Application Server V6.

Enabling memory-to-memory replication

It is assumed in this section that the following tasks have already been completed before enabling data the replication service:

1. You have created a cluster consisting of at least two application servers.

   In this example, we are working with a cluster called MyCluster. It has two servers, server1 and server2.

2. You have installed applications to the cluster.

Note: This example illustrates setting up the replication domain and replicators after the cluster has been created. You also have the option of creating the replication domain and the replicator the first server in the cluster when you create the cluster.

To enable memory-to-memory replication, do the following:

1. Create a replication domain to define the set of replicator processes that communicate with each other.

   a. Select **Environment** →**Replication domains.** Click **New**. See Figure 12-13 on page 730, and enter information in the fields

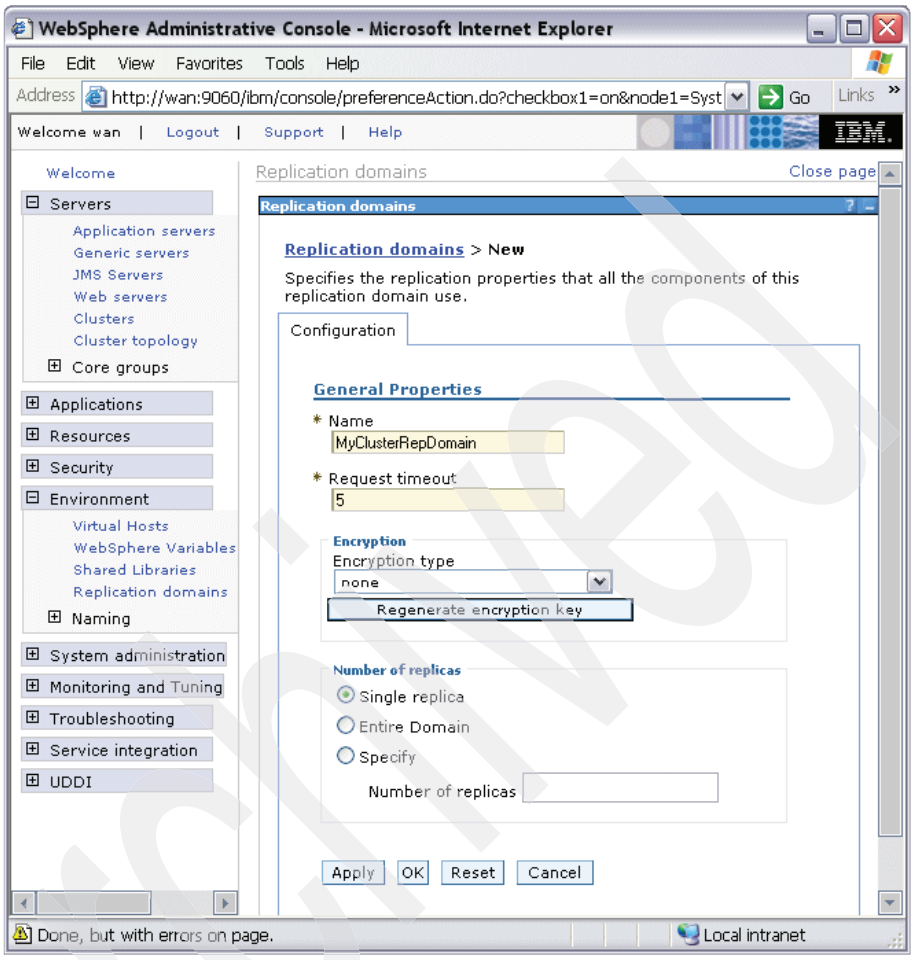


Figure 12-13 Create a replication domain

– **Name**

At a minimum, you need to enter a name for the replication domain. The name must be unique within the cell. In this example, we used `MyClusterRepDomain` as name and defaults are used as other properties.

– **Encryption**

Encrypted transmission achieves better security but can impact performance. If DES or TRIPLE_DES is specified, a key for data transmission is generated. We recommend that you generate a key by clicking the **Regenerate encryption key** button periodically to enhance security.

Archived

– **Number of replicas**

A single replica allows you to replicate a session to only one other server. This is the default. When you choose this option, a session manager picks another session manager connected to the same replication domain to which to replicate the HTTP session during session creation. All updates to the session are only replicated to that single server. This option is set at the replication domain level. When this option is set, every session manager connected to this replication domain creates a single backup copy of HTTP session state information about a backup server.

Alternatively, you can replicate to every application server that is configured as a consumer of the replication domain with the **Entire Domain** option or to a specified number of replicas within the domain.

b. Click **Apply**.

c. Click **OK.**

d. Save the configuration changes.

2. Configure cluster members.

Repeat the following steps for each application server:

a. Select **Servers** →**Application servers.**

b. Click the application server name. In this example, **server1** and **server2** are selected as application servers respectively.

c. Click **Web container** in the Container settings section.

d. Click **Session management**.

e. Click **Distributed environment settings**.

f. Select **Memory-to-memory replication**. See Figure 12-14 on page 732.

Archive

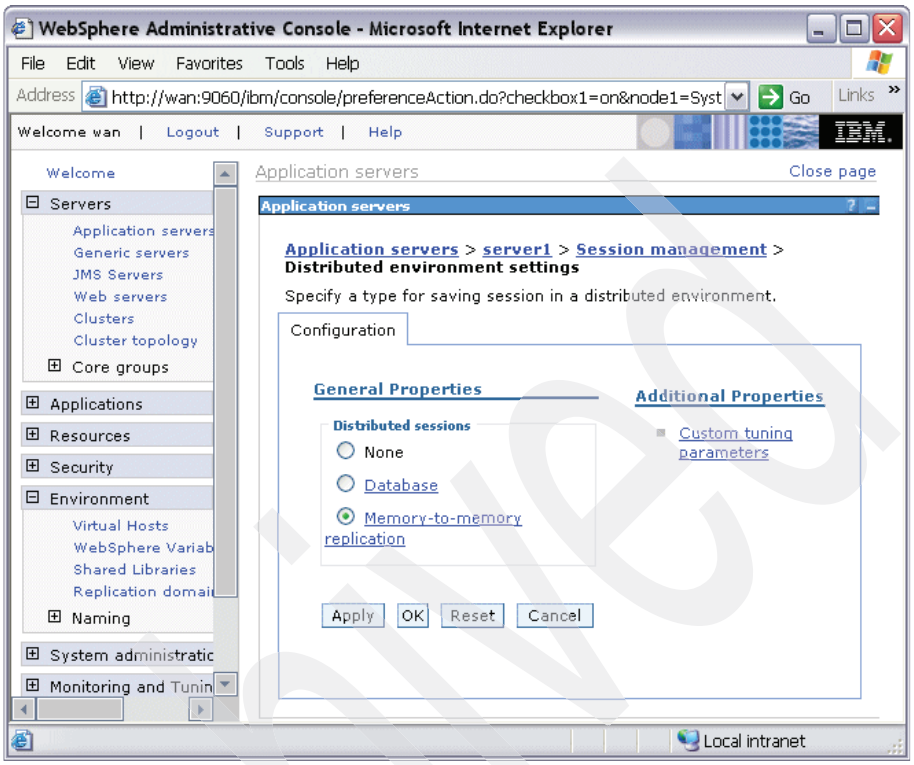


Figure 12-14 Distributed environment settings

g. Choose a replicator domain and replicator mode either from listed domains. See Figure 12-15 on page 733.

Archived

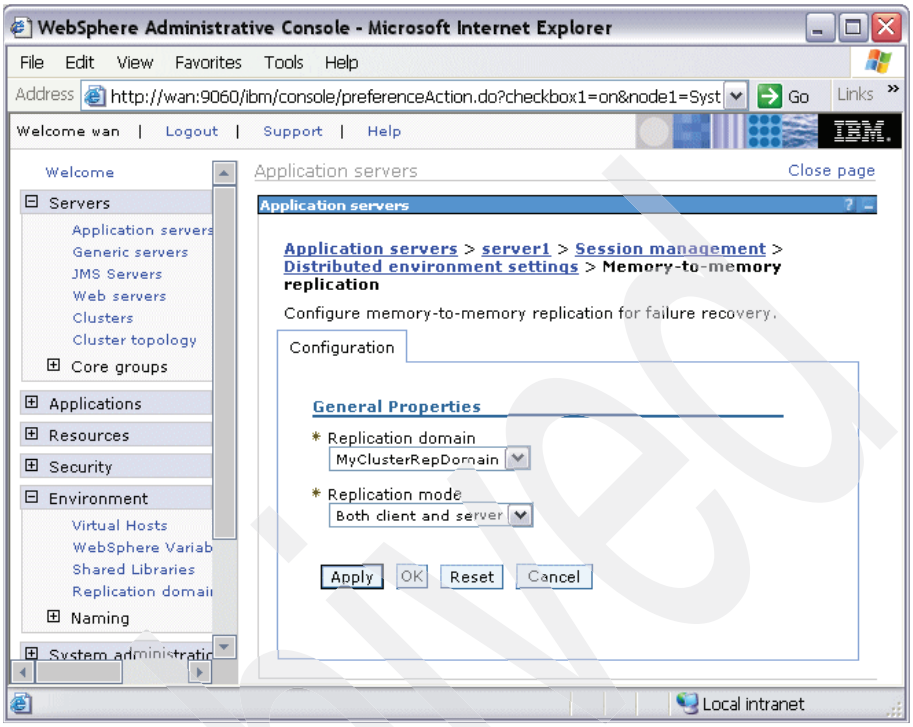


Figure 12-15 Data replication service settings

Select the replication topology by specifying the replication mode. Selecting **Both client and server** identifies this as a peer-to-peer topology. In a client/server topology, select **Client only** for application servers that will be responding to user requests. Select **Server only** for those that will be used as replication servers.

h. Click **OK**.

3. Save the configuration and restart the cluster. You can restart the cluster by selecting **Servers** →**Clusters**. Check the cluster, and click **Stop**. After the messages indicate the cluster has stopped, click **Start**.

Configuration file results

The replication domain configuration is written to

`<profile_home>/config/cells/<cell>/ multibroker.xml`,

See Example 12-5 on page 734.

Example 12-5 Replication domain configuration in multibroker.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<multibroker:DataReplicationDomain xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:multibroker="http://www.ibm.com/websphere/appserver/schemas/5.0/multibrok
er.xmi" xmi:id="DataReplicationDomain_1098124985651" name="MyClusterRepDomain">
  <defaultDataReplicationSettings xmi:id="DataReplication_1098124985651"
requestTimeout="5" encryptionType="NONE" numberOfReplicas="1">
    <partition xmi:id="DRSPartition_1098124985651" size="10"
partitionOnEntry="false"/>
    <serialization xmi:id="DRSSerialization_1098124985651"
entrySerializationKind="BYTES" propertySerializationKind="BYTES"/>
    <pooling xmi:id="DRSConnectionPool_1098124985651" size="10"
poolConnections="false"/>
  </defaultDataReplicationSettings>
</multibroker:DataReplicationDomain>
```

Configuring an application server to use a replication domain for session persistence updates the server.xml file:

<*profile_home*>/config/cells/<*cell*>/nodes/<*node*>/servers/<*server*>/ server.xml

See Example 12-6.

Example 12-6 Server.xml updates

```
<sessionDRSPersistence xmi:id="DRSSettings_1097867741921"
messageBrokerDomainName="MyClusterRepDomain"/>
```

12.9.3 Session management tuning

Performance tuning for session management persistence consists of defining the following:

- How often session data is written (write frequency settings)
- How much data is written (write contents settings)
- When the invalid sessions are cleaned up (session cleanup settings)

These settings are set in the Custom tuning parameters found under the Additional properties section for session management settings. Several combinations of these settings are predefined and available for selection, or you can customize them.

Archive

Writing frequency settings

You can select from three different settings that determine how often session data is written to the persistent data store:

- End of servlet service

  If the session data has changed, it will be written to the persistent store after the servlet finishes processing an HTTP request.

- Manual update

  The session data will be written to the persistent store when the sync() method is called on the IBMSession object.

- Time-based

  The session data will be written to the persistent store based on the specified write interval value.

Note: The last access time attribute is always updated each time the session is accessed by the servlet or JSP, whether the session is changed or not. This is done to make sure the session does not time out.

- If you choose the end of servlet service option, each servlet or JSP access will result in a corresponding persistent store update of the last access time.
- If you select the manual update option, the update of the last access time in persistent store occurs on sync() call or at later time.
- If you use time-based updates, the changes are accumulated and written in a single transaction. This can significantly reduce the amount of I/O to the persistent store.

See 12.12.2, “Reducing persistent store I/O” on page 756 for options to change this database update behavior.

Consider an example where the Web browser accesses the application once every five seconds:

- In End of servlet service mode, the session would be written out every five seconds.
- In Manual update mode, the session would be written out whenever the servlet issues IBMSession.sync(). It is the responsibility of the servlet writer to use the IBMSession interface instead of the HttpSession Interface and the servlets/JSPs must be updated to issue the sync().
- In Time-based mode, the servlet or JSP need not use the IBMSession class nor issue IBMSession.sync(). If the write interval is set to 120 seconds, then the session data is written out at most every 120 seconds.

End of servlet service

When the write frequency is set to the end of servlet service option, WebSphere writes the session data to the persistent store at the completion of the HttpServlet.service() method call. The write content settings determine output.

Manual update

In manual update mode, the session manager only sends changes to the persistent data store if the application explicitly requests a save of the session information.

Note: Manual updates use an IBM extension to HttpSession that is not part of the Servlet 2.4 API.

Manual update mode requires an application developer to use the IBMSession class for managing sessions. When the application invokes the sync() method, the session manager writes the modified session data and last access time to the persistent store. The session data written to the persistent store is controlled by the write contents option selected.

If the servlet or JSP terminates without invoking the sync() method, the session manager saves the contents of the session object into the session cache (if caching is enabled), but does not update the modified session data in the session database. The session manager will only update the last access time in the persistent store asynchronously, at later time. Example 12-7 shows how the IBMSession class can be used to manually update the persistent store.

Example 12-7 Using IBMSession for manual update of the persistent store

```
public void service (HttpServletRequest req, HttpServletResponse res)
   throws ServletException, IOException
{
   // Use the IBMSession to hold the session information
   // We need the IBMSession object because it has the manual update
   // method sync()
   com.ibm.websphere.servlet.session.IBMSession session =
      (com.ibm.websphere.servlet.session.IBMSession)req.getSession(true);

   Integer value = 1;

   //Update the in-memory session stored in the cache
   session.putValue("MyManualCount.COUNTER", value);

   //The servlet saves the session to the persistent store
   session.sync();
}
```

This interface gives the Web application developer additional control of when and if session objects go to the persistent data store. If the application does not invoke the sync() method, and manual update mode is specified, the session updates go only to the local session cache, not the persistent data store. Web developers use this interface to reduce unnecessary writes to the session database, and thereby to improve overall application performance.

All servlets in the Web application server must perform their own session management in manual update mode.

Time-based writes to the session database

Using the time-based write option will write session data to the persistent store at a defined write interval. The reasons for implementing time-based write lies in the changes introduced with the Servlet 2.2 API. The Servlet 2.2 specification introduced two key concepts:

- It limits the scope of a session to a single Web application.
- It both explicitly prohibits concurrent access to an HttpSession from separate Web applications, and allows for concurrent access within a given JVM.

Because of these changes, WebSphere provides the session affinity mechanism that assures an HTTP request is routed to the Web application handling its HttpSession. This assurance still holds in a WLM environment when using persistent HttpSessions. This means that the necessity to immediately write the session data to the persistent store can now be relaxed somewhat in these environments, as well as non-clustered environments, because the persistent store is used now only for failover and session cache full scenarios.

With this in mind, it is now possible to gain potential performance improvements by reducing the frequency of persistent store writes.

Note: Time-based writes requires session affinity for session data integrity.

The following details apply to time-based writes:

- The expiration of the write interval does not necessitate a write to the persistent store unless the session has been touched, getAttribute/setAttribute/removeAttribute was called, since the last write.
- If a session write interval has expired and the session has only been retrieved, request.getSession() was called since the last write, then the last access time will be written to the persistent store regardless of the write contents setting.
- If a session write interval has expired and the session properties have been either accessed or modified since the last write, then the session properties

will be written in addition to the last access time. Which session properties get written is dependent on the write contents settings.

- Time-based write allows the servlet or JSP to issue IBMSession.sync() to force the write of session data to the database.
- If the time between session servlet requests for a particular session is greater than the write interval, then the session effectively gets written after each service method invocation.
- The session cache should be large enough to hold all of the active sessions. Failure to do this will result in extra persistent store writes, because the receipt of a new session request can result in writing out the oldest cached session to the persistent store. To put it another way, if the session manager has to remove the least recently used HttpSession from the cache during a full cache scenario, the session manager will write that HttpSession using the Write contents settings upon removal from the cache.
- The session invalidation time must be at least twice the write interval to ensure that a session does not inadvertently get invalidated prior to getting written to the persistent store.
- A newly created session will always be written to the persistent store at the end of the service method.

Writing content settings

These options control what is written. See 12.9.7, “Contents written to the persistent store using a database” on page 745 before selecting one of the options, because there are several factors to decide. The options available are:

- **Only update attributes** are written to the persistent store.
- **All session attributes** are written to the persistent store.

Session cleanup settings

WebSphere allows the administrator to defer to off hours the clearing of invalidated sessions from the persistent store. *Invalidated sessions* are sessions that are no longer in use and timed out. For more information, see 12.10, “Invalidating sessions” on page 749. This can be done either once or twice a day. The fields available are:

- **First time of day (0-23)** is the first hour during which the invalidated persistent sessions will be cleared from the persistent store. This value must be a positive integer between 0 and 23.
- **Second time of day (0-23)** is the second hour during which the invalidated persistent sessions will be cleared from the persistent store. This value must be a positive integer between 0 and 23.
- Select **Schedule sessions cleanup** to enable this option.

Also consider using schedule invalidation for intranet-style applications that have a somewhat fixed number of users wanting the same HTTP session for the whole business day.

Configuration

The session management tuning parameters can be set by selecting a predefined tuning level or by specifying each parameter. To specify the performance settings for session management, do the following:

1. Select **Servers** →**Application Servers** and click the application server.

   Note: Remember, session management options can also be set at the enterprise application level (see “Application session management properties” on page 700) or at the Web module level (see “Web module session management properties” on page 700).

2. Expand the **Web Container Settings** and click **Web container.**
3. Click **Session management**.
4. Click **Distributed environment settings**.
5. Select from the predefined tuning levels or click **Custom tuning parameters**.

   See Figure 12-16 on page 740.

Archived

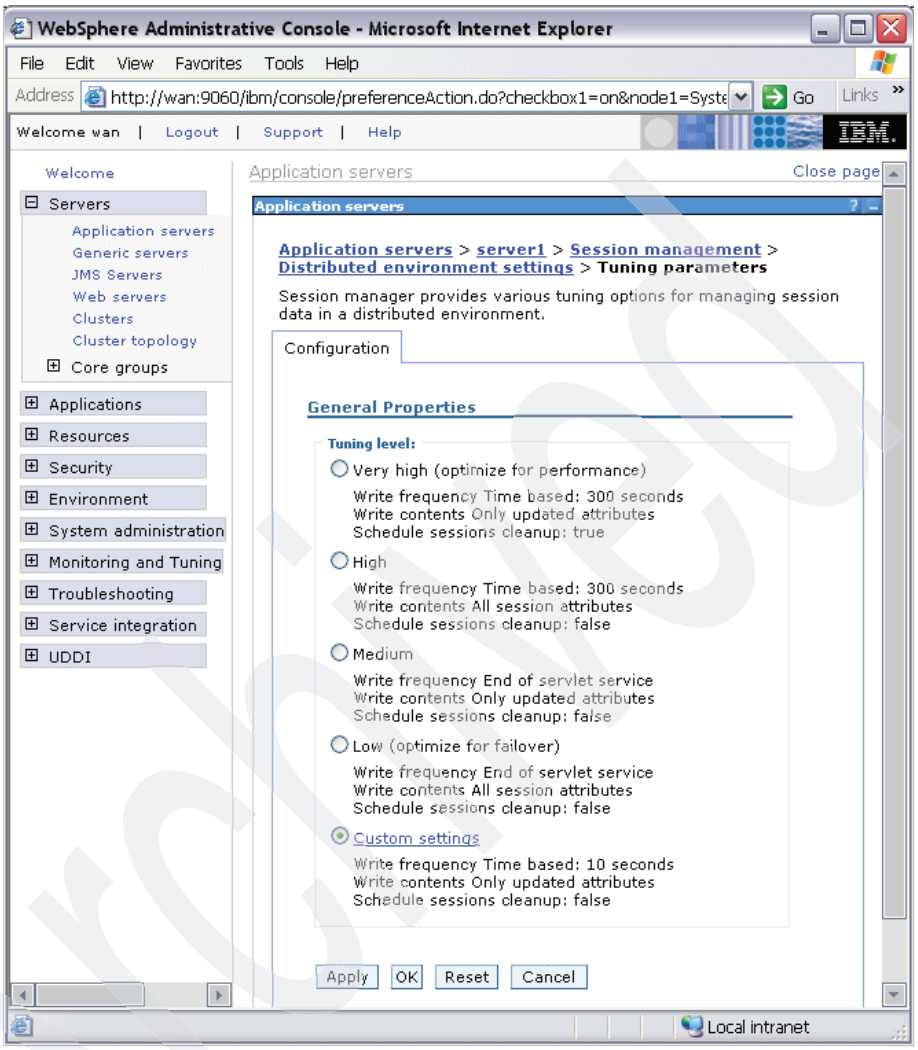


Figure 12-16 Session management tuning parameters

If you want to set each tuning parameter explicitly, select **Custom settings**. See Figure 12-17 on page 741.

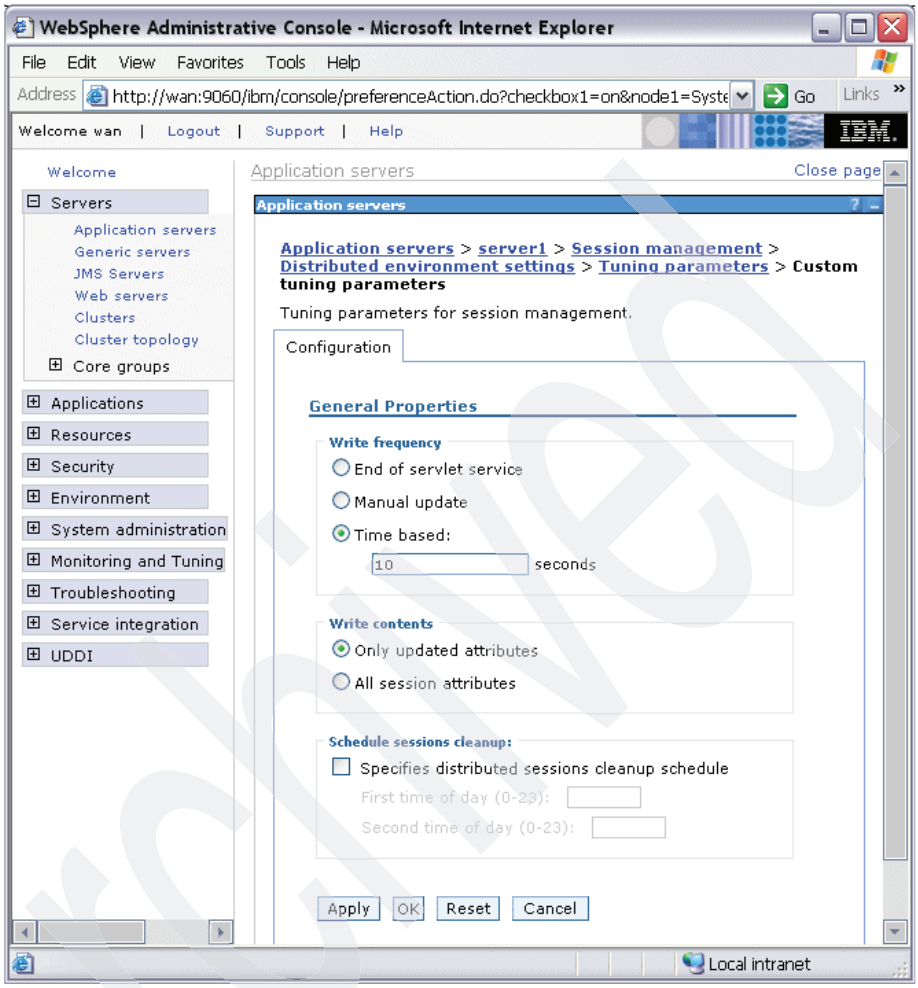


Figure 12-17 Session management tuning parameters

12.9.4 Persistent sessions and non-serializable J2EE objects

In order for the WebSphere session manager to persist a session to the persistent store, all of the Java objects in an HttpSession must be serializable. They must implement the java.io.Serializable interface. The HttpSession can also contain the following J2EE objects, which are not serializable:

- javax.ejb.EJBObject
- javax.ejb.EJBHome
- javax.naming.Context
- javax.transaction.UserTransaction

Archived

The WebSphere session manager works around the problem of serializing these objects in the following manner:

- EJBObject and EJBHome each have Handle and HomeHandle object attributes that are serializable and can be used to reconstruct the EJBObject and EJBHome.
- Context is constructed with a hash table based environment, which is serializable. WebSphere will retrieve the environment, then wrap it with an internal, serializable object. On reentry, it can check the object type and reconstruct the Context.
- UserTransaction has no serializable attributes. WebSphere provides two options:

  a. The Web developer can place the object in the HttpSession, but WebSphere will not persist it outside the JVM.

  b. WebSphere has a new public wrapper object, com.ibm.websphere.servlet.session.UserTransactionWrapper, which is serializable and requires the InitialContext used to construct the UserTransaction. This will be persisted outside the JVM and be used to reconstruct the UserTransaction.

Note: According to J2EE, a Web component can only start a transaction in a service method. A transaction that is started by a servlet or JSP must be completed before the service method returns. That is, transactions cannot span Web requests from a client. If there is an active transaction after returning from the service method, WebSphere will detect it and abort the transaction.

In general, Web developers should consider making all other Java objects held by HttpSession serializable, even if immediate plans do not call for the use of persistent session management. If the Web site grows, and persistent session management becomes necessary, the transition between local and persistent management occurs transparently to the application if the sessions hold only serializable objects. If not, a switch to persistent session management requires coding changes to make the session contents serializable.

12.9.5 Larger DB2 page sizes and database persistence

WebSphere supports 4 KB, 8 KB, 16 KB, and 32 KB page sizes, and can have larger varchar for bit data columns of about 7 KB, 15 KB, or 31 KB. Using this performance feature, we see faster persistence for HttpSession of sizes of 7 KB to 31 KB in the single-row case, or attribute sizes of 4 KB to 31 KB in the multi-row case.

Enabling this feature involves dropping any existing table created with a 4 KB buffer pool and tablespace. This also applies if you subsequently change between 4 KB, 8 KB, 16 KB, or 32 KB.

To use a page size other than the default 4 KB, do the following:

1. If the SESSIONS table already exists, drop it from the DB2 database:

```
DB2 connect to session
DB2 drop table sessions
```

2. Create a new DB2 buffer pool and tablespace, specifying the same page size (8 KB, 16 KB or 32 KB) for both, and assign the new buffer pool to this tablespace. Example 12-8 shows simple steps for creating an 8 KB page:

Example 12-8 Creating an 8K page size

```
DB2 connect to session
DB2 CREATE BUFFERPOOL sessionBP SIZE 1000 PAGESIZE 8K
DB2 connect reset
DB2 connect to session
DB2 CREATE TABLESPACE sessionTS PAGESIZE 8K MANAGED BY SYSTEM USING
('D:\DB2\NODE0000\SQL00005\sessionTS.0') BUFFERPOOL sessionBP
DB2 connect reset
```

   Refer to the DB2 product documentation for details.

3. Configure the correct tablespace name and page size, DB2 row size, in the session management database configuration. See Figure 12-10 on page 723.

Restart WebSphere. On startup, the session manager creates a new SESSIONS table based on the page size and tablespace name specified.

12.9.6 Single and multi-row schemas (database persistence)

When using the single-row schema, each user session maps to a single database row. This is WebSphere's default configuration for persistent session management. With this setup, there are hard limits to the amount of user-defined, application-specific data that WebSphere Application Server can access.

When using the multi-row schema, each user session maps to multiple database rows, with each session attribute mapping to a database row.

In addition to allowing larger session records, using a multi-row schema can yield performance benefits, as discussed in 12.12.3, "Multirow persistent sessions: Database persistence" on page 757.

Archived

Switching from single-row to multi-row schema

To switch from single-row to multi-row schema for sessions, do the following:

1. Modify the session manager properties to switch from single to multi-row schema. Select the **Use multi row schema** on the Database setting of the Session Manager window, shown in Figure 12-10 on page 723.
2. Manually drop the database table or delete all the rows in the session database table. To drop the table:
   a. Determine which data source the session manager is using. This is set in the session management distributed settings window. See 12.9.1, “Enabling database persistence” on page 721.
   b. Look up the database name in the data source settings. Find the JDBC provider, then the data source. The database name is in the custom settings.
   c. Use the database facilities to connect to the database and drop it.
3. Restart the application server or cluster.

Design considerations

Consider configuring direct, single-row usage to one database and multi-row usage to another database while you verify which option suits your application's specific needs. You can do this by switching the data source used, then monitoring the performance. Table 12-2 provides an overview.

Table 12-2 Single versus multi-row schemas

Programming issue	Application scenario
Reasons to use single-row	► You can read/write all values with just one record read/write. ► This takes up less space in a database, because you are guaranteed that each session is only one record long.
Reasons *not* to use single-row	There is a 2 MB limit of stored data per session. The sum of sizes of all session attributes is limited to 2 MB.

Archived

Programming issue	Application scenario
Reasons to use multi-row	► The application can store an unlimited amount of data. You are limited only by the size of the database and a 2 MB-per-record limit. The size of each session attribute can be 2 MB. ► The application can read individual fields instead of the whole record. When large amounts of data are stored in the session but only small amounts are specifically accessed during a given servlet's processing of an HTTP request, multi-row sessions can improve performance by avoiding unneeded Java object serialization.
Reasons *not* to use multi-row	If data is small in size, you probably do not want the extra overhead of multiple row reads when everything could be stored in one row.

In the case of multi-row usage, design your application data objects so they do not have references to each other. This is to prevent circular references.

For example, suppose you are storing two objects A and B in the session using HttpSession.put(..), and A contains a reference to B. In the multi-row case, because objects are stored in different rows of the database, when objects A and B are retrieved later, the object graph between A and B is different from that stored. A and B behave as independent objects.

12.9.7 Contents written to the persistent store using a database

WebSphere supports two modes for writing session contents to the persistent store:

- Only updated attributes

  Write only the HttpSession properties that have been updated via setAttribute() and removeAttribute().

- All session attributes

  Write all the HttpSession properties to the database.

When a new session is initially created with either of the above two options, the entire session is written, including any Java objects bound to the session. When using database persistence, the behavior for subsequent servlet or JSP requests for this session varies depending on whether the single-row or multi-row database mode is in use.

- In single-row mode, choose from the following:
  - Only updated attributes

    If any session attribute has been updated, through setAttribute or removeAttribute, then all of the objects bound to the session will be written to the database.
  - All session attributes

    All bound session attributes will be written to the database.
- In multi-row mode:
  - Only updated attributes

    Only the session attributes that were specified via setAttribute or removeAttribute will be written to the database.
  - All session attributes

    All of the session attributes that reside in the cache will be written to the database. If the session has never left the cache, then this should contain all of the session attributes.

By using the All session attributes mode, servlets and JSPs can change Java objects that are attributes of the HttpSession without having to call setAttribute() on the HttpSession for that Java object in order for the changes to be reflected in the database.

Using the All session attributes mode provides some flexibility to the application programmer and protects against possible side effects of moving from local sessions to persistent sessions.

However, using All session attributes mode can potentially increase activity and be a performance drain. Individual customers will have to evaluate the pros and cons for their installation. It should be noted that the combination of All session attributes mode with time-based write could greatly reduce the performance penalty and essentially give you the best of both worlds.

As shown in Example 12-9 and Example 12-10, the initial session creation contains a setAttribute, but subsequent requests for that session do not need to use setAttribute.

Archived

Example 12-9 Initial servlet

```
HttpSession sess = request.getSession(true);
myClass myObject = new myClass();
myObject.someInt = 1;
sess.setAttribute("myObject", myObject);   // Bind object to the session
```

Example 12-10 Subsequent servlet

```
HttpSession sess = request.getSession(false);
myObject =  sess.getAttribute("myObject");   // get bound session object
myObject.someInt++;  // change the session object
// setAttribute() not needed with write "All session attributes" specified
```

Example 12-11 and Example 12-12 show setAttribute is still required even though the write all session attributes option is enabled.

Example 12-11 Initial servlet

```
HttpSession sess = request.getSession(true);
String myString = new String("Initial Binding of Session Object");
sess.setAttribute("myString", myString);   // Bind object to the session
```

Example 12-12 Subsequent servlet

```
HttpSession sess = request.getSession(false);
String myString  = sess.getAttribute("myString"); // get bound session object
...
myString = new String("A totally new String");  // get a new String object
sess.setAttribute("myString", myString);   // Need to bind the object to the
session since a NEW Object is used
```

HttpSession set/getAttribute action summary

Table 12-3 summarizes the action of the HttpSession setAttribute and removeAttribute methods for various combinations of the row type, write contents, and write frequency session persistence options.

Table 12-3 Write contents versus write frequency

Row type	Write contents	Write frequency	Action for setAttribute	Action for removeAttribute
Single-row	Only updated attributes	End of servlet service / sync() call with Manual update	If any of the session data has changed, then write all of this session's data from cache[1]	If any of the session data has changed, then write all of this session's data from cache[1]

Row type	Write contents	Write frequency	Action for setAttribute	Action for removeAttribute
Single-row	Only updated attributes	Time-based	If any of the session data has changed, then write all of this session's data from cache[1]	If any of the session data has changed, then write all of this session's data from cache[1]
	All session attributes	End of servlet service / sync() call with Manual update	Always write all of this session's data from cache[2]	Always write all of this session's data from cache[2]
		Time-based	Always write all of this session's data from cache	Always write all of this session's data from cache
Multi-row	Only updated attributes	End of servlet service / sync() call with Manual update	Write only thread-specific data that has changed	Delete only thread-specific data that has been removed
		Time-based	Write thread-specific data that has changed for *all* threads using this session	Delete thread-specific data that has been removed for *all* threads using this session
	All session attributes	End of servlet service / sync() call with Manual update	Write all session data from cache	Delete thread-specific data that has been removed for *all* threads using this session
		Time-based	Write all session data from cache	Delete thread-specific data that has been removed for *all* threads using this session

[1] When a session is written to the database while using single-row mode, *all* of the session data is written. Therefore, no database deletes are necessary for properties removed with removeAttribute(), because the write of the entire session does not include removed properties.

Multi-row mode has the notion of thread-specific data. *Thread-specific data* is defined as session data that was added or removed while executing under this thread. If you use End of servlet service or Manual update modes and enable Only updated attributes, then only the thread-specific data is written to the database.

12.10 Invalidating sessions

This section discusses invalidating sessions when the user no longer needs the session object. for example, when the user has logged off a site. Invalidating a session removes it from the session cache, as well as from the persistent store.

WebSphere offers three methods for invalidation session objects:

- Programmatically, you can use the invalidate() method on the session object. If the session object is accessed by multiple threads in a Web application, be sure that none of the threads still have references to the session object.
- An invalidator thread scans for timed-out sessions every `n` seconds, where `n` is configurable from the administrative console. The session timeout setting is in the general properties of the session management settings.
- For persistent sessions, the administrator can specify times when the scan runs. This feature has the following benefits when used with persistent session:
  - Persistent store scans can be scheduled during periods that normally have low demand. This avoids slowing down online applications due to contention in the persistent store.
  - When this setting is used with the End of servlet service write frequency option, WebSphere does not have to write the last access time with every HTTP request. The reason is, WebSphere does not have to synchronize the invalidator thread's deletion with the HTTP request access.

  You can find the session cleanup schedule setting in the session management settings under the custom tuning properties for distributed environments.

  If you are going to use session cleanup, be aware of the following:
  - HttpSession timeouts are not enforced. Instead, all invalidation processing is handled at the configured invalidation times.
  - With listeners, described in 12.10.1, “Session listeners”, processing is potentially delayed by this configuration. It is not recommended if listeners are used.

12.10.1 Session listeners

Some listener classes are defined in the Servlet 2.4 specification to listen for state changes of a session and its attributes. This allows greater control over interactions with sessions, leading programmers to monitor creation, deletion, and modification of sessions. Programmers can perform initialization tasks when a session is created, or clean up tasks when a session is removed. It is also

possible to perform some specific tasks for the attribute when an attribute is added, deleted or modified.

The following are the Listener interfaces to monitor the events associated with the HttpSession object:

- javax.servlet.http.HttpSessionListener

  Use this interface to monitor creation and deletion, including session timeout, of a session.

- javax.servlet.http.HttpSessionAttributeListener

  Use this interface to monitor changes of session attributes, such as add, delete, and replace.

- javax.servlet.http.HttpSessionActivationListener

  This interface monitors activation and passivation of sessions. This interface is useful to monitor if the session exists, whether on memory or not, when persistent session is used.

Table 12-4 is a summary of the interfaces and methods.

Table 12-4 Listener interfaces and their methods

Target	Event	Interface	Method
session	create	HttpSessionListener	sessionCreated()
	destroy	HttpSessionListener	sessionDestroyed()
	activate	HttpSessionActivationListener	sessionDidActivate()
	passivate	HttpSessionActivationListener	sessionWillPassivate()
attribute	add	HttpSessionAttributeListener	attributeAdded()
	remove	HttpSessionAttributeListener	attributeRemoved()
	replace	HttpSessionAttributeListener	attributeReplaced()

For more information, see *Java 2 Platform Enterprise Edition, v 1.4 API Specification* at:

`http://java.sun.com/j2ee/1.4/docs/api/index.html`

Archived

12.11 Session security

WebSphere Application Server maintains the security of individual sessions. When session manager integration with WebSphere security is enabled, the session manager checks the user ID of the HTTP request against the user ID of the session held within WebSphere. This check is done as part of the processing of the request.getSession() function. If the check fails, WebSphere throws an com.ibm.websphere.servlet.session.UnauthorizedSessionRequestException exception. If it succeeds, the session data is returned to the calling servlet or JSP.

Session security checking works with the standard HttpSession. The identity or user name of a session can be accessed through the com.ibm.websphere.servlet.session.IBMSession interface. An unauthenticated identity is denoted by the user name *anonymous*.

The session manager uses WebSphere's security infrastructure to determine the authenticated identity associated with a client HTTP request that either retrieves or creates a session. For information about WebSphere security features, see *WebSphere Application Security V6 Security Handbook,* SG24-6316.

Security integration rules for HTTP sessions

Session management security has the following rules:

- Sessions in unsecured pages are treated as accesses by the anonymous user.
- Sessions created in unsecured pages are created under the identity of that anonymous user.
- Sessions in secured pages are treated as accesses by the authenticated user.
- Sessions created in secured pages are created under the identity of the authenticated user. They can only be accessed in other secured pages by the same user. To protect these sessions from use by unauthorized users, they cannot be accessed from an unsecure page. Do not mix access to secure and unsecure pages.
- Security integration in session manager is not supported in HTTP form-based login with SWAM (Simple WebSphere Authentication Mechanism).

Archived

Table 12-5 lists possible scenarios when security integration is enabled, where outcomes depend on whether the HTTP request was authenticated and whether a valid session ID and user name was passed to the session manager.

Table 12-5 HTTP session security

Request session ID/ user name	**Unauthenticated HTTP request is used to retrieve the session**	**Authenticated HTTP request is used to retrieve the session. The user ID in the HTTP request is FRED.**
No session ID was passed in for this request, or the ID is for a session that is no longer valid.	A new session is created. The user name is anonymous.	A new session is created. The user name is FRED.
A valid session ID is received. The current session user name is anonymous.	The session is returned.	The session is returned. The session manager changes the user name to FRED.
A valid session ID is received. The current session user name is FRED.	The session is not returned. UnauthorizedSession-RequestException is thrown[1].	The session is returned.
A valid session ID is received. The current session user name is BOB.	The session is not returned. UnauthorizedSession-RequestException is thrown[1].	The session is not returned. UnauthorizedSession-RequestException is thrown[1].
[1] com.ibm.websphere.servlet.session.UnauthorizedSessionRequestException is thrown to the servlet or JSP.		

See 12.7, “General properties for session management” on page 711 for more information about the security integration setting.

12.12 Session performance considerations

This section includes guidance for developing and administering scalable, high-performance Web applications using WebSphere Application Server session support.

12.12.1 Session size

Large session objects pose several problems for a Web application. If the site uses session caching, large sessions reduce the memory available in the WebSphere instance for other tasks, such as application execution.

For example, assume a given application stores 1 MB of information for each user session object. If 100 users arrive over the course of 30 minutes, and assume the session timeout remains at 30 minutes, the application server instance must allocate 100 MB just to accommodate the newly arrived users in the session cache:

```
1 MB for each user session * 100 users = 100 MB
```

Note this number does not include previously allocated sessions that have not timed out yet. The memory required by the session cache could be considerably higher than 100 MB.

Web developers and administrators have several options for improving the performance of session management:

- Reduce the size of the session object.
- Reduce the size of the session cache.
- Add additional application servers.
- Invalidate unneeded sessions.
- Increase the memory available.
- Reduce the session timeout interval.

Reducing session object size

Web developers must consider carefully the information kept by the session object:

- Removing information easily obtained or easily derived helps keep the session object small.
- Rigorous removal of unnecessary, unneeded, or obsolete data from the session.
- Consider whether it would be better to keep a certain piece of data in an application database rather than in the HTTP session. This gives the developer full control over when the data is fetched or stored and how it is combined with other application data. Web developers can leverage the power of SQL if the data is in an application database.

Archived

Reducing object size becomes particularly important when persistent sessions are used. Serializing a large amount of data and writing it to the persistent store requires significant WebSphere performance overhead. Even if the Write contents option is enabled, if the session object contains large Java objects or collections of objects that are updated regularly, there is a significant performance penalty in persisting these objects. This penalty can be reduced by using time-based writes.

Notes: In general, you can obtain the best performance with session objects that are less than 2 KB in size. When the session object exceeds 4-5 KB, you can expect a significant decrease in performance.

Even if session persistence is not an issue, minimizing the session object size will help to protect your Web application from scale-up disasters as user numbers increase. Large session objects will require more and more JVM memory, leaving no room to run servlets.

See 12.9.5, “Larger DB2 page sizes and database persistence” on page 742 to learn how WebSphere can provide faster persistence of larger session objects when using DB2.

Session cache size

The session manager allows administrators to change the session cache size to alter the cache’s memory footprint. By default, the session cache holds 1000 session objects. By lowering the number of session objects in the cache, the administrator reduces the memory required by the cache.

However, if the user’s session is not in the cache, WebSphere must retrieve it from either the overflow cache, for local caching, or the session database, for persistent sessions. If the session manager must retrieve persistent sessions frequently, the retrievals can impact overall application performance.

WebSphere maintains overflowed local sessions in memory, as discussed in 12.6, “Local sessions” on page 710. Local session management with cache overflow enabled allows an unlimited number of sessions in memory. To limit the cache footprint to the number of entries specified in session manager, use persistent session management, or disable overflow.

Note: When using local session management without specifying the Allow overflow property, a full cache will result in the loss of user session objects.

Creating additional application servers

WebSphere also gives the administrator the option of creating additional application servers. Creating additional instances spread the demand for memory across more JVMs, thus reducing the memory burden on any particular instance. Depending on the memory and CPU capacity of the machines involved, the administrator can add additional instances within the same machine. Alternatively, the administrator can add additional machines to form a hardware cluster, and spread the instances across this cluster.

Note: When configuring a session cluster, session affinity routing provides the most efficient strategy for user distribution within the cluster, even with session persistence enabled. With cluster members, the Web server plug-in provides affinity routing among cluster member instances.

Invalidating unneeded sessions

If the user no longer needs the session object, for example, when the user has logged out of the site, it should be invalidated. Invalidating a session removes it from the session cache, as well as from the session database. For more information see 12.10, “Invalidating sessions” on page 749.

Increasing available memory

WebSphere allows the administrator to increase an application server’s heap size. By default, WebSphere allocates 256 MB as the maximum heap size. Increasing this value allows the instance to obtain more memory from the system, and thus hold a larger session cache.

A practical limit exists, however, for an instance heap size. The machine memory containing the instance needs to support the heap size requested. Also, if the heap size grows too large, the length of the garbage collection cycle with the JVM might impact overall application performance. This impact has been reduced with the introduction of multi-threaded garbage collection.

Session timeout interval

By default, each user receives a 30-minute interval between requests before the session manager invalidates the user’s session. Not every site requires a session timeout interval this generous. By reducing this interval to match the requirements of the average site user, the session manager purges the session from the cache and the persistent store, if enabled, more quickly.

Avoid setting this parameter too low and frustrating users. The administrator must take into account a reasonable time for an average user to interact with the site when setting the interval. User activities include reading returned data, filling out forms, and so on. Also, the interval must represent any increased response time during peak times on the site, such as heavy trading days on a brokerage site, for example.

Finally, in some cases where the persistent store contains a large number of entries, frequent execution of the timeout scanner reduces overall performance. In cases where the persistent store contains many session entries, avoid setting the session timeout so low it triggers frequent, expensive scans of the persistent store for timed-out sessions. Alternatively, the administrator should consider schedule-based invalidation where scans for invalid object can be deferred to a time that normally has low demand. See 12.10, “Invalidating sessions” on page 749.

12.12.2 Reducing persistent store I/O

From a performance point of view, the Web developer’s considerations are the following:

- Optimize the use of the HttpSession within a servlet. Only store the minimum amount of data required in HttpSession. Data that does not have to be recovered after a cluster member fails or is shut down can be best kept elsewhere, such as in a hash table. Recall that HttpSession is intended to be used as a *temporary* store for state information between browser invocations.
- Specify `session=false` in the JSP directive for JSPs that do not need to access the session object.
- Use time-based write frequency mode. This greatly reduces the amount of I/O, because the persistent store updates are deferred up to a configurable number of seconds. Using this mode, all of the outstanding updates for a Web application are written periodically based on the configured write interval.
- Use the **Schedule sessions cleanup** option. When using the End of servlet service write frequency mode, WebSphere does not have to write out the last access time with every HTTP request. This is because WebSphere does not have to synchronize the invalidator thread's deletion with the HTTP request’s access.

Archive

12.12.3 Multirow persistent sessions: Database persistence

When a session contains multiple objects accessed by different servlets or JSPs in the same Web application, multi-row session support provides a mechanism for improving performance. Multi-row session support stores session data in the persistent session database by Web application and value. Table 12-6 shows a simplified representation of a multi-row database table.

Table 12-6 Simplified multi-row session representation

Session ID	Web application	Property	Small value	Large value
DA32242SSGE2	ShoeStore	ShoeStore.First.Name	Alice	
DA32242SSGE2	ShoeStore	ShoeStore.Last.Name	Smith	
DA32242SSGE2	ShoeStore	ShoeStore.Big.String		A big string....

In this example, if the user visits the ShoeStore application, and the servlet involved needs the user's first name, the servlet retrieves this information through the session API. The session manager brings into the session cache only the value requested. The ShoeStore.Big.String item remains in the persistent session database until the servlet requests it. This saves time by reducing both the data retrieved and the serialization overhead for data the application does not use.

After the session manager retrieves the items from the persistent session database, these items remain in the in-memory session cache. The cache accumulates the values from the persistent session database over time as the various servlets within the Web application request them. With WebSphere's session affinity routing, the user returns to this same cached session instance repeatedly. This reduces the number of reads against the persistent session database, and gives the Web application better performance.

How session data is written to the persistent session database has been made configurable in WebSphere. For information about session updates using single and multi-row session support, see 12.9.6, "Single and multi-row schemas (database persistence)" on page 743. Also see 12.9.7, "Contents written to the persistent store using a database" on page 745.

Even with multi-row session support, Web applications perform best if the overall contents of the session objects remain small. Large values in session objects require more time to retrieve from the persistent session database, generate more network traffic in transit, and occupy more space in the session cache after retrieval.

Multi-row session support provides a good compromise for Web applications requiring larger sessions. However, single-row persistent session management remains the best choice for Web applications with small session objects. Single-row persistent session management requires less storage in the database, and requires fewer database interactions to retrieve a session's contents (all of the values in the session are written or read in one operation). This keeps the session object's memory footprint small, as well as reducing the network traffic between WebSphere and the persistent session database.

Note: Avoid circular references within sessions if using multi-row session support. The multi-row session support does not preserve circular references in retrieved sessions.

12.12.4 Managing your session database connection pool

When using persistent session management, the session manager interacts with the defined database through a WebSphere Application Server data source. Each data source controls a set of database connections known as a connection pool. By default, the data source opens a pool of no more than 10 connections. The maximum pool size represents the number of simultaneous accesses to the persistent session database available to the session manager.

For high-volume Web sites, the default settings for the persistent session data source might not be sufficient. If the number of concurrent session database accesses exceeds the connection pool size, the data source queues the excess requests until a connection becomes available. Data source queuing can impact the overall performance of the Web application (sometimes dramatically).

For best performance, the overhead of the connection pool used by the session manager needs to be balanced against the time that a client cab spend waiting for an occupied connection to become available for use. By definition, a connection pool is a *shared* resource, so in general the best performance is realized typically with a connection pool that has significantly fewer connections than the number of simultaneous users.

A large connection pool does not necessarily improve application performance. Each connection represents memory overhead. A large pool decreases the memory available for WebSphere to execute applications. Also, if database connections are limited because of database licensing issues, the administrator must share a limited number of connections among other Web applications requiring database access as well. This is one area where performance tuning tests are required to determine the optimal setting for a given application.

As discussed above, session affinity routing combined with session caching reduces database read activity for session persistence. Likewise, manual update write frequency, time-based write frequency, and multi-row persistent session management reduce unnecessary writes to the persistent database. Incorporating these techniques can also reduce the size of the connection pool required to support session persistence for a given Web application.

Prepared statement caching is a connection pooling mechanism that can be used to further improve session database response times. A cache of previously prepared statements is available on a connection. When a new prepared statement is requested on a connection, the cached prepared statement is returned, if available. This caching reduces the number of costly prepared statements created, which improves response times.

In general, base the prepared statement cache size on the following:

- The smaller of:
  - Number of concurrent users
  - Connection pool size
- The number of different prepared statements

With 50 concurrent users, a connection pool size of 10, and each user using 2 statements, a select and an insert, the prepared statement cache size should be at least 10 x 2 = 20 statements. To read more, see the “Prepared statement cache size” article in the *WebSphere Tuning Guide* , included with the Information Center.

12.12.5 Session database tuning

While the session manager implementation in WebSphere provides for a number of parameters that can be tuned to improve performance of applications that utilize HTTP sessions, maximizing performance requires tuning the underlying session persistence table. WebSphere provides a first step by creating an index for the sessions table when creating the table. The index is comprised of the session ID, the property ID for multi-row sessions, and the Web application name.

While most database managers provide a great deal of capability in tuning at the table or tablespace level, creating a separate database or instance provides the most flexibility in tuning. Proper tuning of the instance and database can improve performance by 5% or more over that which can be achieved by simply tuning the table or tablespace.

Archived

While the specifics vary, depending on you database and operating system, in general tune and configure the database as appropriate for a database that experiences a great deal of I/O. The database administrator (DBA) should monitor and tune the database buffer pools, database log size, and write frequency. Additionally, maximizing performance requires striping the database or instance across multiple disk drives and disk controllers, and utilizing any hardware or OS buffering available to reduce disk contention.

12.13 Stateful session bean failover

Stateful session bean failover is supported now in WebSphere Application Server V6. This feature utilizes the functions of the data replication service and workload management.

Each EJB container provides a method for stateful session beans to fail over to other servers. This enables you to specify whether failover occurs for the stateful session beans at the EJB module level or container level. You can also override the parent object's stateful session bean replication settings from the module level.

12.13.1 Enabling stateful session bean failover

Depending on the requirement, you might not want to enable failover for every single stateful session bean installed in the EJB container. You can set or override the EJB container settings at either the application or EJB module level. You can either enable or disable failover at each of these levels. For example, consider the following situations:

- You want to enable failover for all applications except for a single application. To do this, you enable failover at the EJB container level and override the setting at the application level to disable failover on the single application.
- You want to enable failover for a single, installed application. To do this, disable failover at the EJB container level and then override the setting at the application level to enable failover on the single application.
- You want to enable failover for all applications except for a single module of an application. To do this, enable failover at the EJB container level, then override the setting at the module application level to disable failover on the single module.
- You want to enable failover for a single, installed EJB module. To do this, disable failover at the EJB container level and then override the setting at the EJB module level to enable failover on the single EJB module.

EJB container stateful session bean failover properties

To access stateful session bean failover properties at the EJB container level from the administrative console:

1. Click **Servers** →**Application servers**.
2. Click the application server.
3. In the Container Settings section of the Configuration tab, click **EJB container**.
4. In the General Properties section, check **Enable stateful session bean failover using memory-to-memory replication**.

   This check box is disabled until you define a replication domain. This selection has a hyperlink to help you configure the replication settings. If no replication domains are configured, the link takes you to a panel where you can create one. If at least one domain is configured, the link takes you to a panel where you can select the replication settings to be used by the EJB container. See Figure 12-18 on page 762.

Archived

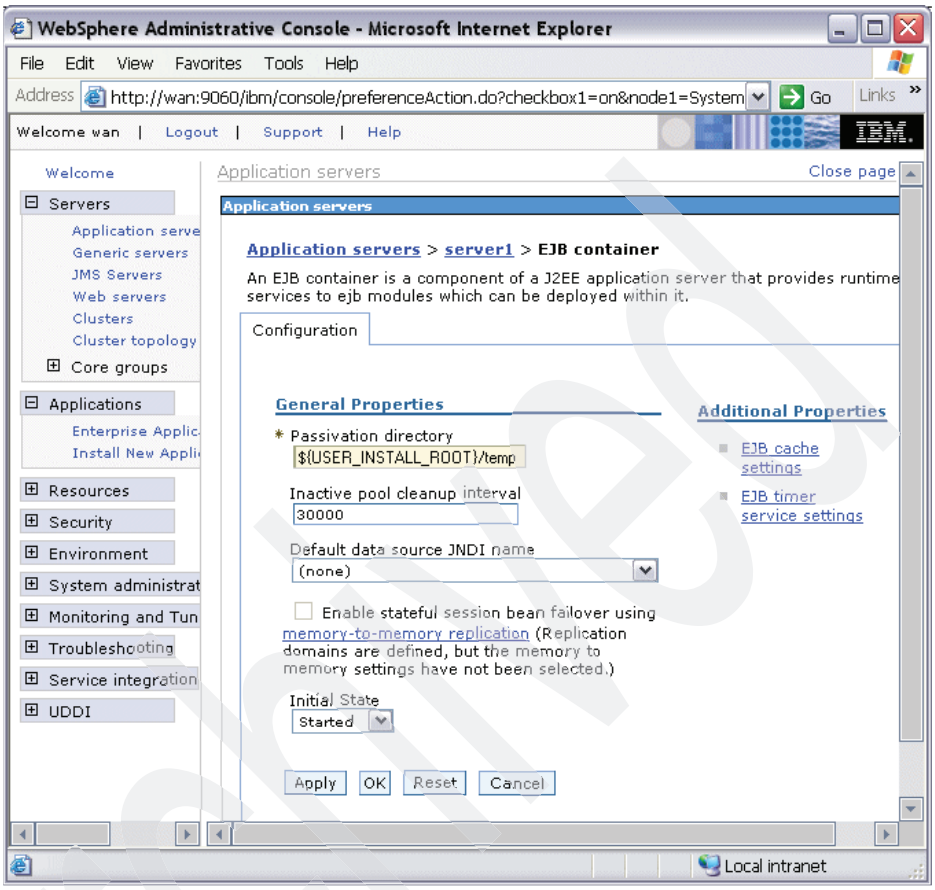


Figure 12-18 Stateful session bean failover settings at the container level

EJB module stateful session bean failover properties

To access stateful session bean failover properties at the EJB module level from the administrative console:

1. Click **Applications** →**Enterprise applications**.
2. Click the application.
3. In the Additional Properties section of the Configuration tab, click **Stateful session bean failover settings.**

   This enables failover for all stateful session beans in this application. If you want to disable the failover, clear this check box. See Figure 12-19 on page 763.

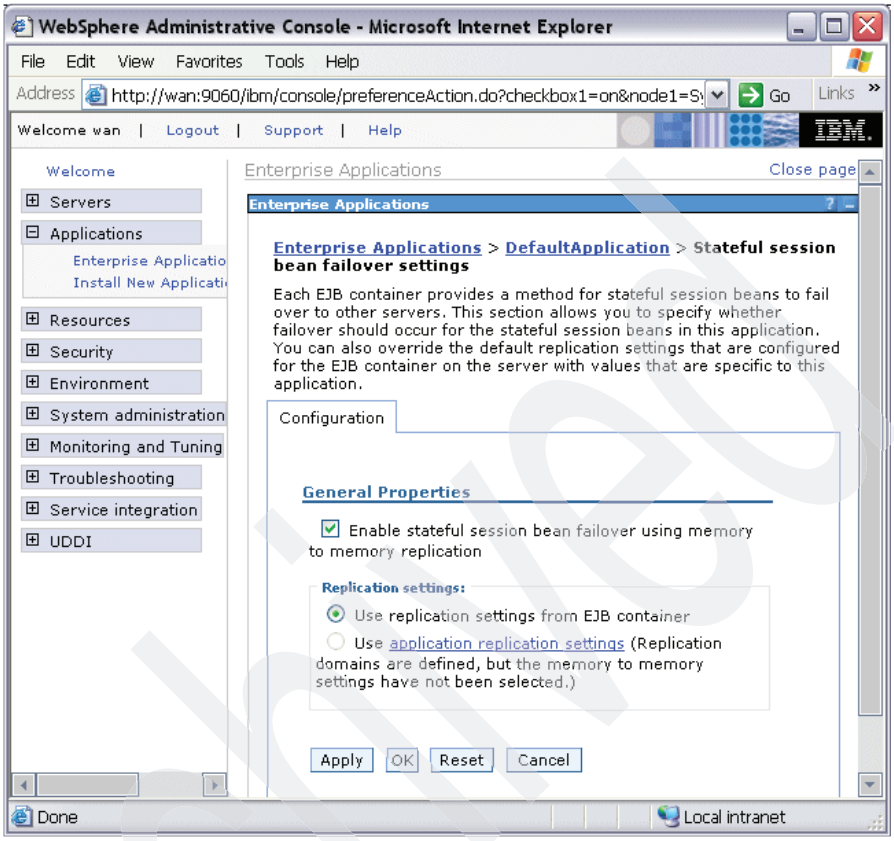


Figure 12-19 Stateful session bean failover settings at the module level

4. In the General Properties section, select your choice of **Replication settings**:
   - **Use replication settings from EJB container**

     If you select this option, any replication settings defined for this application are ignored.

     Important: If you use this radio button, then you must configure memory to memory replication at the EJB container level. Otherwise, the settings on this panel are ignored by EJB container during server startup and the EJB container will log a message indicating that stateful session bean failover is not enabled for this application.

Archive

– **Use application replication settings**

  If you select this option, you override the EJB container settings. This button is disabled until you define a replication domain. This selection has a hyperlink to help you configure the replication settings. If no replication domains are configured, the link takes you to a panel to create one. If at least one domain is configured, the link takes you to a panel where you can select the replication settings to be used by the application.

5. Select your choice of replication from:
   – **Use replication settings from EJB container** or
   – **Use application replication settings using memory-to-memory replication**
6. Select **OK**.

Note: The stateful session bean failover settings are available to WebSphere Application Server V6 enterprise applications. They are ignored by WebSphere Application Server V5 enterprise applications.

12.13.2 Stateful session bean failover considerations

The following presents a few considerations when using the stateful session bean failover feature.

Stateful session bean activation policy with failover enabled

WebSphere Application Server V6 allows an application assembler to specify an activation policy to use for stateful session beans. It is important to consider that the only time the EJB container prepares for failover. by replicating the stateful session bean data using DRS, is when the stateful session bean is passivated. If you configure the bean with an activate once policy, the bean is essentially never passivated. If you configure the activate at transaction boundary policy, the bean is passivated whenever the transaction that the bean is enlisted in completes. For stateful session bean failover to be useful, the activate at transaction boundary policy is required.

Rather than forcing you to edit the deployment descriptor of every stateful session bean and reinstall the bean, the EJB container simply ignores the configured activation policy for the bean when you enable failover. The container automatically uses the activate at transaction boundary policy.

Container or bean managed units of work

The relevant units of work in this case are transactions and activity sections. WebSphere Application Server V6 supports stateful session bean failover for:

- Container managed transactions (CMT)
- Bean managed transactions (BMT)
- Container managed activity sessions (CMAS)
- Bean managed activity sessions (BMAS).

In the container-managed cases, preparation for failover only occurs if a request for an enterprise bean method invocation fails to connect to the server. Also, failover does not take place if the server fails after a request is sent to it and had been acknowledged.

When a failure occurs in the middle of a request or unit of work, WLM cannot safely fail over to another server without some compensation code being executed by the application. When that happens, the application receives a Common Object Request Broker Architecture (CORBA) exception and minor code telling it that transparent failover could not occur because the failure happened during execution of a unit of work. The application should be written to check for the CORBA exception and minor code, and compensate for the failure. After the compensation code executes, the application can retry the requests and if a path exists to a backup server WLM routes the new request to a new primary server for the stateful session bean.

The same is true for bean-managed units of work, transactions or activity sessions. However, bean managed work introduces a new possibility that needs to be considered.

For bean managed units of work, the failover process is not always able to detect that a BMT or BMAS started by a stateful session bean method has not completed. Thus, it is possible that failover to a new server can occur despite the unit of work failing during the middle of a transaction or session. Because the unit of work is implicitly rolled back, WLM behaves as thought it is safe to transparently fail over to another server, when in fact some compensation code might be required. When this happens, the EJB container detects this on the new server and initiates an exception. This exception occurs under the following scenario:

1. A method of a stateful session bean using bean-managed transaction or activity session calls begin on a UserTransaction it obtained from the SessionContext. The method does some work in the started unit of work, but does not complete the transaction or session before returning to the caller of the method.
2. During post invocation of the method started in step 1, the EJB container suspends the work started by the method. This is the action required by EJB specification for bean managed units of work when the bean is a stateful session bean.

3. The client starts several other methods on the stateful session bean. Each invocation causes the EJB container to resume the suspended transaction or activity session, dispatch the method invocation, and then suspend the work again before returning to the caller.
4. The client calls a method on the stateful session bean that completes the transaction or session started in step 1.

This scenario depicts a *sticky* bean-managed unit of work. The transaction or activity session sticks around for more than a single stateful session bean method. If an application uses a sticky BMT or BMAS, and the server fails after a sticky unit of work completes and before another sticky unit of work starts, failover is successful. However, if the server fails before a sticky transaction or activity session completes, the failover is not successful. Instead, when the failover process routes the stateful session bean request to a new server, the EJB container detects that the failure occurred during an active, sticky transaction or activity session. At that time, the EJB container initiates an exception.

Essentially, this means that failover for both container-managed and bean-managed units of work is not successful if the transaction or activity session is still active. The only real difference is the exception that occurs.

Application design considerations

Consider the following when designing applications that use the stateful session bean failover process:

- To avoid the possibility described in the section above, you are encouraged to write your application to configure stateful session beans to use container-managed transactions (CMT) rather than bean-managed transactions (BMT).
- If you want immediate failover, and your application creates either an HTTP session or a stateful session bean that stores a reference to another stateful session bean, then the administrator must ensure the HTTP session and stateful session bean are configured to use the same replication domain.
- Do not use a local and a remote reference to the same stateful session bean.

  J2EE 1.4 specification has added additional requirements for Http Sessions which required the Http Session state objects to be able to contain local references to EJBs.

  Normally a stateful session bean instance with a given primary key can only exist on a single server at any given moment in time. Failover might cause the bean to be moved from one server to another, but it never exists on more than one server at a time. However, there are some unlikely scenarios that can result in the same bean instance, the same primary key, existing on more

than one server concurrently. When that happens, each copy of the bean is unaware of the other, and no synchronization occurs between the two instances to ensure they have the same state data. Thus, your application receives unpredictable results.

Note: To avoid this situation you must remember that with failover enabled, your application should never get both a local (EJBLocalObject) and remote (EJBObject) reference to the same stateful session bean instance.

Archived

Archived

13

WebSphere naming implementation

In this chapter, we describe the concepts behind the naming functionality provided as part of IBM WebSphere Application Server:

- Features in WebSphere Application Server V6
- WebSphere naming architecture
- Interoperable Naming Service (INS)
- Distributed CosNaming
- Configured bindings
- Initial contexts
- Federation of name spaces
- Interoperability
- Examples
- Naming tools
- Configuration

© Copyright IBM Corp. 2005. All rights reserved.

13.1 Features

The following are features of a WebSphere Application Server V6 name space that remain unchanged from WebSphere Application Server V5:

- Distributed name space

  For additional scalability, the name space for a cell is distributed among the various servers. The deployment manager, node agent and application server processes all host a name server.

  The default initial context for a server is its server root. System artifacts, such as EJB homes and resources, are bound to the server root of the server with which they are associated.

- Transient and persistent partitions

  The name space is partitioned into transient areas and persistent areas. Server roots are transient. System-bound artifacts such as EJB homes and resources are bound under server roots. There is a cell persistent root, which can be used for cell-scoped persistent bindings, and a node persistent root, which can be used to bind objects with a node scope.

- Federated name space structure

  A name space is a collection of all names bound to a particular name server. A name space can contain naming context bindings to contexts located in other servers. If this is the case, the name space is said to be a *federated name space*, because it is a collection of name spaces from multiple servers. The name spaces link together to cooperatively form a single logical name space.

  The name space for the WebSphere Application Server V6 cell is federated among the deployment manager, node agents, and application servers of the cell. Every server process hosts a name server. All name servers provide the same logical view of the cell name space, with the various server roots and persistent partitions of the name space being interconnected by means of the single logical name space.

- Configured bindings

  Administrators can configure bindings into the name space. A configured binding is different from a programmatic binding in that the system creates the binding every time a server is started, even if the target context is in a transient partition.

- Support for CORBA Interoperable Naming Service (INS) object URLs

  WebSphere Application Server contains support for CORBA object URLs (corbaloc and corbaname) as JNDI provider URLs and lookup names.

Archived

13.2 WebSphere naming architecture

WebSphere Application Server name servers are an implementation of the CORBA CosNaming interface. WebSphere Application Server provides a JNDI implementation that you can use to access CosNaming name servers through the JNDI interface. CosNaming provides the server-side implementation and is where the name space is stored. JNDI essentially provides a client-side wrapper of the name space stored in CosNaming, and interacts with the CosNaming server on behalf of the client.

The model of JNDI over CosNaming has existed in several releases of WebSphere. Since J2EE 1.3, limited CosNaming functionality has been required for interoperability between application servers from different vendors. This level of CosNaming, known as Interoperable Naming Service (INS), was introduced in WebSphere Application Server V5.

The following sections provide a summary of the WebSphere naming architecture, its federated name space, and its support for JNDI.

For an explanation of the WebSphere implementations of INS and Distributed CosNaming, see 13.3, “Interoperable Naming Service (INS)” on page 785 and 13.4, “Distributed CosNaming” on page 787 respectively.

13.2.1 Components

WebSphere application clients use the naming service to obtain references to objects related to those applications, such as EJB homes. These objects are bound into a mostly hierarchical structure, referred to as a *name space*. In this structure, all non-leaf objects are called *contexts*. Leaf objects can be contexts and other types of objects. Naming operations, such as lookups and binds, are performed on contexts. All naming operations begin with obtaining an initial context. You can view the initial context as a starting point in the name space.

The name space structure consists of a set of *name bindings*, each consisting of a name relative to a specific context and the object bound with that name. For example, the name myApp/myEJB consists of one non-leaf binding with the name myApp, which is a context. The name also includes one leaf binding with the name myEJB, relative to myApp. The object bound with the name myEJB in this example happens to be an EJB home reference. The whole name myApp/myEJB is relative to the initial context, which can be viewed as a starting place when performing naming operations.

The name space can be accessed and manipulated through a *name server*. Users of a name server are referred to as naming clients. Naming clients typically use Java Naming and Directory Interface (JNDI) to perform naming

operations. Naming clients can also use the Common Object Request Broker Architecture (CORBA) CosNaming interface.

Figure 13-1 summarizes the naming architecture and its components.

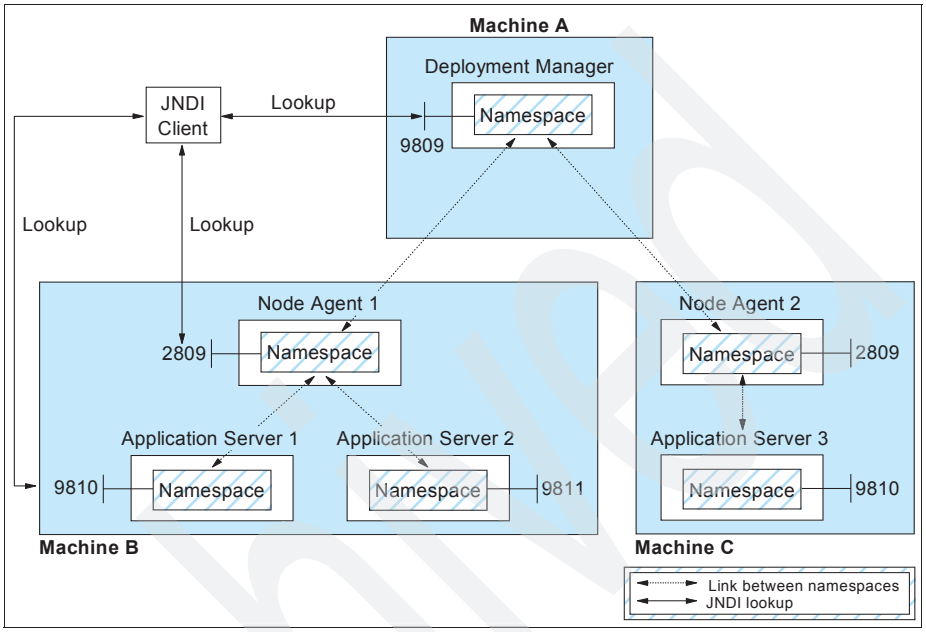


Figure 13-1 Naming topology

Notice that all WebSphere Application Server processes host their own naming service and local name space. Also the name servers in the deployment manager and node agents are listening on their default ports of 9809 and 2809, respectively. The name servers within each application server are listening from a starting default port of 9810.

13.2.2 JNDI support

Each IBM WebSphere Application Server managed process (JVM) includes:

- A name server providing shared access to its components
- An implementation of the javax.naming JNDI package, allowing users to access the WebSphere name server through the JNDI naming interface

Note: The JNDI implementation provided by IBM WebSphere Application Server is based on Version 1.2.1 of the JNDI interface, and was tested with Version 1.2.1 of Sun's JNDI SPI (Service Provider Interface).

IBM WebSphere Application Server does not provide implementations for the following Java extension packages:

- javax.naming.directory
- javax.naming.ldap

In addition, IBM WebSphere Application Server does not support interfaces defined in the javax.naming.event package.

However, to provide access to LDAP servers, the JDK shipped with IBM WebSphere Application Server supports Sun Microsystem's implementation of:

- javax.naming.ldap
- com.sun.jndi.ldap.LdapCtxFactory

13.2.3 JNDI bindings

There are three options available for binding EJB (<ejb-ref>) and resource (<resource-ref>) object names to the WebSphere Application Server name space:

- Simple name
- Compound/fully qualified name
- Corbaname

The binding you can use to look up an object depends on whether or not the application is running within the same application server. The following sections describe each of these in more detail.

Simple name

The simple name binding is guaranteed to succeed if lookup is within the same server or when connected directly to the name space of the server containing the target of the lookup. It can be used in a servlet or EJB, if it is certain that the object is located on the same application server. Here is an example of a simple name:

```
ejb/webbank/Account
```

Lookup names of this form provide a level of indirection such that the name used to look up an object is not dependent on the object's name as it is bound in the name server's name space. The deployment descriptors for the application provide the mapping between the name and the name server lookup name. The container sets up the name space based on the deployment descriptor information so that the name is correctly mapped to the corresponding object.

Archived

Compound name

Applications that do not run in the same server cannot use simple name lookup because the simple name is not local to the application. Instead, an application of this type must look the object up directly from the name server. Each application server contains a name server. System artifacts such as EJB homes are bound relative to the server root context in that name server.

The fully qualified (compound name) JNDI name is always guaranteed to work. Here is an example of a compound name:

```
cell/nodes/node1/servers/server1/ejb/webbank/Account
```

We recommend using compound names for JNDI bindings.

Corbaname

The corbaname binding is always guaranteed to work. However, it requires that you know the correct path to the object at deployment time. Here is an example of a corbaname:

```
corbaname::myhost1:9812/NameServiceServerRoot#ejb/webbank/Account
```

13.2.4 Federated name space

All name servers with a cell are federated into the cell name space. Every application server process contains a name server. All name servers provide the same logical view of the cell name space. The various server roots and persistent partitions of the name space are interconnected by a system name space. You can use the system name space structure to traverse any context in the cell's name space. A logical view of the name space is shown in Figure 13-2 on page 775.

Archived

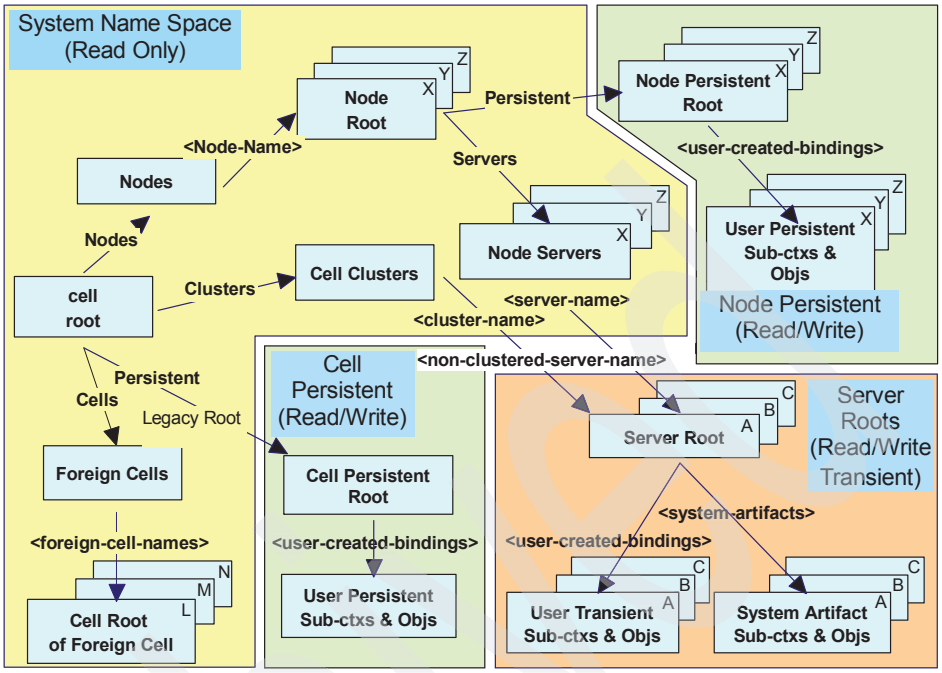


Figure 13-2 Federated name space

The name space can be broken down into distinct partitions that are updateable and persistent.

System partition

The system partition is a reflection of the cell topology and is read-only. This part of the name space cannot be changed programmatically, because it is based on the configuration rather than runtime settings.

The root of this structure is the cell root and contains a node root for each node in the cell. You can access other contexts to a specific node from the node root, such as the node persistent root and server roots for servers configured in that node.

This partition of the name space is persistent, with the data stored in the XML repository containing the topological information.

Cell and node persistent partitions

The persistent partitions are primarily for the storage of resource configuration, such as data sources, JMS destinations etc. This data can be modified by accessing the JNDI APIs directly, or through the administration clients, which

access the APIs on the user's behalf. The persistent data is stored to a group of XML files.

There are two persistent partitions in the federated name space:

- Cell persistent root

  This partition is used to register persistent objects that are available to all the nodes and managed processes of a cell. A binding created under the cell persistent root is saved as part of the cell configuration and continues to exist until it is explicitly removed.

  Applications that need to create additional persistent object bindings associated with the cell can bind those objects under the cell persistent root.

  Note: The cell persistent root is not designed for transient, rapidly changing bindings. Instead, the bindings should be more static in nature, such as part of application setup or configuration, and not created at runtime.

  To bind objects to the cell persistent root, the deployment manager and all node agents in the cell must be running.

- Node persistent root

  This partition is used to register persistent objects available to the nodes and it their managed processes. It is similar to the cell partition except that each node has its own node persistent root. A binding created under a node persistent root is saved as part of that node's configuration and continues to exist until it is explicitly removed.

  Applications that need to create additional persistent object bindings associated with a specific node can bind those objects under that particular node's node persistent root.

Archive

Note: The node persistent area is not designed for transient, rapidly changing bindings. Instead, the bindings should be more static in nature, such as part of application setup or configuration, and not created at runtime.

The node persistent area for a node can be read from any server in the node even if the respective node agent is not running. However, the node agent must be running to update the node persistent area, or for any server outside the node to read from that node persistent partition.

In the federated name space, there is no node root for the deployment manager node because no node agent or application servers run in that node.

Transient partitions

The server root transient partition in Figure 13-2 on page 775 is updateable through APIs, and is meant for information such as EJB bindings and JNDI names. This name space is transient and bindings are created each time a server process starts. It reads configuration data from the file system, for example EJB deployment descriptors, to register the necessary objects in this space.

Note: The Naming Service of each managed process listens to configuration changes. This means the local name space is updated automatically when configuration changes occur. For example, there is no need to restart a node agent to update its name space when a new application server is created.

13.2.5 Local name space structure

The structure of the federated name space is hierarchical, with each process' local name space federated and linked using corbaloc URLs that allow transparent name searches both within a single local name space and from one local name space to another.

The contents of the cell, node, and process local name spaces are described in the following sections.

Cell-level name space

The cell-level name space, hosted by the deployment manager, has a structure as shown in Example 13-1.

`(top)` represents the root of the federated name space.

Example 13-1 Cell name space dump (dumpNameSpace -port <dmgr_bootstrap>)

```
1 (top)
    2 (top)/clusters                                    javax.naming.Context
    3 (top)/clusters/MyCluster                          javax.naming.Context
    3    Linked to URL: corbaloc::wan:9811,:wan:9812/NameServiceServerRoot
    4 (top)/domain                                      javax.naming.Context
    4    Linked to context: ITSOCell
    5 (top)/legacyRoot                                  javax.naming.Context
    5    Linked to context: ITSOCell/persistent
    6 (top)/persistent                                  javax.naming.Context
    7 (top)/persistent/cell                             javax.naming.Context
    7    Linked to context: ITSOCell
    8 (top)/cellname                                    java.lang.String
    9 (top)/cell                                        javax.naming.Context
    9    Linked to context: ITSOCell
   10 (top)/nodes                                       javax.naming.Context
   11 (top)/nodes/ITSOCellManager                       javax.naming.Context
   12 (top)/nodes/ITSOCellManager/domain                javax.naming.Context
   12    Linked to context: ITSOCell
   13 (top)/nodes/ITSOCellManager/servers               javax.naming.Context
   14 (top)/nodes/ITSOCellManager/servers/dmgr          javax.naming.Context
   15 (top)/nodes/ITSOCellManager/servers/dmgr/tm       javax.naming.Context
   16 (top)/nodes/ITSOCellManager/servers/dmgr/tm/default
   16
com.ibm.ws.asynchbeans.timer.TimerManagerImpl
   17 (top)/nodes/ITSOCellManager/servers/dmgr/com.ibm.isc
   17                                                   javax.naming.Context
   18 (top)/nodes/ITSOCellManager/servers/dmgr/com.ibm.isc/PluginRegistry
   18
com.ibm.ws.PluginRegistry
   19 (top)/nodes/ITSOCellManager/servers/dmgr/services  javax.naming.Context
   20 (top)/nodes/ITSOCellManager/servers/dmgr/services/cache
   20                                                   javax.naming.Context
   21 (top)/nodes/ITSOCellManager/servers/dmgr/services/cache/basecache
   21
com.ibm.websphere.cache.DistributedObjectCache
   22 (top)/nodes/ITSOCellManager/servers/dmgr/services/cache/distributedmap
   22
com.ibm.websphere.cache.DistributedObjectCache
   23 (top)/nodes/ITSOCellManager/servers/dmgr/ejb       javax.naming.Context
   24 (top)/nodes/ITSOCellManager/servers/dmgr/ejb/mgmt  javax.naming.Context
   25 (top)/nodes/ITSOCellManager/servers/dmgr/ejb/mgmt/MEJB
   25
javax.management.j2ee.ManagementHome
   26 (top)/nodes/ITSOCellManager/servers/dmgr/cell      javax.naming.Context
   26    Linked to context: ITSOCell
   27 (top)/nodes/ITSOCellManager/servers/dmgr/servername
   27                                                   java.lang.String
```

Archive

```
28 (top)/nodes/ITSOCellManager/servers/dmgr/thisNode  javax.naming.Context
28    Linked to context: ITSOCell/nodes/ITSOCellManager
29 (top)/nodes/ITSOCellManager/node                   javax.naming.Context
29    Linked to context: ITSOCell/nodes/ITSOCellManager
30 (top)/nodes/ITSOCellManager/cell                   javax.naming.Context
30    Linked to context: ITSOCell
31 (top)/nodes/ITSOCellManager/nodename               java.lang.String
32 (top)/nodes/NodeA                                  javax.naming.Context
33 (top)/nodes/NodeA/nodename                         java.lang.String
34 (top)/nodes/NodeA/persistent                       javax.naming.Context
34    Linked to URL: corbaname::wan:2809/NameServiceNodeRoot#persistent
35 (top)/nodes/NodeA/cell                             javax.naming.Context
35    Linked to context: ITSOCell
36 (top)/nodes/NodeA/domain                           javax.naming.Context
36    Linked to context: ITSOCell
37 (top)/nodes/NodeA/nodeAgent                        javax.naming.Context
37    Linked to URL: corbaloc::wan:2809/NameServiceServerRoot
38 (top)/nodes/NodeA/node                             javax.naming.Context
38    Linked to context: ITSOCell/nodes/NodeA
39 (top)/nodes/NodeA/servers                          javax.naming.Context
40 (top)/nodes/NodeA/servers/MyClusterServer2         javax.naming.Context
40    Linked to URL: corbaloc::wan:9812/NameServiceServerRoot
41 (top)/nodes/NodeA/servers/server1                  javax.naming.Context
41    Linked to URL: corbaloc::wan:9810/NameServiceServerRoot
42 (top)/nodes/NodeA/servers/nodeagent                javax.naming.Context
42    Linked to URL: corbaloc::wan:2809/NameServiceServerRoot
43 (top)/nodes/NodeA/servers/MyClusterServer1         javax.naming.Context
43    Linked to URL: corbaloc::wan:9811/NameServiceServerRoot
44 (top)/deploymentManager                            javax.naming.Context
44    Linked to context: ITSOCell/nodes/ITSOCellManager/servers/dmgr
45 (top)/cells                                        javax.naming.Context
```

The cell-level name space contains:

- A link to the cell persistent root, /persistent/cell
- A hierarchy of contexts for nodes and the servers managed by each node
- A full set of entries for the deployment manager node (ITSOCellManager) and the deployment manager server (dmgr)
- The objects registered in JNDI by the dmgr server
- A corbaloc URL link to the local name space of each of the other nodes in the cell
- A number of cross links for the federated name space:
  - /cells
  - /clusters
  - /legacyRoot

Archive

Note: Because of the hierarchical structure of the cell/node/server relationship, the following naming conventions and constraints exist:

1. No two nodes can have the same name. Node names must be unique within a cell.
2. Two application servers on different nodes can have the same name.
3. Two application servers on the same node must have different names. Application server names must be unique within a node.

Node-level name space

A node-level name space, hosted by a node agent, has a structure as shown in Example 13-2.

Example 13-2 Node-level name space (dumpNameSpace -port <NodeA_bootstrap>)

```
1 (top)
    2 (top)/clusters                                    javax.naming.Context
    3 (top)/clusters/MyCluster                          javax.naming.Context
    3    Linked to URL: corbaloc::wan:9811,:wan:9812/NameServiceServerRoot
    4 (top)/domain                                      javax.naming.Context
    4    Linked to context: ITSOCell
    5 (top)/legacyRoot                                  javax.naming.Context
    5    Linked to context: ITSOCell/persistent
    6 (top)/persistent                                  javax.naming.Context
    7 (top)/persistent/cell                             javax.naming.Context
    7    Linked to context: ITSOCell
    8 (top)/cellname                                    java.lang.String
    9 (top)/cell                                        javax.naming.Context
    9    Linked to context: ITSOCell
   10 (top)/nodes                                       javax.naming.Context
   11 (top)/nodes/ITSOCellManager                       javax.naming.Context
   12 (top)/nodes/ITSOCellManager/domain                javax.naming.Context
   12    Linked to context: ITSOCell
   13 (top)/nodes/ITSOCellManager/servers               javax.naming.Context
   14 (top)/nodes/ITSOCellManager/servers/dmgr          javax.naming.Context
   14    Linked to URL: corbaloc::wan:9809/NameServiceServerRoot
   15 (top)/nodes/ITSOCellManager/node                  javax.naming.Context
   15    Linked to context: ITSOCell/nodes/ITSOCellManager
   16 (top)/nodes/ITSOCellManager/nodename              java.lang.String
   17 (top)/nodes/ITSOCellManager/cell                  javax.naming.Context
   17    Linked to context: ITSOCell
   18 (top)/nodes/NodeA                                 javax.naming.Context
   19 (top)/nodes/NodeA/nodename                        java.lang.String
   20 (top)/nodes/NodeA/persistent                      javax.naming.Context
   21 (top)/nodes/NodeA/cell                            javax.naming.Context
   21    Linked to context: ITSOCell
```

```
22 (top)/nodes/NodeA/domain                          javax.naming.Context
22    Linked to context: ITSOCell
23 (top)/nodes/NodeA/nodeAgent                       javax.naming.Context
23    Linked to context: ITSOCell/nodes/NodeA/servers/nodeagent
24 (top)/nodes/NodeA/node                            javax.naming.Context
24    Linked to context: ITSOCell/nodes/NodeA
25 (top)/nodes/NodeA/servers                         javax.naming.Context
26 (top)/nodes/NodeA/servers/MyClusterServer2        javax.naming.Context
26    Linked to URL: corbaloc::wan:9812/NameServiceServerRoot
27 (top)/nodes/NodeA/servers/server1                 javax.naming.Context
27    Linked to URL: corbaloc::wan:9810/NameServiceServerRoot
28 (top)/nodes/NodeA/servers/nodeagent               javax.naming.Context
29 (top)/nodes/NodeA/servers/nodeagent/cell          javax.naming.Context
29    Linked to context: ITSOCell
30 (top)/nodes/NodeA/servers/nodeagent/servername    java.lang.String
31 (top)/nodes/NodeA/servers/nodeagent/thisNode      javax.naming.Context
31    Linked to context: ITSOCell/nodes/NodeA
32 (top)/nodes/NodeA/servers/MyClusterServer1        javax.naming.Context
32    Linked to URL: corbaloc::wan:9811/NameServiceServerRoot
33 (top)/deploymentManager                           javax.naming.Context
33    Linked to URL: corbaloc::wan:9809/NameServiceServerRoot
34 (top)/cells                                       javax.naming.Context
```

The node-level name space contains:

- A full set of entries for the node and the node agent server (<nodename> or nodeAgent)
- A corbaloc URL link to the local name space root (NameServiceServerRoot) of each application server managed by the node
- Links to other nodes in the cell and to the servers on those nodes
- A corbaloc URL link to the root of the deployment manager local name space (NameServiceServerRoot)
- A number of cross-links for the federated name space:
  - /cells
  - /clusters
  - /legacyRoot
- A link to the cell persistent root, /persistent/cell
- A link to the node persistent root, /nodes/<nodename>/persistent

Managed process-level name space

A process-level name space, hosted by a managed process, has a structure as shown in Example 13-3 on page 782.

Archived

Example 13-3 Managed process-level name space

```
1 (top)
    2 (top)/domain                                          javax.naming.Context
    2    Linked to context: ITSOCell
    3 (top)/cellname                                        java.lang.String
    4 (top)/nodes                                           javax.naming.Context
    5 (top)/nodes/ITSOCellManager                           javax.naming.Context
    6 (top)/nodes/ITSOCellManager/servers                   javax.naming.Context
    7 (top)/nodes/ITSOCellManager/servers/dmgr              javax.naming.Context
    7    Linked to URL: corbaloc::wan:9809/NameServiceServerRoot
    8 (top)/nodes/ITSOCellManager/domain                    javax.naming.Context
    8    Linked to context: ITSOCell
   9 (top)/nodes/ITSOCellManager/cell                      javax.naming.Context
   9    Linked to context: ITSOCell
   10 (top)/nodes/ITSOCellManager/nodename                  java.lang.String
   11 (top)/nodes/ITSOCellManager/node                      javax.naming.Context
   11    Linked to context: ITSOCell/nodes/ITSOCellManager
   12 (top)/nodes/NodeA                                     javax.naming.Context
   13 (top)/nodes/NodeA/nodename                            java.lang.String
   14 (top)/nodes/NodeA/persistent                          javax.naming.Context
   15 (top)/nodes/NodeA/cell                                javax.naming.Context
   15    Linked to context: ITSOCell
   16 (top)/nodes/NodeA/domain                              javax.naming.Context
   16    Linked to context: ITSOCell
   17 (top)/nodes/NodeA/nodeAgent                           javax.naming.Context
   17    Linked to URL: corbaloc::wan:2809/NameServiceServerRoot
   18 (top)/nodes/NodeA/node                                javax.naming.Context
   18    Linked to context: ITSOCell/nodes/NodeA
   19 (top)/nodes/NodeA/servers                             javax.naming.Context
   20 (top)/nodes/NodeA/servers/server1                     javax.naming.Context
   21 (top)/nodes/NodeA/servers/server1/servername          java.lang.String
   22 (top)/nodes/NodeA/servers/server1/services            javax.naming.Context
   23 (top)/nodes/NodeA/servers/server1/services/cache      javax.naming.Context
   24 (top)/nodes/NodeA/servers/server1/services/cache/basecache
   24                      com.ibm.websphere.cache.DistributedObjectCache
   25 (top)/nodes/NodeA/servers/server1/services/cache/distributedmap
   25                      com.ibm.websphere.cache.DistributedObjectCache
    26 (top)/nodes/NodeA/servers/server1/thisNode           javax.naming.Context
    26    Linked to context: ITSOCell/nodes/NodeA
   27 (top)/nodes/NodeA/servers/server1/com                 javax.naming.Context
   28 (top)/nodes/NodeA/servers/server1/com/ibm             javax.naming.Context
   29 (top)/nodes/NodeA/servers/server1/com/ibm/websphere
   29                                                       javax.naming.Context
   30 (top)/nodes/NodeA/servers/server1/com/ibm/websphere/ejbquery
   30                                                       javax.naming.Context
```

Archived

```
   31 (top)/nodes/NodeA/servers/server1/com/ibm/websphere/ejbquery/Query
   31                                    com.ibm.websphere.ejbquery.QueryHome
   32 (top)/nodes/NodeA/servers/server1/Increment
                                       com.ibm.defaultapplication.IncrementHome
   33 (top)/nodes/NodeA/servers/server1/DefaultDatasource
   33                                    javax.resource.cci.ConnectionFactory
   34 (top)/nodes/NodeA/servers/server1/eis              javax.naming.Context
   35 (top)/nodes/NodeA/servers/server1/eis/jdbc         javax.naming.Context
   36(top)/nodes/NodeA/servers/server1/eis/jdbc/PlantsByWebSphereDataSource_CMP
   36                                   javax.resource.cci.ConnectionFactory
   37 (top)/nodes/NodeA/servers/server1/eis/DefaultDatasource_CMP
37                                     javax.resource.cci.ConnectionFactory
   38 (top)/nodes/NodeA/servers/server1/jdbc             javax.naming.Context
   39 (top)/nodes/NodeA/servers/server1/jdbc/PlantsByWebSphereDataSource
   39                                    javax.resource.cci.ConnectionFactory
   40 (top)/nodes/NodeA/servers/server1/jdbc/DefaultEJBTimerDataSource
   40                                    javax.resource.cci.ConnectionFactory
   41 (top)/nodes/NodeA/servers/server1/tm               javax.naming.Context
   42 (top)/nodes/NodeA/servers/server1/tm/default
com.ibm.ws.asynchbeans.timer.TimerManagerImpl
   43 (top)/nodes/NodeA/servers/server1/plantsby         javax.naming.Context
   44 (top)/nodes/NodeA/servers/server1/plantsby/LoginHome
   44                  com.ibm.websphere.samples.plantsbywebsphereejb.LoginHome
   45 (top)/nodes/NodeA/servers/server1/plantsby/MailerHome
   45               com.ibm.websphere.samples.plantsbywebsphereejb.MailerHome
   46 (top)/nodes/NodeA/servers/server1/plantsby/BackOrderHome
   46             com.ibm.websphere.samples.plantsbywebsphereejb.BackOrderHome
   47 (top)/nodes/NodeA/servers/server1/plantsby/SuppliersHome
   47            com.ibm.websphere.samples.plantsbywebsphereejb.SuppliersHome
   48 (top)/nodes/NodeA/servers/server1/plantsby/ResetDBHome
   48               com.ibm.websphere.samples.plantsbywebsphereejb.ResetDBHome
   49 (top)/nodes/NodeA/servers/server1/plantsby/ReportGeneratorHome
   49     com.ibm.websphere.samples.plantsbywebsphereejb.ReportGeneratorHome
   50 (top)/nodes/NodeA/servers/server1/plantsby/CatalogHome
   50               com.ibm.websphere.samples.plantsbywebsphereejb.CatalogHome
   51 (top)/nodes/NodeA/servers/server1/plantsby/ShoppingCartHome
   51         com.ibm.websphere.samples.plantsbywebsphereejb.ShoppingCartHome
   52 (top)/nodes/NodeA/servers/server1/plantsby/SupplierHome
   52               com.ibm.websphere.samples.plantsbywebsphereejb.SupplierHome
   53 (top)/nodes/NodeA/servers/server1/plantsby/BackOrderStockHome
   53     com.ibm.websphere.samples.plantsbywebsphereejb.BackOrderStockHome
   54 (top)/nodes/NodeA/servers/server1/mail             javax.naming.Context
   55 (top)/nodes/NodeA/servers/server1/mail/PlantsByWebSphere
   55                                                   javax.mail.Session
   56 (top)/nodes/NodeA/servers/server1/jta              javax.naming.Context
```

```
   57 (top)/nodes/NodeA/servers/server1/jta/usertransaction
   57                                                  java.lang.Object
   58 (top)/nodes/NodeA/servers/server1/cell           javax.naming.Context
   58    Linked to context: ITSOCell
   59 (top)/nodes/NodeA/servers/server1/wm             javax.naming.Context
   60 (top)/nodes/NodeA/servers/server1/wm/default
com.ibm.websphere.asynchbeans.WorkManager
   61 (top)/nodes/NodeA/servers/MyClusterServer1       javax.naming.Context
   61    Linked to URL: corbaloc::wan:9811/NameServiceServerRoot
   62 (top)/nodes/NodeA/servers/MyClusterServer2       javax.naming.Context
   62    Linked to URL: corbaloc::wan:9812/NameServiceServerRoot
   63 (top)/nodes/NodeA/servers/nodeagent             javax.naming.Context
   63    Linked to URL: corbaloc::wan:2809/NameServiceServerRoot
   64 (top)/clusters                                  javax.naming.Context
   65 (top)/clusters/MyCluster                        javax.naming.Context
   65    Linked to URL: corbaloc::wan:9811,:wan:9812/NameServiceServerRoot
   66 (top)/legacyRoot                                javax.naming.Context
   66    Linked to context: ITSOCell/persistent
   67 (top)/persistent                                javax.naming.Context
   68 (top)/persistent/cell                           javax.naming.Context
   68    Linked to context: ITSOCell
   69 (top)/cell                                      javax.naming.Context
   69    Linked to context: ITSOCell
   70 (top)/deploymentManager                         javax.naming.Context
   70    Linked to URL: corbaloc::wan:9809/NameServiceServerRoot
   71 (top)/cells                                     javax.naming.Context
```

The process-level name space contains:

- A full set of entries for objects registered in the local name space of the process.

  These entries include resources (JDBC, JMS, etc.) read from the resources.xml of the process, as well as those registered at runtime by applications, for example EJB homes.

- A corbaloc URL link to the local name space root (NameServiceServerRoot) of the node agent.

- A corbaloc URL link to the root of the deployment manager local name space (NameServiceServerRoot).

- A number of cross-links for the federated name space:
  - /cells
  - /clusters
  - /legacyRoot

- A link to the cell persistent root, /persistent/cell.

- A link to the node persistent root, /nodes/<nodename>/persistent.

Archive

13.3 Interoperable Naming Service (INS)

It is a requirement in J2EE 1.4 to provide a CosNaming service to support the EJB interoperability through the Interoperable Naming Service (INS). The INS allows J2EE application servers to deal with and understand names formulated according to the CORBA 2.3 naming scheme. The main advantage of INS is that it improves interoperability with other application server products, as well as CORBA servers. The naming architecture of WebSphere Application Server is compliant with the Interoperable Naming Service (INS). The requirements of INS CosNaming include:

- *corbaloc* and *corbaname* URLs must be supported, in addition to the IIOP URL supported in WebSphere Application Server V4.
  - Corbaloc designates an endpoint, such as a host machine.
  - Corbaname designates an object's name.
- The default bootstrap port must be 2809, as compared to the default of 900 used in earlier versions of IBM WebSphere Application Server.

13.3.1 Bootstrap ports

Every WebSphere Application Server V6 process, has a bootstrap server and port assignment.

Each process on a given machine and WebSphere logical node requires unique ports, including the bootstrap port. The default port assignments are:

- Application server

  The default for application server is 9810. Each subsequently created application server will be assigned a unique ascending port number that does not conflict.
- Network deployment

  The default for node agent is 2809. Application servers are each assigned a unique non-default port, either explicitly by the administrator, or automatically determined by the administration tool.

13.3.2 CORBA URLs

CORBA URL syntax, both corbaloc and corbaname, is supported by IBM WebSphere Application Server.

corbaloc

The corbaloc form of the CORBA 2.3 URL has the following syntax:

```
corbaloc:<protocol>:<addresslist>/<key>
```


Archived

Table 13-1 corbaloc options

Setting	Description
protocol	The protocol used for the communication. Currently, the only valid value is iiop.
addresslist	List of one or more addresses (host name and port number). The addresses are separated by commas, and each address has a colon prefix.
key	Define the type of root to access. See Table 13-5 on page 795 for further information.

The following list illustrates how the corbaloc URL can range from simple to complex, depending upon whether fault tolerance (request retry with second, third, and so on, server) is required:

- Basic

  ```
  corbaloc::myhost
  ```

- Cell's name space root from a specific server

  ```
  corbaloc:iiop:1.2@myhost.raleigh.ibm.com:9344/NameServiceCellRoot
  ```

- Server name space root with fault tolerance

  ```
  corbaloc::myhost1:9333,:myhost2:9333,:myhost2:9334/NameServiceServerRoot
  ```

Note: corbaloc URLs are usually used for the provider URL when retrieving an InitialContext.

corbaname

A corbaname can be useful at times as a lookup name. If, for example, the target object is not a member of the federated name space and cannot be located with a qualified name, a corbaname can be a convenient way to look up the object.

The corbaloc form of the CORBA 2.3 URL has the following syntax:

```
corbaname:<protocol>:<addresslist>/<key>#<INS string-formatted-name>
```

Table 13-2 corbaname options

Setting	Description
protocol	Use this protocol used the communication. Currently, the only valid value is iiop.
addresslist	This is a list of one or more addresses (host name and port number). The addresses are separated by commas, and each address has a colon prefix.

Setting	Description
key	Define the type of root to access. See Table 13-5 on page 795 for details.
<INS string-formatted-name>	This is the fully qualified path to entry under the specific root context.

The following examples illustrate how the corbaname URL can range from simple to complex, depending upon whether fault tolerance, request retry with second, third, etc. server, is required.

- Fully qualified name access

  ```
  corbaname::myhost:9333#cell/nodes/node1/servers/server5/someEjb
  ```

- Object access through a specific server root

  ```
  corbaname::myhost:9333/NameServiceServerRoot/someEjb
  ```

Note: corbaname URLs are usually used when performing a direct URL lookup using a previously obtained InitialContext, for example ic.lookup("urlstring").

13.4 Distributed CosNaming

One of the advantages of the distributed nature of CosNaming in WebSphere Application Server is that it removes the bottleneck of having a single name server for all naming lookups. Each WebSphere process, such as deployment manager, node agent and application server, hosts its own ORB, NameService and local name space. Lookups are made by accessing the NameService in the most convenient process. They are not bottlenecked through a single server process in the cell.

The WebSphere Application Server naming architecture uses CORBA CosNaming as its foundation. The CosNaming architecture has been changed to support a distributed and federated collection of CosNaming servers. Each deployment manager, node agent, and application server is a CosNaming server and is responsible for managing the names of the objects that are bound locally. Objects are bound into the local context. Lookups start in the local process and end in the process where the target object is located. This reduces the dependency of the WebSphere Application Server network on a single name server for all lookups.

A single, logical name space exists across the cell. The separate name spaces of each server process are linked and federated via context links in the cell name space. It is possible to navigate to specific subcontexts, as every server cluster and non-clustered server has its own context stored in a level of the cell name space.

The contents of the federated name space are mostly transient, built from configuration data read from the XML configuration files on the startup of each server process. Persistent roots are provided at the cell and node level of the name space to provide locations where objects can be persistently bound. These bindings are persisted to XML files on the file system.

Each separate server process has its own bootstrap port, thereby reducing bottlenecks.

13.5 Configured bindings

With the configured bindings feature you can add objects to the name space using the administrative interfaces. This feature allows an administrator to explicitly add bindings to the cell name space without having to write code. The administrator configures an alias in a persistent name space that refers to a real reference in one of the local name spaces, thus providing an additional level of indirection for names. (The configuration details are covered later in 13.11.1, “Name space bindings” on page 811.)

The functionality is useful in these areas:

- Federation of name spaces

  As long as it is CORBA 2.3 compliant, supporting INS, the name space of other WebSphere Application Server V6 or V5 cells, WebSphere Application Server V4 administrative domains, third-party application servers and even CORBA servers can be federated into the cell’s name space.

- Interoperability with WebSphere Application Server V4

  The default context of WebSphere Application Server V4 clients is the global, or legacy, context. However, WebSphere Application Server V6 processes bind their objects in local, transient name spaces. Therefore, WebSphere Application Server V4 clients looking up and accessing objects in WebSphere Application Server V6 without requiring changes to the client, requires the WebSphere Application Server V6 object to be bound to the legacy name space accessible to the client. Enter configured bindings. An alias can be configured into the legacy name space. When used by the WebSphere Application Server V4 client, the client is transparently redirected to the real object reference in one of the cell’s local name spaces.

Archive

13.5.1 Types of objects

The following types of objects can be bound using configured bindings:

- EJB hosted by a server in the cell

  The configured binding identifies an EJB home based on its configured JNDI name and the server in which it is deployed.

  A possible use of this is to put a binding for an EJB into the cell-scoped name space so that a lookup can be done without knowledge about the server in which the EJB is deployed. This mechanism is useful for allowing WebSphere Application Server V4 clients to look up WebSphere Application Server V6 EJBs without having to redeploy.

- CORBA object

  The configured binding identifies a CORBA object bound somewhere in this or another name space by using a corbaname URL string. Included is also an indicator of whether the object is a CosNaming NamingContext, in which case the binding is a federated link from one name space to another.

- JNDI name

  The configured binding identifies a provider URL and a JNDI name that can be used to look up an object. This can be used to reference a resource or other Java serialized object bound elsewhere in this name space or another name space.

- String constant

  The string constant can be used to bind environment data into the name space.

13.5.2 Types of binding references

There are several different references that can be specified for configured bindings. Valid types are summarized in Table 13-3.

Archive

Table 13-3 Types of binding reference

Binding type	Required Settings
EJB (EjbNameSpaceBinding)	1. The binding identifier is the name that uniquely identifies this configured binding. 2. The name in name space is relative to the configured root. 3. The JNDI is the name of EJB. 4. Use the server or server cluster where the EJB is deployed.
CORBA (CorbaObjectNameSpaceBinding)	1. The binding identifier is the name that uniquely identifies this configured binding. 2. The name in name space is relative to configured root. 3. Use the corbaname URL. 4. It is an indicator if the target object is a federated context object, or a leaf node object.
Indirect (IndirectLookupNameSpaceBinding)	1. The binding identifier is the name that uniquely identifies this configured binding. 2. The name in name space is relative to configured root. 3. Use the Provider URL. 4. Use the JNDI name of object.
String (StringNameSpaceBinding)	1. The binding identifier is the name that uniquely identifies this configured binding. 2. The name in name space is relative to configured root. 3. Set the constant string value.

The configured bindings can be relative to one of the following context roots:

- Server root
- Node persistent root
- Cell persistent root

Archive

13.6 Initial contexts

In WebSphere, an initial context for a name server is associated with a bootstrap host and bootstrap port. These combined values can be viewed as the address of the name server owning the initial context. To get an initial context, you must know the bootstrap host and port for the initial context's name server.

JNDI clients should assume the correct environment is already configured, so there is no need to explicitly set property values and pass them to the InitialContext constructor.

However, a JNDI client might need to access a name space other than the one identified in its environment. In this case, it is necessary to explicitly set the javax.naming.provider.url (provider URL) property used by the InitialContext constructor. A provider URL contains bootstrap server information that the initial context factory can use to obtain an initial context. Any property values passed directly to the InitialContext constructor take precedence over settings of those same properties found elsewhere in the environment.

Two provider URL forms can be used with WebSphere's initial context factory:

- CORBA object URL
- IIOP URL

CORBA object URLs are more flexible than IIOP URLs and are the recommended URL format to use. CORBA object URLs are part of the OMG CosNaming Interoperable Naming Specification. The IIOP URLs are the legacy JNDI format, but are still supported by the WebSphere initial context factory. The examples in the following sections illustrate the use of these URLs.

Using a CORBA object URL

An example of using a corbaloc URL with a single address to obtain an initial context is shown in Example 13-4.

Example 13-4 Initial context using CORBA object URL

```
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.InitialContext;

Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
"com.ibm.websphere.naming.WsnInitialContextFactory");
env.put(Context.PROVIDER_URL, "corbaloc:iiop:myhost.mycompany.com:2809");

Context initialContext = new InitialContext(env);
```

Using a CORBA object URL with multiple addresses

CORBA object URLs can contain more than one bootstrap server address. This feature can be used in WebSphere when attempting to obtain an initial context from a server cluster. The bootstrap server addresses for all servers in the cluster can be specified in the URL. The operation will succeed if at least one of the servers is running, eliminating a single point of failure.

Note: There is no guarantee of any particular order in which the address list will be processed. For example, the second bootstrap server address might be used to obtain the initial context even though the first bootstrap server in the list is available.

An example of using a corbaloc URL with multiple addresses to obtain an initial context is shown in Example 13-5 on page 792.

Example 13-5 Initial context using CORBA object URL with multiple addresses

```
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.InitialContext;

Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
"com.ibm.websphere.naming.WsnInitialContextFactory");
env.put(Context.PROVIDER_URL,
"corbaloc::myhost1:2809,:myhost2:2809,:myhost3:2809");

Context initialContext = new InitialContext(env);
```

Using a CORBA object URL from a non-WebSphere JNDI

To access a WebSphere name server from a non-WebSphere environment, such that the WebSphere initial context factory is not used, a corbaloc URL must be used that has an object key of *NameServiceServerRoot* to identify the server root context.

The server root is where system artifacts such as EJB homes are bound. The default key of NameService can be used when fully qualified names are used for JNDI operations.

Example 13-6 shows a CORBA object type URL from a non-WebSphere JNDI implementation. It assumes full CORBA object URL support by the non-WebSphere JNDI implementation.

Example 13-6 Using a CORBA object URL from non-WebSphere JNDI

```
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.InitialContext;

Hashtable env = new Hashtable(); env.put(Context.INITIAL_CONTEXT_FACTORY,
"com.somecompany.naming.TheirInitialContextFactory");
env.put(Context.PROVIDER_URL,
"corbaname:iiop:myhost.mycompany.com:2809/NameServiceServerRoot");

Context initialContext = new InitialContext(env);
```

Using an IIOP URL

The IIOP type of URL is a legacy format that is not as flexible as CORBA object URLs. However, URLs of this type are still supported by the WebSphere initial context factory.

Example 13-7 shows an IIOP type URL as the provider URL.

Example 13-7 Initial context using an IIOP URL

```
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.InitialContext;

Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
"com.ibm.websphere.naming.WsnInitialContextFactory");
env.put(Context.PROVIDER_URL, "iiop://myhost.mycompany.com:2809");

Context initialContext = new InitialContext(env);
```

13.6.1 Setting initial root context

Each server contains its own server root context. When bootstrapping to a server, the server root is the default initial JNDI context. Most of the time, this is the desired initial context, because system artifacts such as EJB homes are bound at this point. However, other root contexts exist that might contain bindings of interest. It is possible to specify a provider URL to select other root contexts.

Note: The default is that the name will be resolved based upon the context associated with the server bootstrap to which the client is connected.

Archived

The initial root context can be selected using the following settings:

- CORBA object URL
- Name space root property

Default initial context

The default initial context depends on the type of client. Table 13-4 summarizes the different categories of clients and the corresponding default initial context.

Table 13-4 Default initial context versus client type

Client type	Description	Default initial context
WebSphere Application Server V6 or V5 JNDI	EJB applications use the JNDI interface to perform name space lookups. WebSphere clients by default use WebSphere's CosNaming JNDI plug-in implementation.	Server root
WebSphere Application Server V4 JNDI	WebSphere clients running in releases prior to V5 by default use WebSphere's V4 CosNaming JNDI plug-in implementation.	Cell persistent root (legacy root)
Other JNDI	Some applications might perform name space lookups with a non-WebSphere CosNaming JNDI plug-in implementation.	Cell root
CORBA	Standard CORBA client obtains an initial org.omg.CosNaming.NamingContext reference with the key NamingContext.	Cell root

Selecting initial root context with a CORBA object URL

There are several object keys registered with the bootstrap server that you can use to select the root context to be used as the initial context. To select a particular root context with a CORBA object URL object key, set the object key to the corresponding value. The default object key is NameService. Using JNDI, this will yield the server root context.

Table 13-5 lists the different root contexts and their corresponding object key.

Archived

Table 13-5 CORBA object URL root context values

Root context	CORBA object URL object key	Description
Server root	NameServiceServerRoot	Server root for the accessed server
Cell persistent root	NameServiceCellPersistentRoot	The persistent cell root for the accessed server
Cell root	NameServiceCellRoot	The cell root for the accessed server
Node root	NameServiceNodeRoot	The node root for the accessed server

Note: The name server running in the deployment manager process has no node root registered under the NameServiceNodeRoot key, because there is no node agent, nor application servers, running in its node.

Example 13-8 shows the use of a corbaloc URL with the object key set to select the cell persistent root context as the initial context.

Example 13-8 Select cell persistent root context using corbaloc URL

```
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.InitialContext;

Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
"com.ibm.websphere.naming.WsnInitialContextFactory");
env.put(Context.PROVIDER_URL,
"corbaloc:iiop:myhost.mycompany.com:2809/NameServiceCellPersistentRoot");

Context initialContext = new InitialContext(env);
```

Selecting initial root context with name space root property

You can select the initial root context by passing a name space root property setting to the InitialContext constructor. Generally, the object key setting is sufficient.

Sometimes, a property setting might be preferable. For example, the root context property can be set on the Java invocation to make it transparent to the application which server root is being used as the initial context. The default server root property setting is *defaultroot*, which will yield the server root context.

Tip: If a simple name is used, the root context that will be assumed can be set by passing the `com.ibm.websphere.naming.namespaceroot` property to InitialContext.

Table 13-6 Name space root values

Root context	CORBA object URL object key
Server root	bootstrapserverroot
Cell persistent root	cellpersistentroot
Cell root	cellroot
Node root	bootstrapnoderoot

The name space root property is used to select the default root context only if the provider URL does not contain an object key or contains the object key, *NameService*. Otherwise, the property is ignored.

Example 13-9 shows use of the name space root property to select the cell persistent root context as the initial context.

Tip: WebSphere makes available constants that can be used instead of hard-coding the property name and value, for example:

```
env.put(PROPS.NAME_SPACE_ROOT, PROPS.NAME_SPACE_ROOT_CELL_PERSISTENT);
```

Example 13-9 Use of name space root property to select cell persistent root context

```
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.InitialContext;
import com.ibm.websphere.naming.PROPS;

Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
"com.ibm.websphere.naming.WsnInitialContextFactory");
env.put(Context.PROVIDER_URL, "corbaloc:iiop:myhost.mycompany.com:2809");
env.put(PROPS.NAME_SPACE_ROOT, PROPS.NAME_SPACE_ROOT_CELL_PERSISTENT);

Context initialContext = new InitialContext(env);
```

13.7 Federation of name spaces

Federating name spaces involves binding contexts from one name space into another name space. In a WebSphere Application Server V6 name space, federated bindings can be created with the following restrictions:

- Federation is limited to CosNaming name servers. A WebSphere name server is a CORBA CosNaming implementation.

  Federated bindings to other CosNaming contexts can be created, but bindings to LDAP name server implementation contexts cannot.
- If JNDI is used to federate the name space, the WebSphere initial context factory must be used to obtain the reference to the federated context. If any other initial context factory implementation is used, the binding might not be created, or the level of transparency might be reduced.
- A federated binding to a non-WebSphere naming context has the following functional limitations:
  - JNDI operations are restricted to the use of CORBA objects. For example, EJB homes can be looked up, but non-CORBA objects such as data sources cannot.
  - JNDI caching is not supported for non-WebSphere name spaces. This only affects the performance of lookup operations.
- Do not federate two WebSphere single server name spaces. If this is done, incorrect behavior can result. If you require federation of WebSphere name spaces, then servers running under IBM WebSphere Application Server Network Deployment are required.

In the example in Figure 13-3, assume that a name space, Namespace 1, contains a context under the name *a/b*. Also assume that a second name space, Namespace 2, contains a context under the name *x/y*. If context *x/y* in Namespace 2 is bound into context *a/b* in Namespace 1 under the name *f2*, the two name spaces are federated. Binding *f2* is a federated binding because the context associated with that binding comes from another name space. As shown in Figure 13-3, from Namespace 1, a lookup of the name *a/b/f2* would return the context bound under the name *x/y* in Namespace 2. Furthermore, if context *x/y* contained an EJB home bound under the name *ejb1*, the EJB home could be looked up from Namespace1 with the lookup name *a/b/f2/ejb1*. Notice that the name crosses name spaces. This fact is transparent to the naming client.

Archive

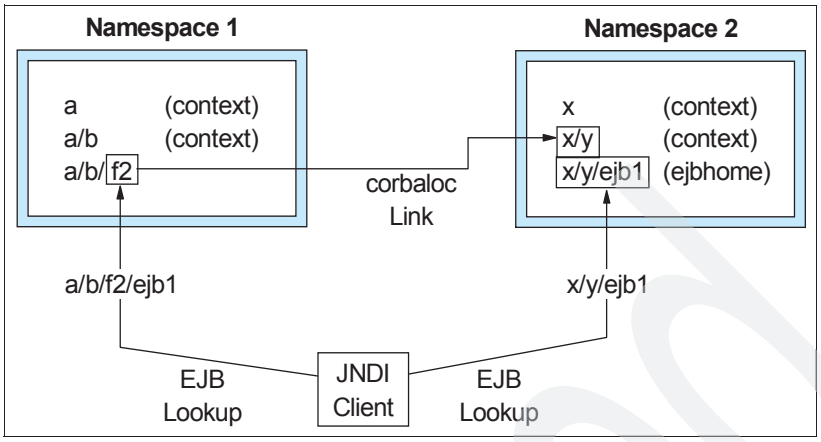


Figure 13-3 JNDI access using federated name spaces

13.8 Interoperability

The name space in IBM WebSphere Application Server V5 is the same as in WebSphere Application Server V6. Thus, an EJB client running on IBM WebSphere Application Server V5 accessing EJB applications running on WebSphere Application Server V6 will have no interoperability issues.

WebSphere Application Server V6 provides the following support for interoperating with previous releases of WebSphere and with non-WebSphere JNDI clients:

- EJB clients running on WebSphere V4.0.x, accessing EJB applications running on WebSphere Application Server V6
- EJB clients running on WebSphere Application Server V6, accessing EJB applications running on WebSphere V4.0.x servers
- EJB clients running in an environment other than WebSphere, accessing EJB applications running on WebSphere Application Server V6 servers

13.8.1 WebSphere V4.0 EJB clients

Applications migrated from previous WebSphere releases can still have clients running in a previous release. The default initial JNDI context for EJB clients running on previous versions of WebSphere is the cell persistent root, the legacy root. However, the home for an EJB deployed in V6.0 is bound to the server root context. For the EJB lookup name for down-level clients to remain unchanged, configure a binding for the EJB home under the cell persistent root.

Archived

The following options enable interoperability with WebSphere Application Server V4 clients:

- Set the client's default initial context to legacyRoot. This option is equivalent to the cell persistent root of WebSphere Application Server V6.
- Redeploy the clients using the Application Server Toolkit so that the JNDI names can be fixed to reflect the real, fully qualified names in the WebSphere Application Server V6 name space.
- Use aliases for the names the clients look up. These transparently redirect to the correct object in the WebSphere Application Server V6 name space. This option uses configured bindings.

Options for EJB lookup

The following options support EJB lookup from a WebSphere Application Server V4 client to a WebSphere Application Server V6 hosted EJB:

- Redeploy the WebSphere Application Server V4 client.

  Update the `<ejb-ref>` to reflect the WebSphere Application Server V6 compatible JNDI name.
- In WebSphere Application Server V6, configure EjbNameSpaceBinding:
  a. Use the same JNDI name as looked up by the WebSphere Application Server V4 client.
  b. Identify the JNDI name and server, or cluster, of the target EJB.
  c. Configure the binding in the cell persistent root.

Options for resources bound in external name space

Options for resources bound in external name spaces include the following:

- Redeploy the WebSphere Application Server V4 client.

  Update the `<resource-ref>` to reflect the WebSphere Application Server V6 compatible JNDI name.
- In WebSphere Application Server V6, run the program to bind the resource into the WebSphere Application Server V6 cell persistent root.
- In WebSphere Application Server V6, configure IndirectLookupNameSpaceBinding by doing the following:
  a. Use the same JNDI name as looked up by the WebSphere Application Server V4 client.

Archived

b. Specify the provider URL and JNDI name of the name space where the resource is already bound (a WebSphere Application Server V4 name space).

c. Configure the binding in the cell persistent root.

13.8.2 WebSphere V4.0 server

The default initial context for a WebSphere V4.0 server is the correct context. WebSphere Application Server V6 clients simply look up the JNDI name under which the EJB home is bound.

13.8.3 EJB clients hosted by non-WebSphere environment

When an EJB application running in WebSphere Application Server V6 is accessed by a non-WebSphere EJB client, the JNDI initial context factory is presumed to be a non-WebSphere implementation. In this case, the default initial context is the cell root. If the JNDI service provider being used supports CORBA object URLs, use the following corbaname format to look up the EJB home:

Example 13-10 corbaname format for EJB home lookup

```
initialContext.lookup("corbaname:iiop:myHost:2809#cell/clusters/myCluster/myEJB
");
```

According to the URL in Example 13-10, the bootstrap host and server (node agent) port are `myHost` and 2809. The EJB is installed in a server cluster named `myCluster`. The EJB is bound in that cluster under the name `myEJB`.

Note: The server name could also be the name of a non-clustered server. This form of lookup works in the following situations:

- With any name server bootstrap host and port configured in the same cell
- If the bootstrap host and port belong to a member of the cluster itself

To avoid a single point of failure, the bootstrap server host and port for each cluster member could be listed in the URL as shown in Example 13-11.

Example 13-11 corbaname format with multiple addresses for EJB home lookup

```
initialContext.lookup("corbaname:iiop:host1:9810,host2:9810#cell/clusters/myClu
ster/myEJB");
```

The name prefix `cell/clusters/<clustername>/` is not necessary if bootstrapping to the cluster itself, but it always works. The prefix is required, however, when looking up EJBs in other clusters. The server binding for the

prefix used to access another cluster is implemented in a way that avoids a single point of failure during a lookup.

If the JNDI initial context factory you use does not support CORBA object URLs, the initial context can be obtained from the server, and the lookup can be performed on the initial context as shown in Example 13-12.

Example 13-12 corbaname format with multiple addresses for EJB home lookup

```
Hashtable env = new Hashtable();
env.put(CONTEXT.PROVIDER_URL, "iiop://myHost:2809");
Context ic = new InitialContext(env);
Object o = ic.lookup("cell/clusters/myCluster/myEJB");
```

This form of lookup works from any server in the same cell as the EJB home being looked up. However, this approach does not allow multiple hosts and ports to be specified in the provider URL and does not incorporate the availability advantages of a corbaloc or corbaname URL with multiple hosts and ports belonging to the server cluster members.

13.9 Examples

The following examples highlight a number of different server topologies and the affect the topologies have on the use of the Naming Service:

- Single server
- Single server with a non-default port
- Two single servers on the same box
- Two Network Deployment application servers on the same box
- WebSphere Application Server V4 client

13.9.1 Single server

In the single-server environment, the naming functionality works in exactly the same way as in WebSphere Application Server V4. There is only one server and only one root context and, therefore, no ambiguity in the location of a named object. This is illustrated by the example in Figure 13-4.

Archived

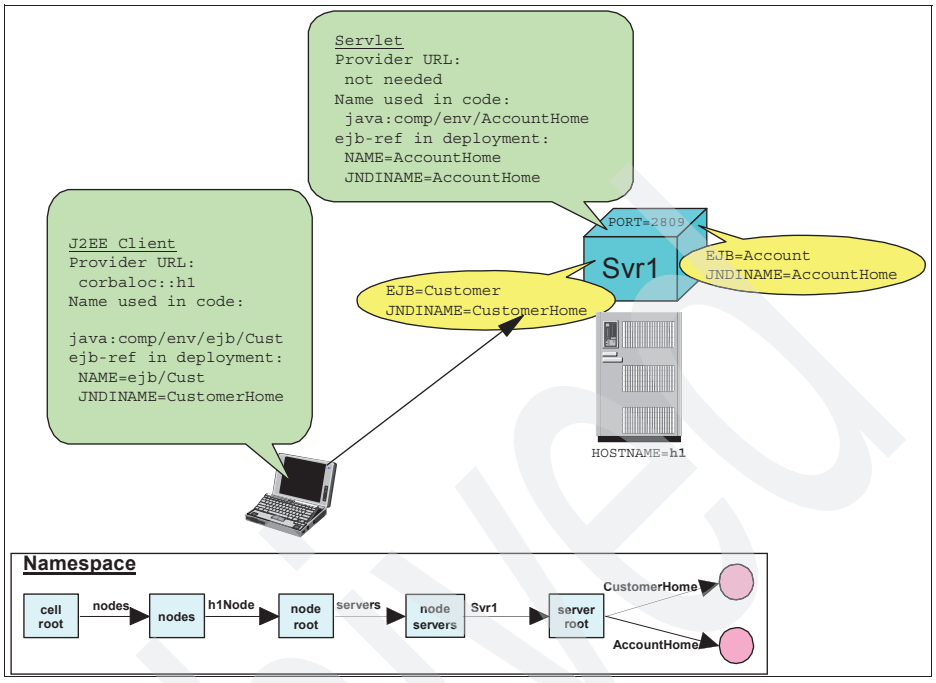


Figure 13-4 Single server

By accessing the server root directly, the J2EE or or servlet client does not need to traverse the cell name space (cell root → servers →server root → object).

Note: Even in a single-server case, clients can still use the fully qualified JNDI name to look up an object. This removes any dependency on the particular topology. However, there is a small performance degradation.

In a single-server, the server root acts as the default bootstrap, and should be assigned port 2809. Clients external to the server process using the provider URL do not need a port number.

If the named object is looked up by a client running in the same process, then a provider URL, and corbaloc, is not needed. By default, the lookup is performed against the local process name space. Table 13-7 illustrates the Provider URL.

Table 13-7 Lookup settings required for a single server

Component	Provider URL	JNDI name
Servlet (same process)	*Not needed*	CustomerHome

Component	Provider URL	JNDI name
J2EE client (external process)	corbaloc::<hostname>	CustomerHome

13.9.2 Two single servers on the same box

When more than one instance of the application server runs on a single machine, then you must configure each server's bootstrap to run on a different port. In this case, you can have a J2EE component in one server looking up objects in the other server. This is illustrated by the example in Figure 13-5.

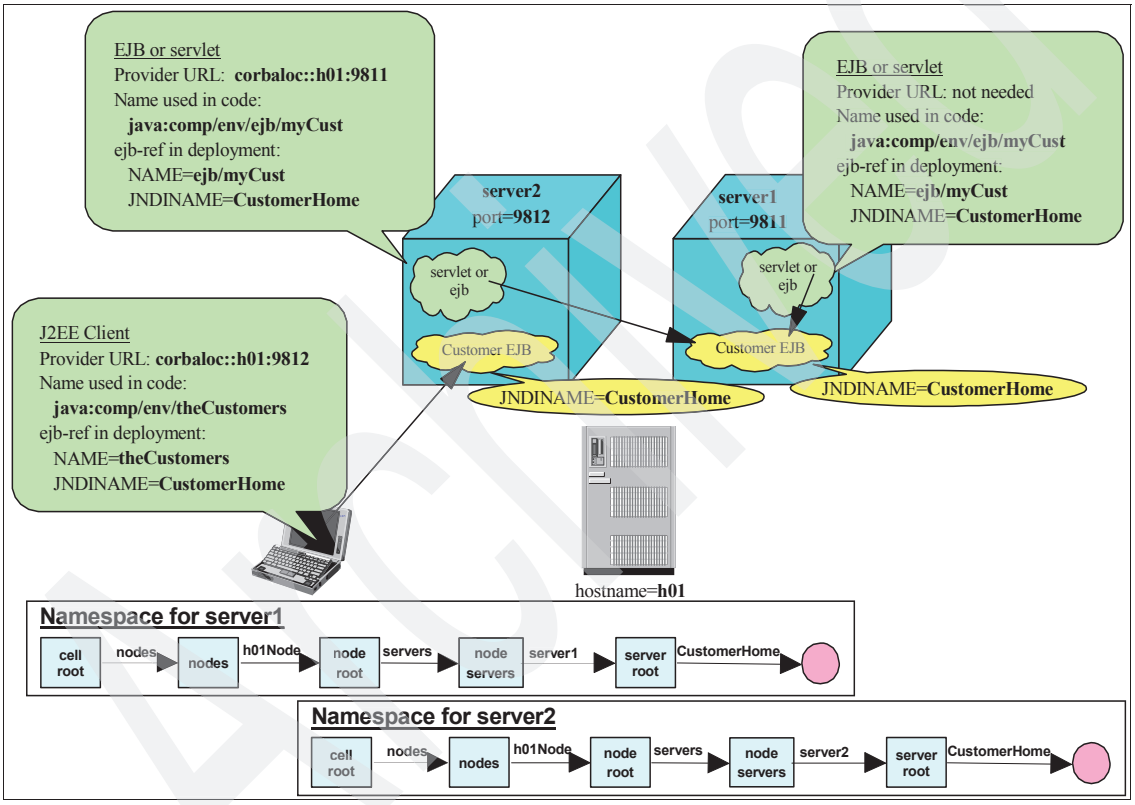


Figure 13-5 Two single servers on the same box

Because each application server name space is separate, the different objects can use the same name, CustomerHome. There is no name collision. The fully qualified JNDI name can be used to uniquely identify the name registered in one server from the name in another. If objects are registered under the name CustomerHome on two servers, look up the name using:

```
cell/nodes/<nodename>/servers/<server1>/CustomerHome
cell/nodes/<nodename>/servers/<server2>/CustomerHome
```

Table 13-8 illustrates the required Provider URL settings.

Table 13-8 Lookup settings for two single servers on the same box

Component	Provider URL	JNDI name
Servlet (same process)	*Not needed*	CustomerHome
Servlet (external process)	corbaloc::<hostname>:<port#>	CustomerHome
J2EE client (external process)	corbaloc::<hostname>:<port#>	CustomerHome

13.9.3 Network Deployment application servers on the same box

The configuration becomes more complex when we move from an standalone server environment to a Network Deployment distributed server environment. In this topology, there can be separate application servers as well as a node agent process, all of which have a bootstrap port and host a local name space:

- The node agent is the default bootstrap for the node, and has its bootstrap port configured on 2809.
- The application servers are not the default bootstrap, and, therefore, each is configured to use a non-default bootstrap port.

This concept is illustrated by the example in Figure 13-6 on page 805.

Archive

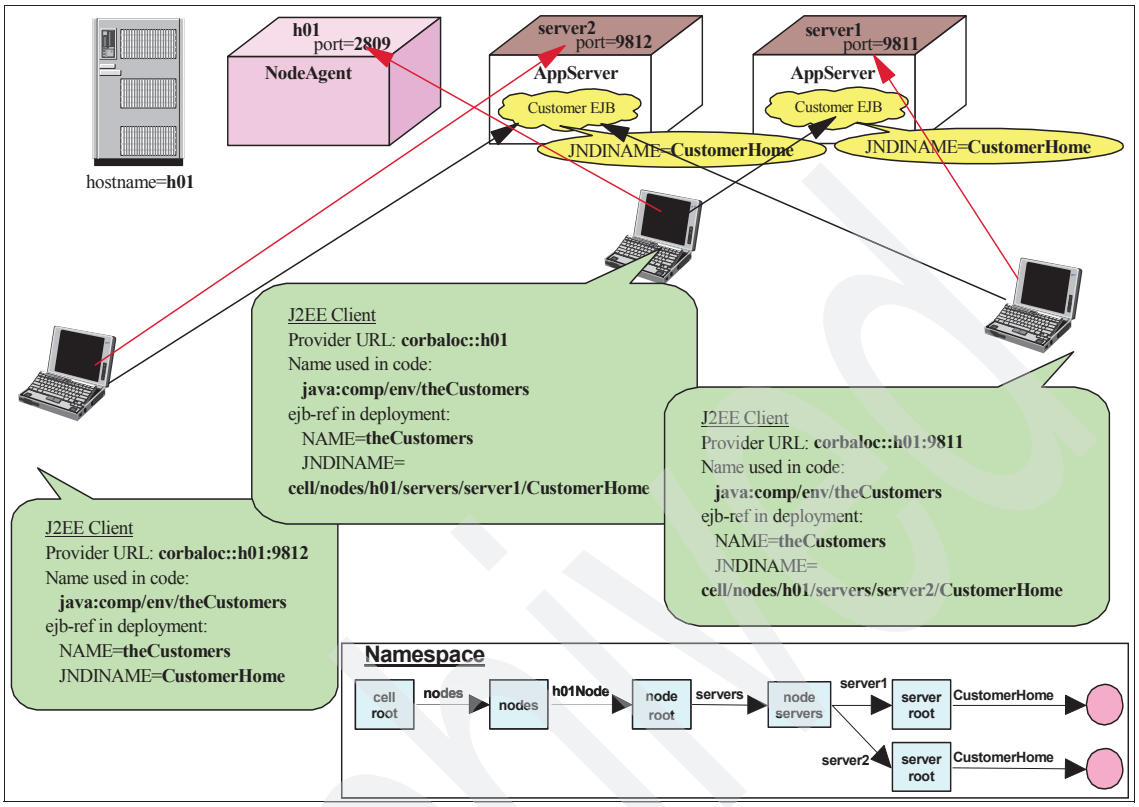


Figure 13-6 Two network deployment application servers on the same box

Unless a client uses a specific application server in its provider URL, the lookup is performed on the node agent. For the lookup to succeed, the bindings have to specify the fully qualified name of the object:

```
cell/nodes/<nodename>/servers/<servername>/<name of object>
```

That is, the client needs to specify where the object is located. This is a big difference from the behavior in WebSphere Application Server V4, where all named objects were registered in a single global name space.

Tip: When you need server clusters for high availability, bootstrap to a server cluster so that the initial context has failover support. Lookups which resolve to other clusters from that bootstrap cluster also have failover support from the name server implementation. The provider URL should have the bootstrap address of each cluster member to avoid a single point of failure when obtaining the initial context.

In a distributed server environment, choose a bootstrap server which has a stable bootstrap address, such as a designated cluster, server, or node agent.

Table 13-9 illustrates the Provider URL settings required.

Table 13-9 Lookup settings for two Network Deployment servers on the same box

Component	Provider URL	JNDI name
Servlet (same process)	*Not needed*	CustomerHome
Servlet (external process accessing local name space to access local object)	*Not needed*	cell/nodes/<nodename> /servers/<server2>/CustomerHome
Servlet (external process accessing other appserver's name space to access object on that appserver)	corbaloc::<appserver2 hostname>:<port#>	CustomerHome
	(or) Not needed	cell/persistent/CustomerHome2[1]
J2EE client (external process accessing appserver1 with object located on appserver1)	corbaloc::<appserver hostname>:<port#>	CustomerHome
	(or) Not needed	cell/persistent/CustomerHome1[1]
J2EE client (external process accessing node agent)	corbaloc::<node agent hostname>	cell/nodes/<nodename> /servers/<server2>/CustomerHome
	(or) Not needed	cell/persistent/CustomerHome2[1]
J2EE client (external process accessing appserver1 with object located on appserver2)	corbaloc::<appserver1 hostname>:<port#>	cell/nodes/<nodename> /servers/<server2>/CustomerHome
	(or) Not needed	cell/persistent/CustomerHome2[1]

[1] You must manually configure indirect JNDI references to the respective EJB in the cell/persistent name space.

Archived

13.9.4 WebSphere Application Server V4 client

In WebSphere Application Server V4, there is no need to specify a path to a named object, because all objects are registered in a single global name space. Although convenient, this causes naming conflicts because no two objects can be registered across all application servers with the same names.

The use of configured bindings, aliases, in the cell persistent root provides a mechanism by which the V4 naming structure can be mapped to the fully qualified names of V6. This is illustrated in Figure 13-7.

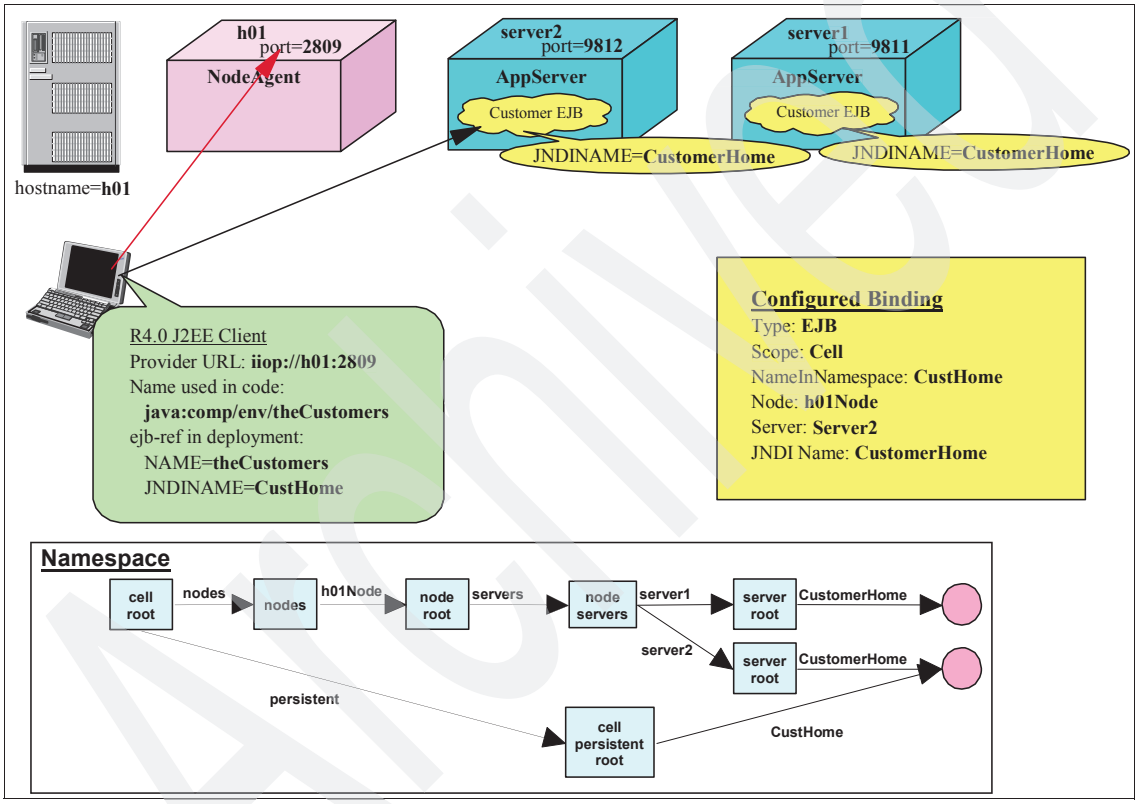


Figure 13-7 WebSphere Application Server V4 client

Table 13-10 illustrates the Provider URL settings required.

Table 13-10 Lookup settings for WebSphere Application Server V4 client interoperability

Component	Provider URL	JNDI name
V4 client	iiop://<hostname>:2809	CustHome

CustHome is the name registered in the cell-level persistent root, the legacy root, for cell/nodes/<nodename>/servers/<servername>/CustomerHome.

The WebSphere Application Server V4 client accesses the JNDI alias registered in the cell persistent root of the WebSphere Application Server V6 cell. The WebSphere Application Server V6 runtime *transparently* redirects the client to the JNDI entry located in a specific local name space hosted by one of the name servers of the cell.

13.10 Naming tools

IBM WebSphere Application Server provides the following tools for the support of the naming architecture.

13.10.1 dumpNameSpace

Run the **dumpNameSpace** command against any bootstrap port to get a listing of the names bound with that provider URL.

The output of the command:

- Does not present a full logical view of the name space
- Shows CORBA URLs where the name space transitions to another server

The tool indicates that certain names point to contexts external to the current server and its name space. The links show the transitions necessary to perform a lookup from one name space to another.

Tip: An invocation of the dumpNameSpace command cannot generate a dump of the entire name space, only the objects bound to the bootstrap server and links to other local name spaces that compose the federated name space. Use the correct host name and port number for the server to be dumped.

Syntax

To run the dumpNameSpace command, type the following:

```
dumpNameSpace [options]
```

All arguments are optional. Table 13-11 on page 809 shows the available options.

Table 13-11 Options for dumpNameSpace

Option	Description
-host <hostname>	This option is the host name of bootstrap server. If it is not defined, then the default is localhost.
-port <portnumber>	This option is the bootstrap server port number. If i is not defined, then the default is 2809.
-factory <factory>	This option is the initial context factory to be used to get initial context. The default of com.ibm.websphere.naming.WsnInitialContextFactory is ok for most use.
-root [cell | server | node | host | legacy | tree | default]	WebSphere V5.0 or later ► cell: dumpNameSpace default. Dump the tree starting at the cell root context. ► server: Dump the tree starting at the server root context. ► node: Dump the tree starting at the node root context. (Synonymous with "host") WebSphere V4.0 ► legacy: dumpNameSpace default. Dump the tree starting at the legacy root context. ► host: Dump the tree starting at the bootstrap host root context (Synonymous with node) ► tree: Dump the tree starting at the tree root context. All WebSphere and other name servers ► default: Dump the tree starting at the initial context which JNDI returns by default for that server type. This is the only -root choice that is compatible with WebSphere servers prior to V4.0 and with non-WebSphere name servers.
-url <url>	This option is the value for the java.naming.provider.url property used to get the initial JNDI context. This option can be used in place of the -host, -port, and -root options. If the -url option is specified, the -host, -port, and -root options are ignored.
-startAt <context>	This option is the path from the requested root context to the top level context where the dump should begin. Recursively dumps subcontexts below this point. Defaults to empty string, that is, root context requested with the -root option.
-format <format>	► jndi: Display name components as atomic strings. ► ins: Display name components parsed against INS rules (id.kind). The default format is jndi.

Archived

Option	Description
-report <length>	► short: Dumps the binding name and bound object type, essentially what JNDI Context.list() provides. ► long: Dumps the binding name, bound object type, local object type, and string representation of the local object. In other words, IORs, string values, and so on, are printed. The default report option is short.
-traceString <tracespec>	Trace string of the same format used with servers, with output going to the file DumpNameSpaceTrace.out.
-help or -?	Prints a usage statement.

Finding the bootstrap address

To find the bootstrap address for node agents, servers, and the cell, do the following:

- For application servers, click **Servers** →**Application Servers**. Click on the server to open the configuration. Select **Ports** from the Communications section, then **BOOTSTRAP_ADDRESS**.
- For node agents, click **System Administration** →**Node Agents**. Select the node agent to open the configuration. Select **Ports** from the Additional Properties section, then **BOOTSTRAP_ADDRESS**.
- For the cell, click **System Administration** →**Deployment Manager**. Select **Ports** from the Additional Properties section, then **BOOTSTRAP_ADDRESS**.

To find the dumpNameSpace usage, see Example 13-13.

Example 13-13 dumpNameSpace usage

```
$ cd c:\ibm\was60\AppServer\bin

Get help on options:
$ dumpNameSpace -?

Dump server on localhost:2809 from cell root:
$ dumpNameSpace

Dump server on localhost:2806 from cell root:
$ dumpNameSpace -port 2806

Dump server on yourhost:2811 from cell root:
$ dumpNameSpace -port 2811 -host yourhost

Dump server on localhost:9810 from server root:
$ dumpNameSpace -root server‘
```

```
Dump server at corbaloc
dumpNameSpace -url corbaloc:iiop:yourhost:901
```

13.11 Configuration

This section discusses how to configure a name binding for an enterprise bean, a CORBA CosNaming naming context or CORBA leaf node object, an object that can be looked up using JNDI, or a constant string value using the administrative console.

13.11.1 Name space bindings

The configured bindings feature allows objects to be added to the name space using the administrative console. An administrator can now explicitly add bindings to the cell name space without having to write code. With this feature, an administrator can configure an alias in a persistent name space for a reference in one of the local name spaces.

Name space bindings can be created for the following four object types:

- String
- EJB
- CORBA
- Indirect

As an example, look at Figure 13-7. In this scenario, an alias is configured to allow an application using the WebSphere V4 naming style to access an EJB while running on WebSphere V6. Because the V4 application code does not specify a path to the named object, a binding is added to the cell persistent root to redirect the client to the JNDI entry in the local name space.

Archive

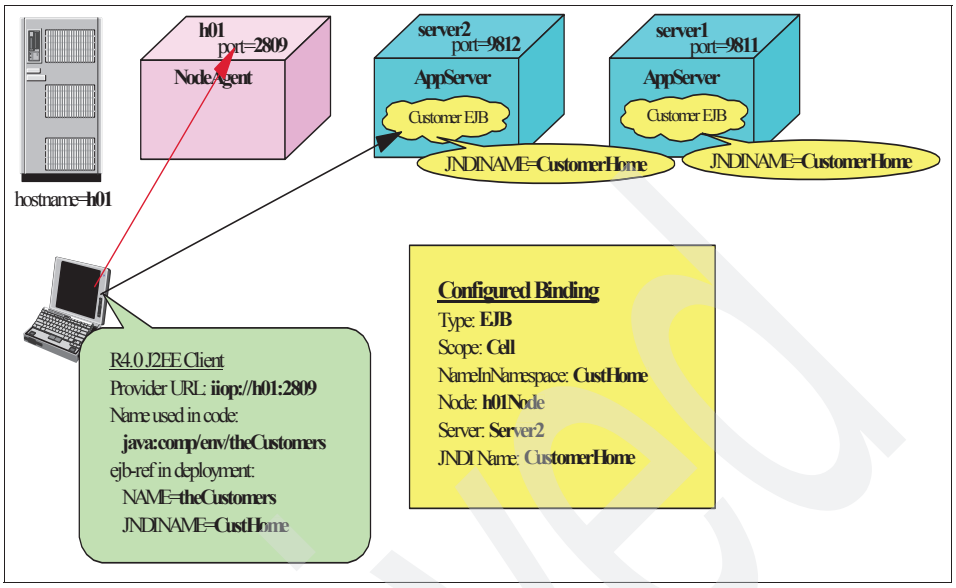


Figure 13-8 WebSphere Application Server V4 client

To create the binding, do the following:

1. Select **Environment →Naming → Name Space Bindings**.
2. Set the scope to `cell`.
3. Click **New**. See Figure 13-9 on page 813.

Archive

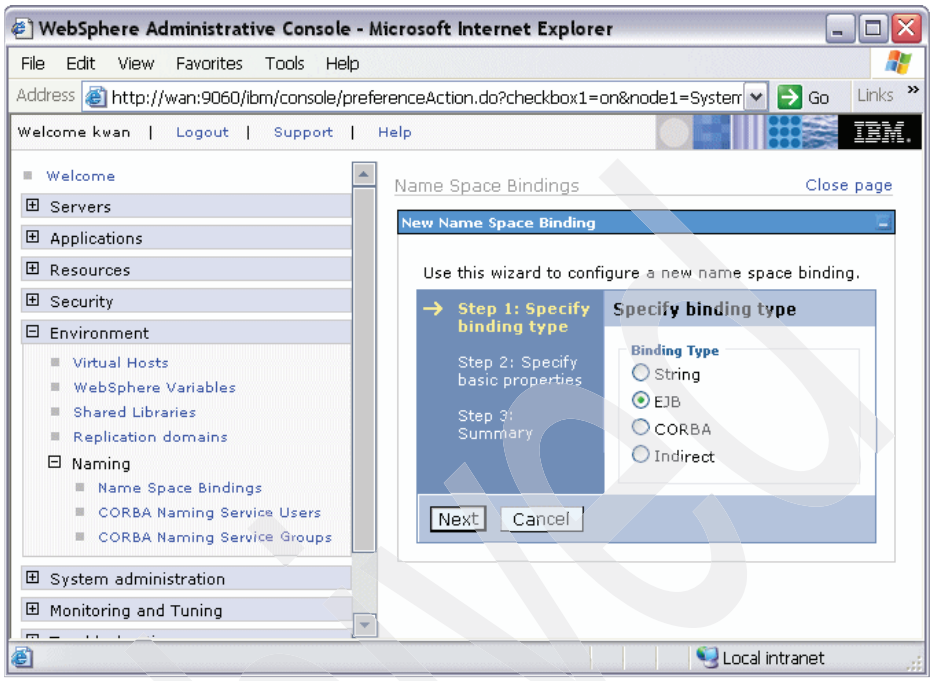


Figure 13-9 Name space binding

4. Choose **EJB** and click **Next.**
5. Enter the values as shown in Figure 13-10 on page 814.
   - **Binding identifier** is a unique identifier for the binding.
   - **Name in Name Space** matches the JNDI name used in the application code.
   - **Enterprise Bean Location** is the cluster or node where the EJB resides.
   - **Server** is the name of the server where the EJB resides.
   - **JNDI Name** is the the JNDI name of the deployed EJB. Use the name in the enterprise beans bindings, not the java:comp name.

Archive

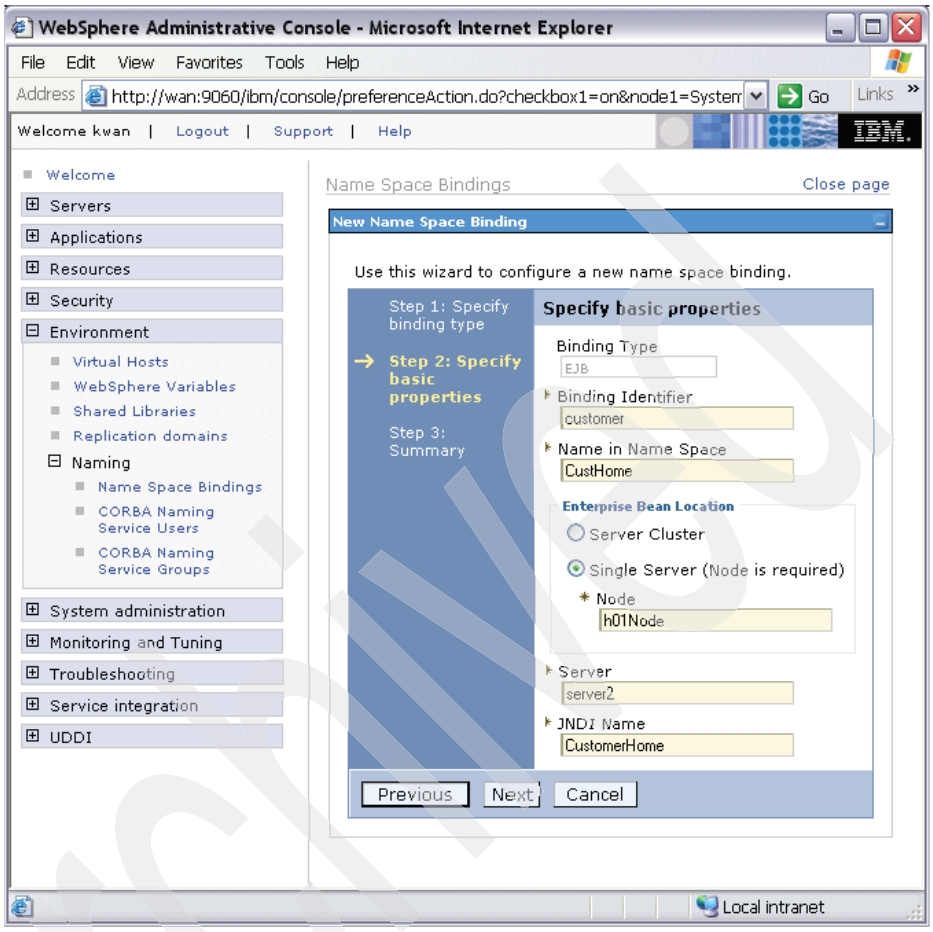


Figure 13-10 Defining an EJB name space binding

6. Click **Next**.
7. Click **Finish** and save your changes.

Note: Name space bindings can be configured at the cell, node, and server scope:

- Bindings configured at the cell scope are included in the local runtime name space of all application servers in that cell.
- Bindings configured at the node scope area included in the local runtime name space of all application servers in that node.
- Bindings configured at the server scope are included in the local runtime name space of only that application server.

13.11.2 CORBA naming service users and groups

The J2EE role-based authorization concept has been extended to protect the WebSphere CosNaming service. CosNaming security offers increased granularity of security control over CosNaming functions, which affect the content of the WebSphere name space. There are generally two ways in which client programs will make a CosNaming call. The first is through the JNDI interfaces. The second is CORBA clients invoking CosNaming methods directly.

Note: The authorization policy is only enforced when global security is enabled. Before enabling global security, you should design your entire security solution. See *WebSphere Application Security V6 Security Handbook,* SG24-6316 for information about designing and implementing WebSphere security.

You can design authorization based on users and groups of users defined to the active user registry. Design the authorization by assigning an authority level to one of the following:

- User
- Group
- ALL_AUTHENTICATED (special subject that acts as a group)

  This means any user who authenticates by entering a valid user ID and password.
- EVERYONE (special subject that acts as a group)

  All users are authorized. No authentication is necessary.

The roles now have authority level from low to high as follows:

- Users assigned the **CosNamingRead** role are allowed to do queries of the WebSphere Name Space, such as through the JNDI lookup method. The special subject “Everyone” is the default policy for this role.
- Users assigned to the **CosNamingWrite** role are allowed to do write operations such as JNDI bind, rebind, or unbind, plus CosNamingRead operations. The special subject All_Authenticated is the default policy for this role.
- Users assigned to the **CosNamingCreate** role are allowed to create new objects in the Name Space through such operations as JNDI createSubcontext, plus CosNamingWrite operations. The special subject, All_Authenticated, is the default policy for this role.

- Users assigned to the **CosNamingDelete** role are able to destroy objects in the Name Space, for example using the JNDI destroySubcontext method, as well as CosNamingCreate operations.

By default, you have the following:

- The ALL_AUTHENTICATED group has the following role privileges: CosNamingRead, CosNamingWrite, CosNamingCreate, and CosNamingDelete.
- The EVERYONE group has CosNamingRead privileges only.

Working with the CORBA naming service authorization is straightforward.

Working with CORBA naming service users

To work with users, do the following:

1. Select **Environment** →**Naming** →**CORBA Naming Service Users**. See Figure 13-11.

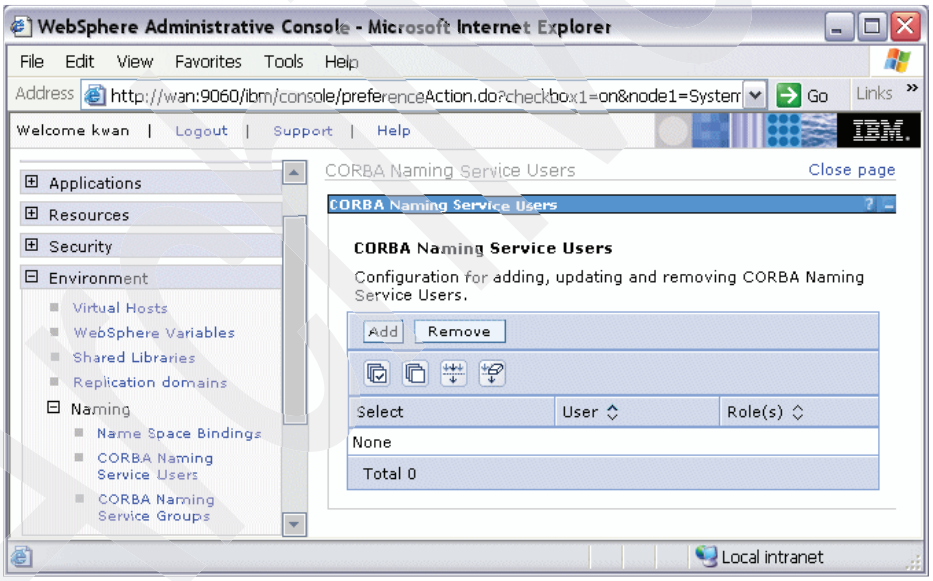


Figure 13-11 Add CORBA naming service users

2. Click **Add**.

   Enter a case-sensitive user ID and select an authorization level. The user must be a valid user in the active user registry. If you have not activated global security, the local operating system user registry will be used.

Note: Before these settings take effect, you will have to enable and configure WebSphere global security.

To specify multiple roles, hold the Ctrl key while you click the applicable roles. See Figure 13-12.

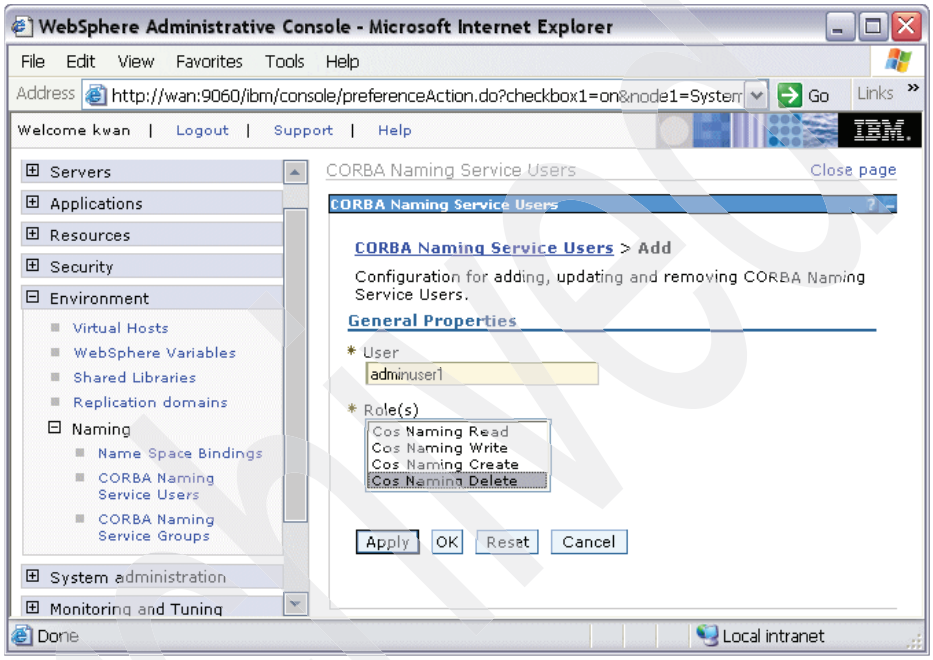


Figure 13-12 Assign an authorization level

3. Click **Apply**.
4. Click **OK** and save your changes.

Working with CORBA naming service groups

To work with groups, do the following:

1. Select **Environment** →**Naming** →**CORBA Naming Service Groups**. Notice that the default settings are defined. Figure 13-13 on page 818 shows the initial settings.

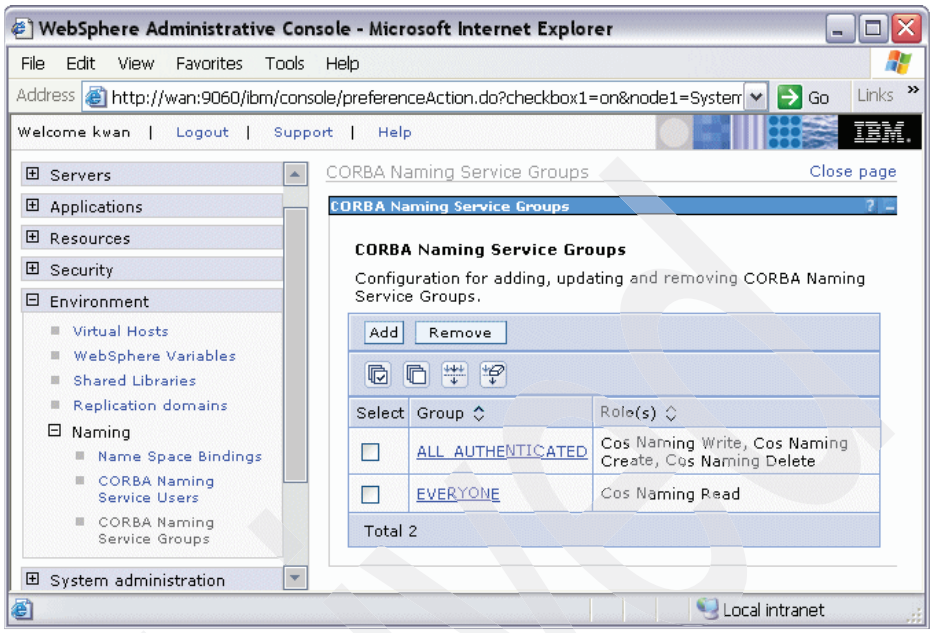


Figure 13-13 Default settings for CORBA naming service groups

Note: The two special groups are already defined and have roles assigned. To change the roles assigned to the two special groups, click the group name link).

2. To add a new group, click **Add**. See Figure 13-14 on page 819.

Archive

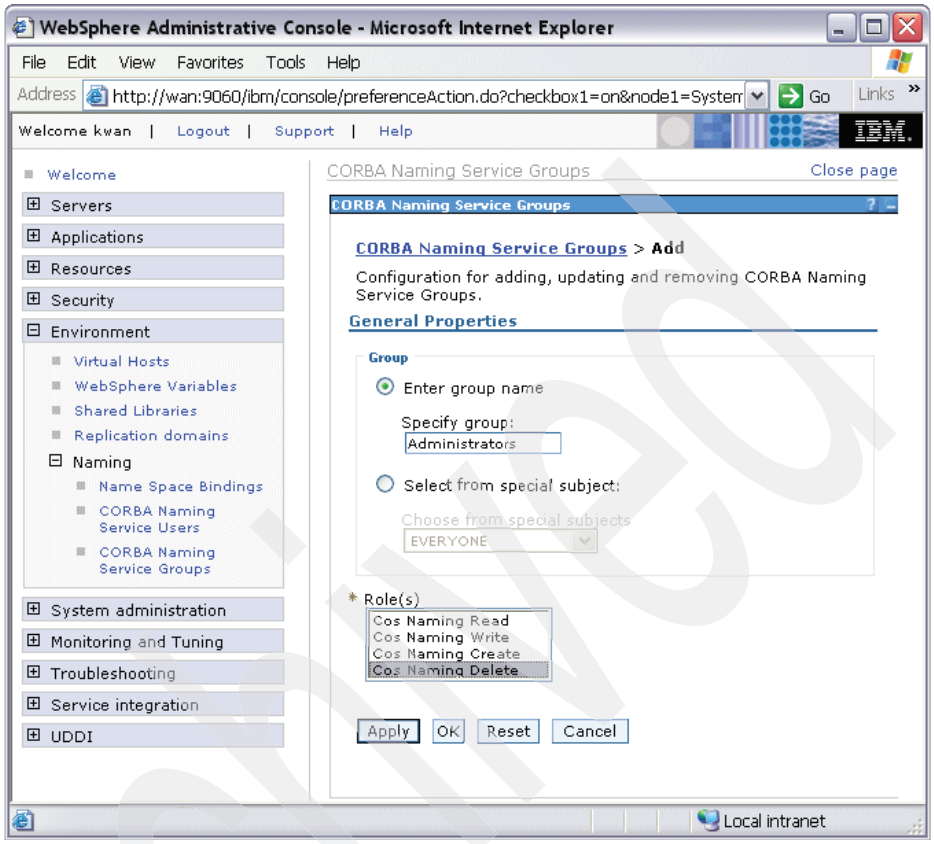


Figure 13-14 Assign an authorization level

3. Select the **Specify Group** button, enter a case-sensitive group name and select an authorization level, or role. The group must be a valid user in the active user registry. If you have not activated global security, the local operating system user registry will be used. Remember, before these settings take effect, you will have to enable and configure WebSphere global security.

   To specify multiple roles, hold the Ctrl key while you click the applicable roles.

4. Click **Apply**.
5. Click **OK** and save your changes.

Archived

Archived

14

Understanding class loaders

Understanding how the Java and WebSphere class loaders work is critical to J2EE packaging. Failure to set up the class loaders properly most likely results in a cascade of the infamous class loading exceptions like ClassNotFoundException when trying to start your application.

This chapter starts by giving a little background on Java class loaders. Then, we describe the different WebSphere class loaders and finally how you can customize the behavior of the WebSphere class loaders to suit your particular application's requirements.

The chapter concludes with an example designed to illustrates these concepts.

© Copyright IBM Corp. 2005. All rights reserved.

14.1 A brief introduction to Java class loaders

Class loaders enable the Java virtual machine (JVM) to load classes. Given the name of a class, the class loader locates the definition of this class. Each Java class must be loaded by a class loader.

When you start a JVM, you use three class loaders: the Bootstrap class loader, the Extensions class loader, and the System class loader.

- The *bootstrap* class loader is responsible for loading the core Java libraries, that is core.jar, server.jar etc. in the <JAVA_HOME>/lib directory. This class loader, which is part of the core JVM, is written in native code.

  Note: Beginning with JDK 1.4 the core, Java libraries in the IBM JDK are no longer packaged in rt.jar, but, instead, are split into multiple JAR files. Previously, they were packaged together and still are for the Sun JDKs.

- The *extensions* class loader is responsible for loading the code in the extensions directories (<JAVA_HOME>/lib/ext or any other directory specified by the `java.ext.dirs` system property). This class loader is implemented by the sun.misc.Launcher$ExtClassLoader class.
- The *system* class loader is responsible for loading the code that is found on java.class.path, which ultimately maps to the system CLASSPATH variable. This class loader is implemented by the sun.misc.Launcher$AppClassLoader class.

Delegation is a key concept to understand when dealing with class loaders. It states that a *custom* class loader, a class loader other than the bootstrap, extension or system class loaders, delegates class loading to its parent before trying to load the class itself. The parent class loader can be either another custom class loader or the bootstrap class loader.

The Extensions class loader is the parent for the System class loader. The Bootstrap class loader is the parent for the Extensions class loader. The class loaders hierarchy is shown in Figure 14-1 on page 823.

If the System class loader needs to load a class, it first delegates to the Extensions class loader, which, in turn, delegates to the Bootstrap class loader. If the parent class loader cannot load the class, the child class loader tries to find the class in its own repository. In this manner, a class loader is only responsible for loading classes that its ancestors cannot load.

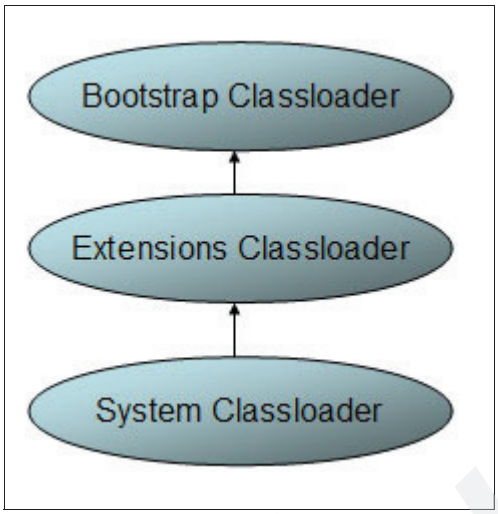


Figure 14-1 Java class loaders hierarchy

This behavior can lead to some interesting problems if a class is loaded from a class loader that is not on a leaf node in the class loader tree. Consider Example 14-1. A class called WhichClassLoader1 loads a class called WhichClassLoader2, in turn invoking a class called WhichClassLoader3.

Example 14-1 WhichClassLoader1 and WhichClassLoader2 source code

```
public class WhichClassLoader1 {

   public static void main(String[] args)
      throws javax.naming.NamingException
   {
      // Getting Classpathes Value
      StringBuffer bootstrapClassPath =
         new StringBuffer(System.getProperty("sun.boot.class.path"));
      StringBuffer extClassPath =
         new StringBuffer(System.getProperty("java.ext.dirs"));
      StringBuffer systemClassPath =
         new StringBuffer(System.getProperty("java.class.path"));
      // Printing them out
      System.out.println("Bootstrap classpath=\t" +
                                          bootstrapClassPath + "\t" );
      System.out.println("\nExtension classpath=\t" + extClassPath + "\t");
      System.out.println("\nSystem classpath=\t" + systemClassPath + "\t\n");

      //Loading Classes
      Object obj = new Object();
      WhichClassLoader1 wcl1 = new WhichClassLoader1();
      WhichClassLoader2 wcl2 = new WhichClassLoader2();
```

Archived

```
      //Who loaded what?
      System.out.println("->Object was loaded by " +
                                    obj.getClass().getClassLoader());
      System.out.println(
         "->WCL1 class was loaded by " + wcl1.getClass().getClassLoader());
      System.out.println(
         "->WCL2 class was loaded by " + wcl2.getClass().getClassLoader());
      wcl2.getTheClass();
   }
}
=============================================================================
public class WhichClassLoader2 {

   //This method is invoked from WhichClassLoader1.
   public void getTheClass() {
      WhichClassLoader3 wcl3 = new WhichClassLoader3();
      System.out.println("->WCL 3 was loaded
by:"+wcl3.getClass().getClassLoader());
   }
}
```

If all WhichClassLoaderX classes are put on the system class path, the three classes are loaded by the System class loader, and this sample runs just fine. Now suppose you package the WhichClassLoader2.class file in a JAR file that you store under *<JAVA_HOME>*/lib/ext directory. You see the output in Example 14-2.

Example 14-2 NoClassDefFoundError exception trace

```
Bootstrap classpath=
F:\WSADV5\eclipse\jre\lib\rt.jar;F:\WSADV5\eclipse\jre\lib\i18n.jar;F:\WSADV5\e
clipse\jre\classes
Extension classpath=F:\WSADV5\eclipse\jre\lib\ext
System classpath=E:\WSADworkspaces\WebbankV5\ClassloadersTest\bin

->Object was loaded by null
->WCL1 class was loaded by sun.misc.Launcher$AppClassLoader@5059e39d
->WCL2 class was loaded by sun.misc.Launcher$ExtClassLoader@505ca39d
java.lang.NoClassDefFoundError:
com/ibm/wss/lge/classloaders/test/WhichClassLoader3
at
com.ibm.wss.lge.classloaders.test.WhichClassLoader2.getTheClass(WhichClassLoade
r2.java:6)
at
com.ibm.wss.lge.classloaders.test.WhichClassLoader1.main(WhichClassLoader1.java
:27)
Exception in thread "main"
```

Archived

As you can see, the program fails with a NoClassDefFoundError exception, which might sound strange because WhichClassLoader3 is on the system class path. The problem is that it is on the wrong class path.

As you can see from the trace, the WhichClassLoader2 class was loaded by the Extensions class loader. In fact, the System class loader delegated the load of the WhichClassLoader2 class to the Extensions class loader, which delegated the load to the Bootstrap class loader. Because the Bootstrap class loader could not find the class, the class loading control was returned to the Extensions class loader. The Extensions class loader found the class and loaded it.

Now, the Extensions class loader needs to load the WhichClassLoader3 class. It delegates to the Bootstrap class path, which cannot find the class, then tries to load it itself and does not find it either. A NoClassDefFoundError exception is thrown. Once a class is loaded by a class loader, any new classes that it tries to load reuse the same class loader, or go up the hierarchy to find a class.

A class loader can only find classes by going up the hierarchy, never down.

Note: Remember that developers very often also load property files through the class loader mechanism using the following syntax:

```
Properties p = new Properties();
p.load(MyClass.class.getClassLoader().getResourceAsStream("myApp.properties"
));
```

This means, if the class MyClass is loaded by the Extension class loader and the myApp.properties file is only seen by the System class loader, the loading of the property file fails.

14.2 WebSphere class loaders overview

Note: Keep in mind when reading the following discussion that each JVM has its own setup of class loaders. In a WebSphere environment hosting multiple application servers (JVMs), such as a Network Deployment configuration, this means the class loaders for the JVMs are completely separated even if they are running on the same physical machine.

WebSphere provides several custom delegated class loaders as shown in Figure 14-2 on page 826.

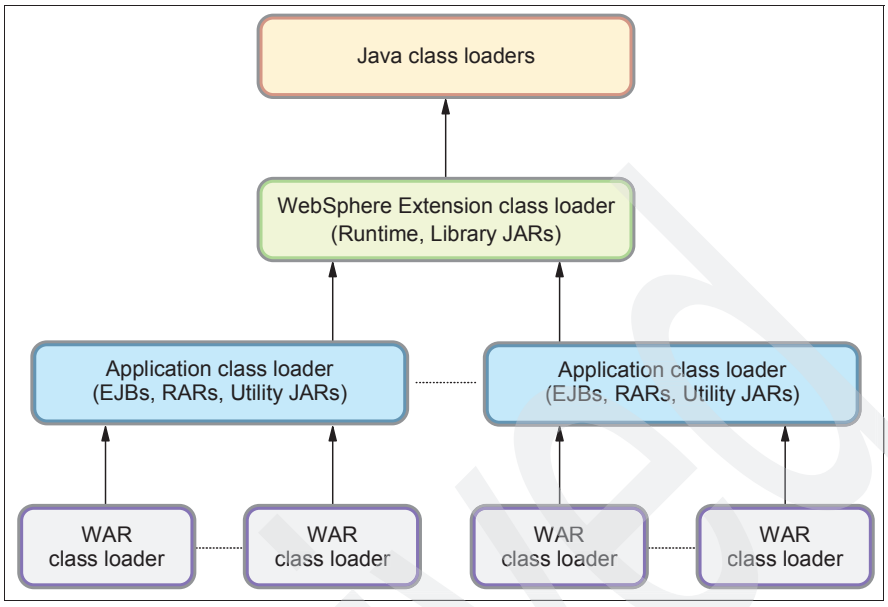


Figure 14-2 WebSphere class loaders hierarchy

The top box in red represents the Java (Bootstrap, Extension and System) class loaders. WebSphere loads just enough here to get itself bootstrapped and initialize the WebSphere extension class loader.

14.2.1 WebSphere extensions class loader

The WebSphere extensions class loader is where WebSphere itself is loaded. It uses the following directories to load the required WebSphere classes:

- <JAVA_HOME>\lib
- <WAS_HOME>\classes (Runtime Class Patches directory, or RCP)
- <WAS_HOME>\lib (Runtime class path directory, or RP)
- <WAS_HOME>\lib\ext (Runtime Extensions directory, or RE)
- <WAS_HOME>\installedChannels

The WebSphere runtime is loaded by the WebSphere extensions class loader based on the ws.ext.dirs system property, initially derived from the WS_EXT_DIRS environment variable set in the setupCmdLine script file. The default value of ws.ext.dirs is displayed in Example 14-3 on page 827.

Example 14-3 Default value of we.ext.dirs

```
SET
WAS_EXT_DIRS=%JAVA_HOME%\lib;%WAS_HOME%\classes;%WAS_HOME%\lib;%WAS_HOME%\instal
lledChannels;%WAS_HOME%\lib\ext;%WAS_HOME%\web\help;%ITP_LOC%\plugins\com.ibm.e
tools.ejbdeploy\runtime
```

The RCP directory is intended to be used for fixes and other APARs that are applied to the application server runtime. These patches override any copies of the same files lower in the RP and RE directories. The RP directory contains the core application server runtime files. The bootstrap class loader first finds classes in the RCP directory then in the RP directory. The RE directory is used for extensions to the core application server runtime.

Each directory listed in the ws.ext.dirs environment variable is added to the WebSphere extensions class loaders class path. In addition, every JAR file and ZIP file in the directory is added to the class path.

You can extend the list of directories and files loaded by the WebSphere extensions class loaders by setting a ws.ext.dirs custom property to the Java virtual machine settings of an application server.

14.2.2 Application and Web module class loaders

J2EE applications consist of five primary elements: Web modules, EJB modules, application client modules, resource adapters (RAR files), and Utility JARs. Utility JARs contain code used by both EJBs and servlets. Utility frameworks such as log4j are good examples of a utility JAR.

EJB modules, utility JARs, resource adapters files, and shared libraries associated with an application are always grouped together into the same class loader. This class loader is called the Application class loader. Depending on the application class loader policy, this application class loader can be shared by multiple applications (EARs), or be unique for each application, which is the default.

By default, Web modules receive their own class loader, a WAR class loader, to load the contents of the WEB-INF/classes and WEB-INF/lib directories. Modify the default behavior by changing the application's WAR class loader policy. The default is Module. If the WAR class loader policy is set to Application, the Web module contents are loaded by the *Application class loader* in addition to the EJBs, RARs, utility JARs, and shared libraries. The Application class loader is the parent of the WAR class loader.

The application and the Web module class loaders are reloadable class loaders. They monitor changes in the application code to automatically reload modified classes. You can modify this behavior at deployment time.

14.2.3 Handling JNI code

Due to a JVM limitation, code that access to native code through a Java Native Interface (JNI) must be placed on a static class path, not on a reloadable class path. This includes shared libraries for which you can define a native class path, or the application server class path. If you you have a class loading native code through JNI, this class must not be placed on the WAR or Application class loaders, but rather on the WebSphere extensions class loader.

You can break out just the lines of code that load the native library into a class of its own and place this class on a static class loader. This way you can have all the other code on a reloadable class loader.

14.3 Configuring WebSphere for class loaders

In the previous topic, you learned about WebSphere class loaders and how they work together to load classes. There are settings in WebSphere Application Server that allow you to influence WebSphere class loader behavior. This section discusses these options.

14.3.1 Class loader policies

For each application server in the system, the application class loader policy can be set to Single or Multiple.

When the application class loader policy is set to Single, a single application class loader is used to load all EJBs, utility JARs, and shared libraries within the application server (JVM). If the WAR class loader policy then has been set to Application, the Web module contents for this particular application are also loaded by this single class loader.

When the application class loading policy is set to Multiple, the default, each application will receive its own class loader for loading EJBs, utility JARs, and shared libraries. Depending on whether the WAR class loader loading policy is set to Module or Application, the Web module might or might not receive its own class loader.

Here is an example to illustrate. You have two applications, Application1 and Application2, running in the same application server. Each application has one EJB module, one utility JAR and two Web modules. If the application server has

its class loader policy set to Multiple, the default, and the class loader policy for all the Web modules are set to Module, also the default, the result is as shown in Figure 14-3.

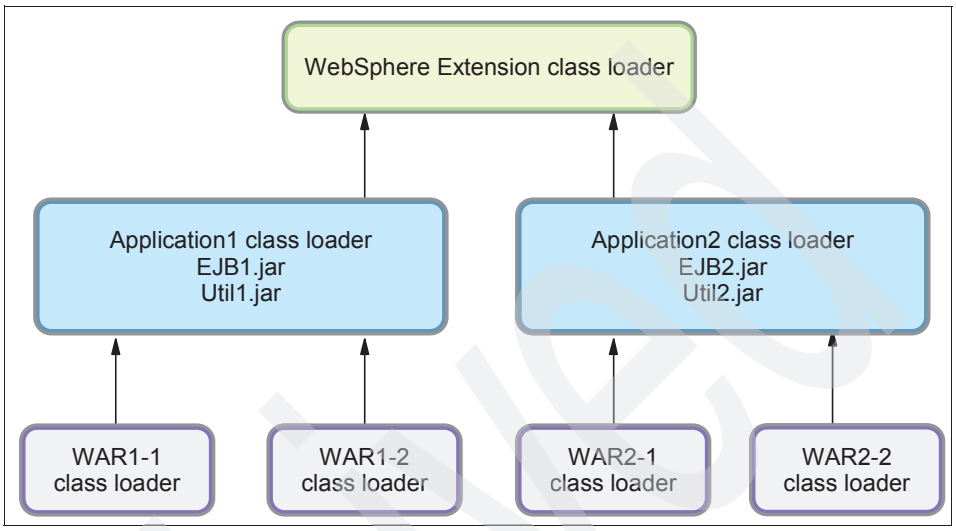


Figure 14-3 Class loader policies: Example 1

Each application is completely separated from the other and each Web module is also completely separated from the other one in the same application. WebSphere's default class loader policies results in total isolation between the applications and the modules.

If we now change the class loader policy for the WAR2-2 module from Module to Application, the result is as shown in Figure 14-4 on page 830.

Archives

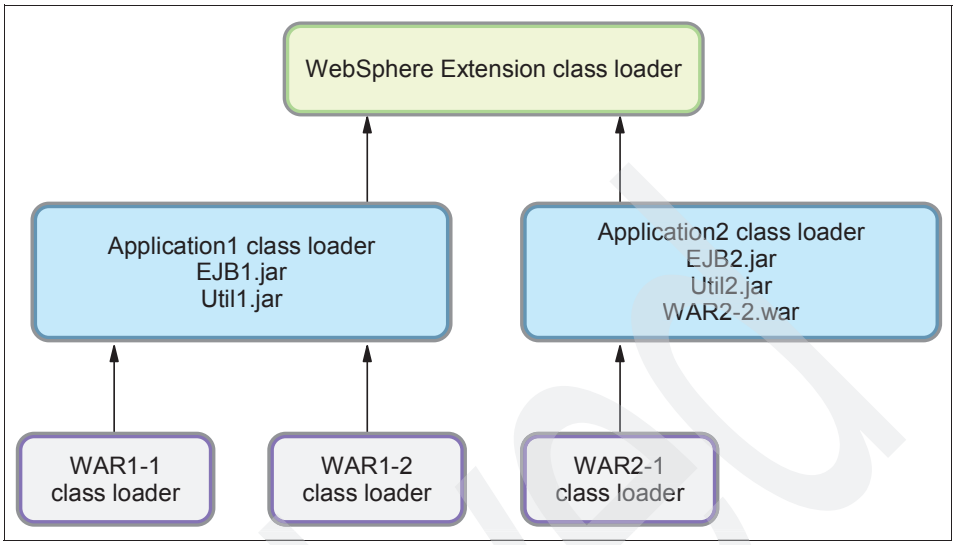


Figure 14-4 Class loader policies: Example 2

Web module WAR2-2 is loaded by Application2's class loader and classes for example in Util2.jar are able to see classes in WAR2-2's /WEB-INF/classes and /WEB-INF/lib directories.

As a last example, if we change the class loader policy for the application server from Multiple to Single and also change the class loader policy for WAR2-1 from Module to Application, the result is as shown in Figure 14-5 on page 831.

Archive

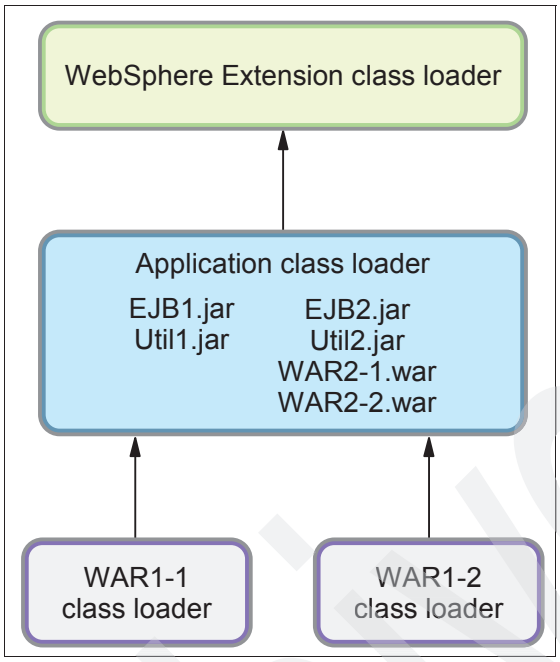


Figure 14-5 Class loader policies: Example 3

There is now only a single application class loader loading classes for both Application1 and Application2. Classes in Util1.jar can see classes in EJB2.jar, Util2.jar, WAR2-1.war and WAR2-2.war. The classes loaded by the application class loader can still not, however, see the classes in the WAR1-1 and WAR1-2 modules because a class loader can only find classes by going up the hierarchy, never down.

14.3.2 Class loader/delegation mode

WebSphere's application class loader and WAR class loader both have a setting called the class loader mode. This setting determines whether they should follow the normal Java class loader delegation mechanism as described in 14.1, "A brief introduction to Java class loaders" on page 822 or override it. There are two possible values for the class loader mode: PARENT_FIRST and PARENT_LAST.

The default value for class loader mode is PARENT_FIRST. This mode causes the class loader to first delegate the loading of classes to its parent class loader before attempting to load the class from its local class path. This is the default policy for standard Java class loaders.

Archived

If the class loading policy is set to PARENT_LAST, the class loader attempts to load classes from its local class path before delegating the class loading to its parent. This policy allows an application class loader to override and provide its own version of a class that exists in the parent class loader.

Delegation mode can be set for the following class loaders:

- Application class loader
- WAR class loader
- Shared library class loader

Assume you have an application, similar to Application1 in the previous examples, and it uses the popular log4j package to perform logging from both the EJB module and the two Web modules. Also assume that each module has its own, unique, log4j.properties file packaged into the module. It is tempting to configure log4j as a utility JAR so you only have a single copy of it in your EAR file.

However, if doing so you would perhaps be surprised to see that all modules, also the Web modules, load the log4j.properties file from the EJB module. The reason is that when a Web module initializes the log4j package, the log4j classes are loaded by the application class loader. Log4j is configured as a utility JAR. Log4j then looks for a log4j.properties file on its class path and finds it in the EJB module.

Even if you do not use log4j for logging from the EJB module and the EJB module does not, therefore, contain a log4j.properties file, log4j does not find the log4j.properties file in any of the Web modules anyway. The reason is that a class loader can only find classes by going up the hierarchy, never down.

To solve this problem you can either:

- Create a separate file, for example Resource.jar, configure it as a utility JAR, move all log4j.properties files from the modules into this file, make their names unique (like war1-1_log4j.properties, war1-2_log4j.properties and ejb1_log4j.properties). When initializing log4j from each module, tell it to load the proper configuration file for the module instead of the default (log4j.properties).
- Keep the log4j.properties for the Web modules in their original place (/WEB-INF/classes), add log4j.jar to both Web modules (/WEB-INF/lib) and set the class loader mode for the Web modules to PARENT_LAST. When initializing log4j from a Web module, it loads the log4j.jar from the module itself and log4j would find the log4j.properties on its local classpath, the Web module itself. When the EJB module initializes log4j, it loads from the application class loader and it finds the log4j.properties file on the same class path, the one in the EJB1.jar file.

- Merge, if possible, all log4j.properties files into one and place it on the application class loader, in a Resource.jar file, for example).

Singletons: The Singleton pattern is used to ensure that a class is instantiated only once. However, *once* only means *once for each class loader.* If you have a Singleton being instantiated in two separated Web modules, two separate instances of this class will be created, one for each WAR class loader. So in a multi-class loader environment, special care must be taken when implementing Singletons.

14.3.3 Class preloading

The first time that a WebSphere Application Server process starts, the name of each run-time class that is loaded and the name of the JAR file that contains the class are written to a preload file. The names of non-runtime classes such as custom services, resource classes such as db2jcc.jar, classes on the JVM class path, and application classes, your own J2EE applications, are not written to the preload file. Subsequent startups of the process use the preload file to start the process more quickly.

Preload files have the .preload extension. WebSphere Application Server processes that have preload files include those listed in Table 14-1.

Table 14-1 WebSphere class preloading file names

Process	Preload file name
Application server	cell_name.node_name.server_name.preload
startServer	WsServerLauncher.preload
launchClient	launchClient.preload

Running the startServer server1 command causes the startServer command to use a WsServerLauncher.preload file and the server to use a cell_name.node_name.server1.preload file. Later, running a command such as startServer server1 -script, where the -script option creates a new script, uses the cell_name.node_name.server1.preload file only.

Preload files, by default, are created in the logs/preload directory of the profile directory.

New classes required during the startup of a process are added to the preload file. Any classes removed from a process are ignored during subsequent startups. Although it is not necessary, an administrator can delete the preload file and force a refresh that removes the ignored classes from the file.

Class preloading is enabled by default. If you want to disable it for a particular application server, set the JVM property `ibm.websphere.preload.classes` to `false` for that application server. To do this, do the following:

1. In the administrative console, click **Servers > Application Servers >** server_name **> Java and Process Management > Process Definition > Java Virtual Machine**.
2. On the Java Virtual Machine page, specify `-Dibm.websphere.preload.classes=false` for Generic JVM arguments.
3. Click **OK**.
4. Save your administrative configuration.
5. Stop the application server and then restart it.

To enable class preloading again, either remove the ibm.websphere.preload.classes setting or set it to true and then stop and restart the application server.

14.3.4 Shared libraries

Shared libraries are files used by multiple applications. Examples of shared libraries are commonly used frameworks like Apache Struts or log4j. You use shared libraries typically to point to a set of JARs and associate those JARs to an application or application server. Shared libraries are especially useful when you have different versions of the same framework you want to associate to different applications.

Shared libraries are defined using the administration tools. They consist of a symbolic name, a Java class path, and a native path for loading JNI libraries. They can be defined at the cell, node, or server level. However, defining a library at one of the three levels does not cause the library to be loaded. You must associate the library to an application and an application server, or just an application server, for the classes represented by the shared library to be loaded.

You can associate the shared file to an application in one of two ways:

- You can use the administration tools. The library is added under the Libraries section of the additional properties for the enterprise application.
- You can use the manifest file of the application and the shared library. The shared library contains a manifest file that identifies it as an extension. The dependency to the library is declared in the application's manifest file by listing the library extension name in an extension list.

  For more information about this method, search *installed optional packages* in the Information Center.

Archived

Shared files are associated with an application server using the administrative tools. The settings are found in the Server Infrastructure section. Expand the Java and Process Management. Select **Class loader**. At this point, you have two options:

- If you associate the shared library to an application, the JARs listed on the shared library path are loaded by the application class loader together with EJB JARs, RARs and utility JARs.
- If you associate the shared library at the application server level, the JARs listed on the shared library path are loaded by a specific class loader, which you define.

See “Step 4: Sharing utility JARs among multiple applications” on page 840 for more details.

14.4 Learning class loaders by example

We have now described all the different options for influencing class loader behavior. In this section we take an example and use all the different options we have discussed to this point so that you can better evaluate the best solution for your applications.

We have created a very simple application, with one JSP and one EJB. Both call a class, VersionChecker, shown in Example 14-4. This class can print which class loader was used to load the class. The VersionChecker class also has an internal value that can be printed to check which version of the class we are using. This will be used later to demonstrate the use of multiple versions of the same utility JAR.

Example 14-4 VersionChecker class source code

```
public class VersionChecker {

   static final public String classVersion = "1.0";

   public String printClassLoaderInfo() {
      return ("Class Version Checker was loaded by: " +
                  this.getClass().getClassLoader());
   }

   public String printVersionInfo () {
      return ("Class version is: " + classVersion);
   }
}
```

Once installed, the application can be invoked through `http://localhost:9080/testclassloaders`. This invokes a JSP (callVersionChecker.jsp) which returns the sample information in Example 14-5.

Example 14-5 Invoking callVersionChecker.jsp

```
Class Version Checker was loaded by:
com.ibm.ws.classloader.CompoundClassLoader@1284e9f8
Local ClassPath:
C:\WebSphere\AppServer\profiles\AppSrv01\installedApps\kcfn7r6Cell\ClassloaderT
est.ear\ClassloaderTestEJB.jar;C:\WebSphere\AppServer\profiles\AppSrv01\install
edApps\kcfn7r6Cell\ClassloaderTest.ear\myjars\ClassloadersTest.jar; Delegation
Mode: PARENT_FIRST
Class version is: 1.0
```

The ClassloadersTest.jar file contains the VersionChecker class file. For all the following tests, we have, unless otherwise noted, left the class loader policies and loading modes to their defaults. In other words, we have one class loader for the application and one for the WAR file. Both have their delegation modes set to `PARENT_FIRST`. We assume the application has been deployed to an application server called AppSrv01. See Example 14-6.

Example 14-6 CallVersionChecker.jsp source

```
...
<TITLE>WAS V6 Classloaders Info</TITLE>
</HEAD>
<BODY>
<h2> Version Checker Information </h2>
<h3> Direct call from Servlet to Version Checker Class </h3>
<%
 com.ibm.wss.lge.classloaders.VersionChecker versionChecker
            = new com.ibm.wss.lge.classloaders.VersionChecker();
 out.println("<br/>" + versionChecker.printClassLoaderInfo());
 out.println("<br/>" + versionChecker.printVersionInfo());
%>
<h3> Call the VersionChecker via Access bean-> enterprise session bean </h3>
<%
 com.ibm.wss.lge.ejbs.ClassLoaderTesterAccessBean versionCheckerAB
            = new com.ibm.wss.lge.ejbs.ClassLoaderTesterAccessBean();
 out.println("<br />" + versionCheckerAB.callVersionChecker());
%>
</BODY>
...
```

During the tests, we worked directly on the file system by modifying the contents of the *<profile_home>*\installedApps\kcfn7r6Cell\ClassloaderTest.ear directory. Obviously, this is not a wise thing to do in production!

14.4.1 Step 1: Simple WAR packaging

Start with the following assumption: our utility class is only used by a servlet. In the JSP, we have commented out the lines that invoke the EJB part of the application. We have placed the ClassloadersTest.jar file under the WEB-INF/lib directory of the WAR file.

> **Tip:** You put under **WEB-INF/lib** JAR files used by a **single** Web module, or a JAR file that *only* this Web module should see.

When we run the application in such a configuration, we obtain the results shown in Example 14-7.

Example 14-7 Class loader: Example 1

```
Version Checker Information
Direct call from Servlet to Version Checker Class

Class Version Checker was loaded by:
com.ibm.ws.classloader.CompoundClassLoader@31d13f47

Local ClassPath:
C:\WebSphere\AppServer\profiles\AppSrv01\installedApps\Cell01\ClassloadersTest2
.ear\ClassloaderTestWeb2.war\WEB-INF\classes;C:\WebSphere\AppServer\profiles\Ap
pSrv01\installedApps\Cell01\ClassloadersTest2.ear\ClassloaderTestWeb2.war\WEB-I
NF\lib\ClassloadersTest.jar;C:\WebSphere\AppServer\profiles\AppSrv01\installedA
pps\Cell01\ClassloadersTest2.ear\ClassloaderTestWeb2.war;

Delegation Mode: PARENT_FIRST
Class version is: 1.0
```

There are a few things we can learn from this trace:

1. The type of the WAR class loader is `com.ibm.ws.classloader.CompoundClassLoader`.
2. It searches classes in the following order:

   ```
   ClassloaderTestWeb.war\WEB-INF\classes
   ClassloaderTestWeb.war\WEB-INF\lib\ClassloadersTest.jar
   ClassloaderTestWeb.war
   ```

The WEB-INF/classes folder is usually used for servlets and property files, while the WEB-INF/lib is used for JARs (not ZIPs). Note that the class loader class path is built at application startup, which is why you see only the ClassloadersTest JAR listed there. If multiple JARs had been placed under this folder, you would see all of them listed on the local class path. The root of the WAR file is the next place where you can put code or properties classes.

Archived

14.4.2 Step 2: Sharing the utility JAR among multiple modules

Next, we decided to run the EJB part of the application, which also depends on our ClassloadersTest.jar JAR file. Because the JAR file is to be used by multiple modules in the same application, the best solution is to place it relative to the root of the EAR file and to reference it from a Class-Path entry in the JAR manifest file.

In Example 14-7 on page 837, the VersionChecker stored in in WEB-INF/LIB/ClassloadersTest.jar returned a class version of 1.0 in the output. For illustration purposes, we have created a new JAR file called ClassloadersTestV2.jar. The VersionChecker stored in this JAR file returns version 2.0.

The Manifest Class-Path entry

In this situation, we place the JAR file in a /myjars/ directory at the root of the EAR file. From a J2EE perspective, this directory is unknown, and the directory contents will not be added to the class path automatically. The solution is to create a `Class-Path` entry in the META-INF/MANIFEST.MF file for each module which uses classes from the classloadersTest.jar file. Paths are relative to the root of the EAR file.

Update the WAR file as in Example 14-8:

Example 14-8 Updating the WAR file

```
#This is the Manifest file for ClassloaderTestWeb file
Manifest-Version: 1.0
Class-Path: ClassloaderTestEJB.jar myjars/ClassloadersTestV2.jar
```

In the EJB file, update the MANIFEST.MF as in Example 14-9:

Example 14-9 Updating MANIFEST.MF

```
#This is the Manifest file for ClassloaderTestEJB
Manifest-Version: 1.0
Class-Path:  myjars/ClassloadersTestV2.jar
```

The test results are then as shown in Example 14-10.

Example 14-10 Class loader: Example 2

```
Version Checker Information
Direct call from Servlet to Version Checker Class

Class Version Checker was loaded by:
com.ibm.ws.classloader.CompoundClassLoader@3096c631
```

```
Local ClassPath:
C:\WebSphere\AppServer\profiles\AppSrv01\installedApps\Cell01\ClassloadersTest.
ear\ClassloaderTestEJB.jar;C:\WebSphere\AppServer\profiles\AppSrv01\installedAp
ps\Cell01\ClassloadersTest.ear\myjars\ClassloadersTestV2.jar;

Delegation Mode: PARENT_FIRST
Class version is: 2.0

Call the VersionChecker via Access bean to enterprise session bean
VersionChecker Classloader Info

Class Version Checker was loaded by:
com.ibm.ws.classloader.CompoundClassLoader@3096c631 Local ClassPath:
C:\WebSphere\AppServer\profiles\AppSrv01\installedApps\Cell01\ClassloadersTest.
ear\ClassloaderTestEJB.jar;C:\WebSphere\AppServer\profiles\AppSrv01\installedAp
ps\Cell01\ClassloadersTest.ear\myjars\ClassloadersTestV2.jar;

Delegation Mode: PARENT_FIRST
Class version is: 2.0
```

Although the class loader type is the same (CompoundClassLoader), this is the application class loader, not the WAR class loader. You can see this from the class path, which is very different. Remember, all utility JARs are loaded by the application class loader. As soon as you reference a JAR with a Class-Path directive, it is considered a utility JAR. The *same* class loader is used by the servlet and the EJBs to load the VersionChecker class.

14.4.3 Step 3: Changing the WAR class loader delegation mode

What happened to the ClassloadersTest.jar under WEB-INF/lib? It is still there, but it is ignored because the delegation mode of the WAR is set to PARENT_FIRST.

Set the delegation mode to PARENT_LAST, using the following steps:

1. Select the **Enterprise Applications** entry in the navigation area.
2. Select the **ClassloaderTest** application.
3. Select **Web Modules** under the Related Items section.
4. Select the **ClassloaderTestWeb.war**.
5. Change the Class loader Mode to PARENT_LAST.
6. Click **OK**.
7. Save the configuration.
8. Restart the application.

The VersionChecker in WEB-INF/lib returns a class version of 1.0. You can see in Example 14-11 on page 840 that this is the version used by the WAR file.

Archived

Example 14-11 Class loader: Example 3

```
Version Checker Information
Direct call from Servlet to Version Checker Class

Class Version Checker was loaded by:
com.ibm.ws.classloader.CompoundClassLoader@30a70501

Local ClassPath:
C:\WebSphere\AppServer\profiles\AppSrv01\installedApps\Cell01\ClassloadersTest.
ear\ClassloaderTestWeb.war\WEB-INF\classes;C:\WebSphere\AppServer\profiles\AppS
rv01\installedApps\Cell01\ClassloadersTest.ear\ClassloaderTestWeb.war\WEB-INF\l
ib\ClassloadersTest.jar;C:\WebSphere\AppServer\profiles\AppSrv01\installedApps\
Cell01\ClassloadersTest.ear\ClassloaderTestWeb.war;

Delegation Mode: PARENT_LAST
Class version is: 1.0

Call the VersionChecker via Access bean to enterprise session bean
VersionChecker Classloader Info

Class Version Checker was loaded by:
com.ibm.ws.classloader.CompoundClassLoader@30c94501

Local ClassPath:
C:\WebSphere\AppServer\profiles\AppSrv01\installedApps\Cell01\ClassloadersTest.
ear\ClassloaderTestEJB.jar;C:\WebSphere\AppServer\profiles\AppSrv01\installedAp
ps\Cell01\ClassloadersTest.ear\myjars\ClassloadersTestV2.jar;

Delegation Mode: PARENT_FIRST
Class version is: 2.0
```

Tip: Use this to specify that a Web module should use a specific version of a library such as Struts, or to override classes coming with the WebSphere runtime. Put the common version at the top of the hierarchy, and the specialized version in WEB-INF/lib.

14.4.4 Step 4: Sharing utility JARs among multiple applications

In this situation, the ClassloaderTest.jar file is used by a single application. What if you wanted to share it among multiple applications? Of course, you could package it within each EAR file. But changes to this utility JAR file require redeploying all applications again. To avoid this, you can externalize global utility JARs.

You have two solutions to this problem:

- Use the application extensions class loader, inherited from Version 4.0.
- Use shared libraries, the recommended solution.

Using the application extensions class loader

The role of this class loader is to load any JAR file placed in the *<was_home>*/lib/app directory. This class loader has a delegation mode set to PARENT_LAST.

Note: All applications running within the same WebSphere node running with PARENT_FIRST delegation mode see what is contained in this directory, regardless of the application server in which they run. This might be what you want to achieve. You might have a framework that needs to be shared by all your applications. But if some applications want to use the level 1.0 of the framework while others want to use Version 2.0, using the lib/app directory is not the right solution.

By default, the *<was_home>*/lib/app directory does not exist. However, at application startup, the class loader runtime checks for its existence and starts the application extensions class loader if it does exist.

For testing purposes only, if you create this *<was_home>*/lib/app directory, drop in the ClassloadersTest.jar, and restart the application server, the results are shown in Example 14-12.

Example 14-12 Class loader: Example 4

```
Version Checker Information
Direct call from Servlet to Version Checker Class

Class Version Checker was loaded by:
com.ibm.ws.classloader.CompoundClassLoader@3128395a

Local ClassPath:
C:\WebSphere\AppServer\profiles\AppSrv01\installedApps\Cell01\ClassloadersTest.
ear\ClassloaderTestWeb.war\WEB-INF\classes;C:\WebSphere\AppServer\profiles\AppS
rv01\installedApps\Cell01\ClassloadersTest.ear\ClassloaderTestWeb.war\WEB-INF\l
ib\ClassloadersTest.jar;C:\WebSphere\AppServer\profiles\AppSrv01\installedApps\
Cell01\ClassloadersTest.ear\ClassloaderTestWeb.war;

Delegation Mode: PARENT_LAST
Class version is: 1.0

Call the VersionChecker via Access bean to enterprise session bean
VersionChecker Classloader Info
```

```
Class Version Checker was loaded by:
com.ibm.ws.classloader.ExtJarClassLoader@1483061593 Local ClassPath:
C:\WebSphere\AppServer\lib\app\ClassloadersTest.jar;

Delegation Mode: PARENT_LAST
Class version is: 1.0
```

Note that the servlet still uses the code from WEB-INF/lib because its delegation mode is set to PARENT_LAST. The EJB, however, uses the code from the lib/app directory.

Using shared libraries

A much cleaner and manageable way to work is to use the shared libraries support. Shared libraries can be defined at the cell, node, and application server levels. Once you have defined a shared library, you must associate it to an application or to a server. JARs and folders listed on a shared library are always loaded by the application class loader.

You can define as many shared libraries as you want. You can also associate multiple shared libraries with an application or application server.

Using shared libraries at the application level

To define a shared library named VersionCheckerV2_SharedLib and associate it to our ClassloaderTest application, do the following:

1. In the administrative console, select **Environment** →**Shared Libraries**.
2. Select the level at which you want this shared library to be defined, such as Cell, and click **New**.
3. Specify the following properties as in Figure 14-6 on page 843.

Archived

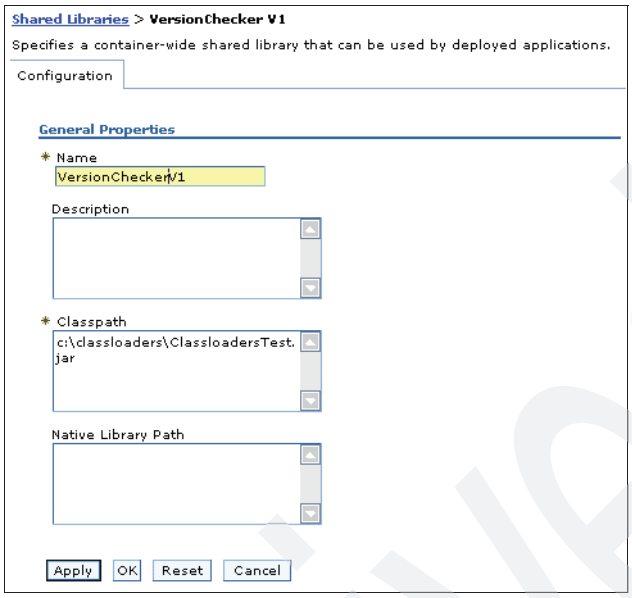


Figure 14-6 Shared library configuration

- **Name:** Enter `VersionCheckerV1`.
- **Class path:** Enter the list of entries on the class path. Press Enter between each entry. We highly recommend that if you need to provide an absolute path that you use WebSphere variables, such as %FRAMEWORK_JARS%/ClassloadersTest.jar. Make sure that you declare this variable at the same scope as the shared library for cell, node, or server).
- **Native library path:** Enter a list of DLLs and .so files for use by the JNI code.

4. Click **OK**.
5. Select **Applications** →**Enterprise Applications**.
6. Select the **ClassloadersTest** application.
7. In Additional Properties, select **Libraries**.
8. Click **Add**. You should see the library you just created. If not, you probably have a scoping problem.
9. Select the shared library, and click **OK**.
10. Save the configuration.

Archived

If we now remove the ClassloadersTest.jar file from the lib/app folder and restart the application server, we see the following test results in Example 14-13:

Example 14-13 Class loader: Example 5

```
Call the VersionChecker via Access bean to enterprise session bean
VersionChecker Classloader Info

Class Version Checker was loaded by:
com.ibm.ws.classloader.CompoundClassLoader@30ec422c Local ClassPath:
C:\WebSphere\AppServer\profiles\AppSrv01\installedApps\Cell01\ClassloadersTest.
ear\ClassloaderTestEJB.jar;C:\WebSphere\AppServer\profiles\AppSrv01\installedAp
ps\Cell01\ClassloadersTest.ear\myjars\ClassloadersTestV2.jar;C:\classloaders\Cl
assloadersTest.jar;

Delegation Mode: PARENT_FIRST
Class version is: 2.0
```

Although we have put the V1.0 of the ClassloadersTest.jar on the shared library, we see V2.0 being loaded. This is because we are loading the ClassloadersTestV2.jar, which is inside the EAR file. We have not removed it! The list of JAR files you declare in a shared library is appended to the application class loader class path. Keep in mind that order is very important.

Therefore, if you want the shared library code to be loaded, remove the `Class-path` entry from the EJB module MANIFEST.MF file, restart the application, and try again.

Using shared libraries at the application server level

A shared library can also be associated to an application server. All applications deployed in this server see the code listed on that shared library. To associate a shared library to an application server, you must first create a class loader as follows:

1. Select an application server.
2. In the Server Infrastructure section, expand the **Java and Process Management**. Select **Class loader**.
3. Choose **New**, and select a class loading policy for this class loader, **PARENT_FIRST** or **PARENT_LAST**. If you set the application policy to **PARENT_LAST**, you will be able to use your own classes in place of some WebSphere runtime classes , for example typically, XML parsers or transformers.
4. Click **Apply**.
5. Select the **Libraries** entry.

6. Click **Add**, and select the library you want to associate to this application server. Repeat this operation to associate multiple libraries to this class loader.
7. Click **OK**.
8. Save the configuration.

If we attach the VersionCheckerV1 shared library to the application server, we obtain the results in Example 14-14 with our application:

Example 14-14 Class loader: Example 6

```
Version Checker Information
Direct call from Servlet to Version Checker Class

Class Version Checker was loaded by:
com.ibm.ws.classloader.CompoundClassLoader@3118cdc9

Local ClassPath:
C:\WebSphere\AppServer\profiles\AppSrv01\installedApps\Cell01\ClassloadersTest.
ear\ClassloaderTestWeb.war\WEB-INF\classes;C:\WebSphere\AppServer\profiles\AppS
rv01\installedApps\Cell01\ClassloadersTest.ear\ClassloaderTestWeb.war\WEB-INF\l
ib\ClassloadersTest.jar;C:\WebSphere\AppServer\profiles\AppSrv01\installedApps\
Cell01\ClassloadersTest.ear\ClassloaderTestWeb.war;

Delegation Mode: PARENT_LAST
Class version is: 1.0

Call the VersionChecker via Access bean to enterprise session bean
VersionChecker Classloader Info

Class Version Checker was loaded by:
com.ibm.ws.classloader.ExtJarClassLoader@1204276682

Local ClassPath: C:\classloaders\ClassloadersTest.jar;

Delegation Mode: PARENT_FIRSTClass version is: 1.0
```

Note that the WAR class loader continues to load its own version due to the delegation mode, PARENT_LAST, while the EJB module loads the code from the highest class loader in the hierarchy, the one we just defined at the application server level.

Archived

Archived

15

Packaging applications

In this chapter, we show you how to perform some common tasks involved in packaging a J2EE application. For this purpose we will use the WebSphere Bank sample application that ships with WebSphere Application Server V6. We will show you how to:

- Set the JNDI bindings for its EJBs
- Change the sample to use a DB2 database
- Generate deployment code for the application

We will also discuss some advanced configuration options such as EJB caching and access intents.

Finally, we explain the concept of the Enhanced EAR file, introduced in WebSphere Application Server V6, and show how to add more deployment information to the WebSphere Bank EAR file.

To package the application, we will use the Application Server Toolkit. The toolkit is based on the Eclipse V3 platform and contains a subset of the plug-ins that make up Rational Application Developer V6. It allows you to create and modify J2EE applications and modules, edit deployment descriptors, and map databases. The toolkit also allows you to perform basic testing, debugging and profiling of WebSphere applications.

© Copyright IBM Corp. 2005. All rights reserved.

15.1 WebSphere Bank sample application

WebSphere Bank is an Internet bank that provides online checking and savings banking. Using WebSphere Bank, customers can open accounts, get account balances, and transfer funds between accounts. It has both a Web front-end and a simple text-mode interface accessible by running its J2EE application clients.

The application uses many of the J2EE and WebSphere Application Server functions such as EJBs (session, entity and message-driven beans), servlets, JSPs, Web services, JMS, and so forth.

We will not go into details on how WebSphere Bank works. For more detailed information, refer to the WebSphere Information Center. If you have installed the sample applications on your system, you can also find information at:

`http://localhost:9080/WSsamples`

When WebSphere Bank is installed and configured properly, it can be invoked at:

`http://localhost:9080/WebSphereBank`

The WebSphereBank EAR file consists of the modules show in Figure 15-1.

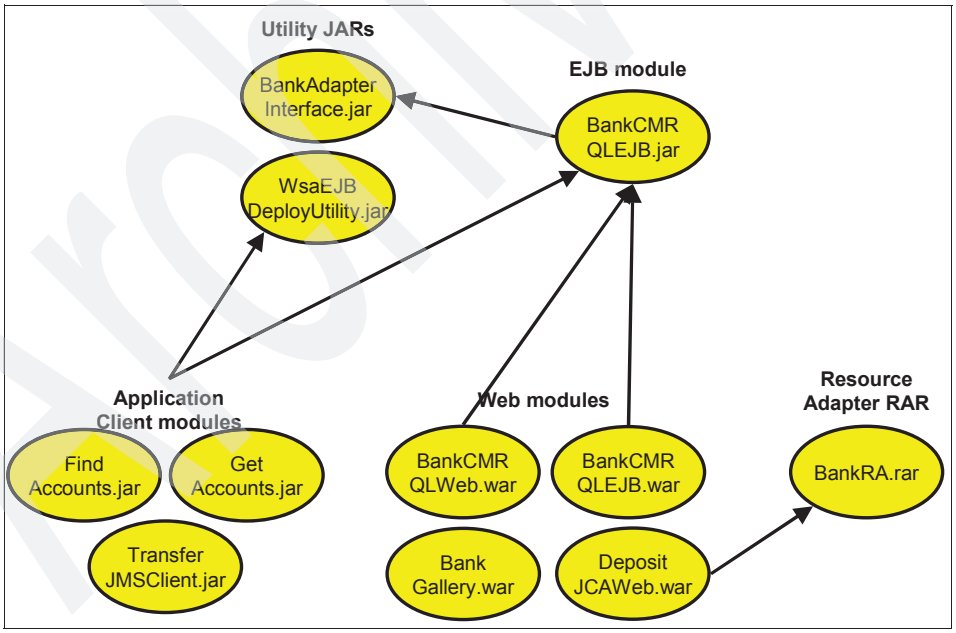


Figure 15-1 WebSphereBank modules

Archived

As shown, the WebSphere Bank EAR file has two utility JARs, one EJB module, four Web modules and one resource adapter module. It also has three application client modules which are used to invoke services on the system from a standalone JVM.

15.1.1 WebSphere Bank resources used

To run successfully, WebSphere Bank requires the following resources to be configured:

- JAAS authentication alias
  - Scope: cell level
  - Name: `<cellname>/samples`
  - Userid: `samples`
  - Password: `s1amples`
- JDBC provider
  - Scope: server level
  - Name: `Samples Cloudscape JDBC Provider (XA)`
  - Driver: Built-in Cloudscape JDBC Provider (XA)
- Data source
  - Scope: server level
  - JDBC provider: `Samples Cloudscape JDBC Provider (XA)`
  - Name: `BANKDS`
  - JNDI name: `jdbc/Bank`
  - Database name: `${APP_INSTALL_ROOT}/${CELL}/WebSphereBank.ear/Database/BankDB`
- Data source connection factory
  - Scope: server level
  - JDBC provider: `Samples Cloudscape JDBC Provider (XA)`
  - Datasource name: `BANKDS`
  - Name: `BANKDS_CF`
  - Authentication mechanism: BASIC_PASSWORD
  - Authentication alias: N_O_N_E
  - Interface: javax.resource.cci.ConnectionFactory
- Service integration bus
  - Scope: node level
  - Name: `<nodename>SamplesBus`
- JMS connection factory
  - Scope: node level
  - Name: `BankJMSConnFactory`

Archived

  - JNDI name: jms/BankJMSConnFactory
  - Authentication alias: <*cell_name*>/samples
  - Service Integration Bus name: <*node_name*>SamplesBus
- JMS queue
  - Scope: node level
  - Name: BankJMSQueue
  - JNDI name: jms/BankJMSQueue
  - Service Integration Bus queue: BankJSQueue
- Service integration bus queue
  - Scope: server level
  - Name: BankJSQueue
  - Service Integration Bus: <*node_name*>SamplesBus
- JMS activation spec
  - Scope: node level
  - Name: BankActivationSpec
  - JNDI name: eis/BankActivationSpec
  - Destination JNDI name: jms/BankJMSQueue
  - Destination type: javax.jms.Queue
  - Authentication alias: <*cell_name*>/samples
  - Service integration bus: <*node_name*>SamplesBus

The resources can be configured automatically by running the WebSphere Bank install script, which uses wsadmin. When you first access the WebSphere samples, you will see two options in the navigation panel on the left: Installed Samples and Installable Samples. Expand **Installable Samples** and select **WebSphere Bank** under Applications. This will show you information about how to run the script.

You can also define the environment manually by inspecting the installation scripts and using the administrative console to define the resources.

15.2 Packaging using the Application Server Toolkit

To illustrate packaging techniques, we will import the WebSphere Bank application into the Application Server Toolkit and go through the various aspects of packaging an application for deployment. The application can then be exported as an EAR file for deployment.

Archived

15.2.1 Preparing the sample code

In our example, we used the WebSphere Bank sample application shipped with WebSphere Application Server V6. To obtain sample code suitable for import to the Application Server Toolkit, we used the documentation available with the Samples Gallery to rebuild the sample code using ANT. The steps required to prepare the code for import on a Windows system are as follows:

1. Create an application server profile. The new profile includes the sample code and the build script.
2. Open a command prompt and change to this directory:

   ```
   <profile_root>\samples\src\WebSphereBank
   ```

3. Run the Sample build script:

   ```
   <profile_root>\samples\bin\WebSphereBank\buildbank.bat
   ```

   The script will remove the deploy code from the sample and repackage the EAR file. The resulting EAR file will be in the following directory:

   ```
   <profile_root>\samples\lib\WebSphereBank
   ```

15.2.2 Importing an EAR file

To work with the WebSphere Bank EAR file in the Application Server Toolkit, we first need to import it.

1. Select **Start** →**Programs** → **IBM WebSphere Application Server Toolkit, V6.0** → **Application Server Toolkit**.
2. When asked for a default location for a workspace, browse to a suitable directory and then click **OK**. Do not check the Use this as the default and do not ask again check box.
3. When the toolkit has launched, close the Welcome page by clicking the X in the Welcome tab, as in Figure 15-2.

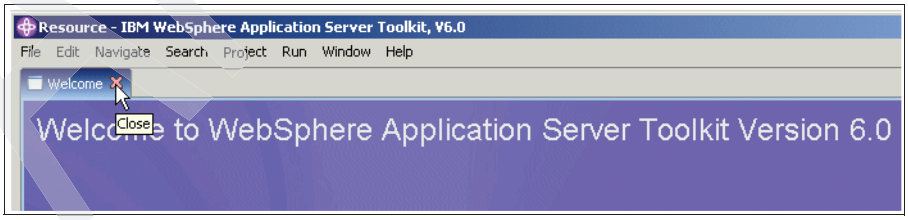


Figure 15-2 Welcome page

4. Select **File** →**Import**.
5. Select **EAR file** from the list, click **Open**, and then click **Next**.

Archived

6. Click **Browse,** select the **WebSphereBank.ear** file and click **Open**. The file is located in the <*was_home*>\samples\lib\WebSphereBank directory. Then click **Next**.
7. On the second page, accept the defaults with nothing selected and click **Next**.
8. On the third page, accept the defaults and click **Finish**. This will import the code into your workspace.
9. When the code has been imported, click **Yes** on the dialog box asking you to switch to the J2EE perspective.

> **Note:** When you import an EAR file, the workspace rebuilds the project. The build process runs as a separate background thread and can take a minute or two, depending on the application. In the lower right corner of the Application Server Toolkit window a progress indicator tells you what is happening. When rebuild is complete, any error messages will appear in the Problems view.

The Project explorer view in Figure 15-3 shows the EAR file contents organized by module.

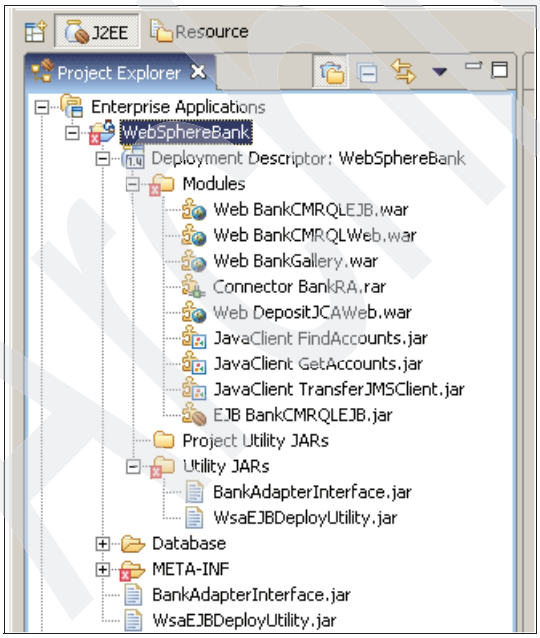


Figure 15-3 WebSphere Bank modules list

As you can see, it has the modules shown in Figure 15-1 on page 848.

If you click the **Problems** tab you see that the workspace has a lot of problems. See Figure 15-4. WebSphere V6 samples were not built using Application Server Toolkit or Rational Application Developer and do not have the extra meta information about the classpaths these tools need. Before doing anything else, we need to remove these problems.

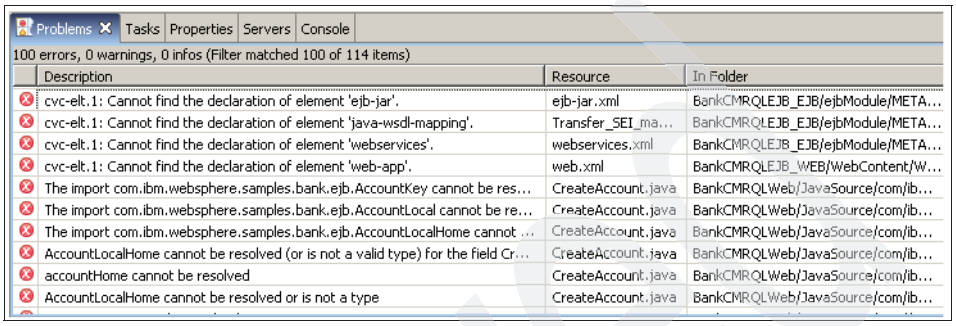


Figure 15-4 Problems after importing WebSphere Bank EAR file

1. In the Project Explorer, expand **Dynamic Web Projects**.
2. Select and right-click the **BankCMRQLWeb** and select **Properties**. See Figure 15-5 on page 854.

Archive

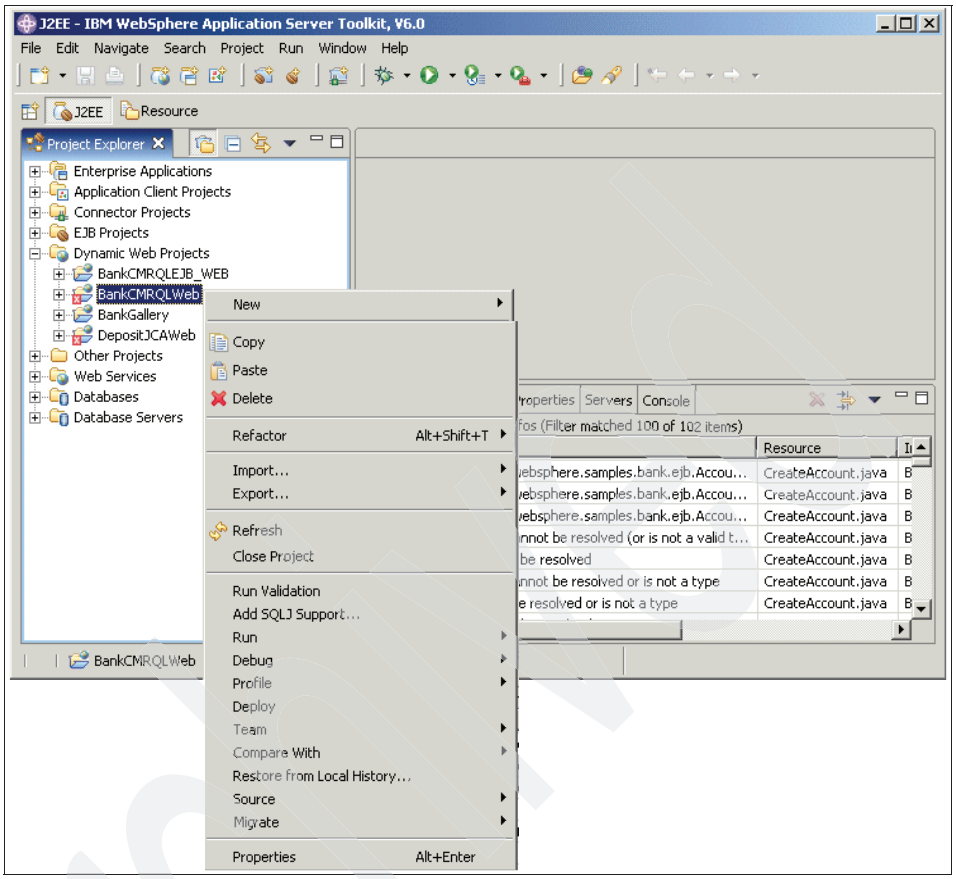


Figure 15-5 Open the properties for a Web module

3. Select the **Java Build Path** section and click the **Projects** tab.
4. Check the **BankCMRQLEJB_EJB** project in the list as shown in Figure 15-6 on page 855 and click **OK**.

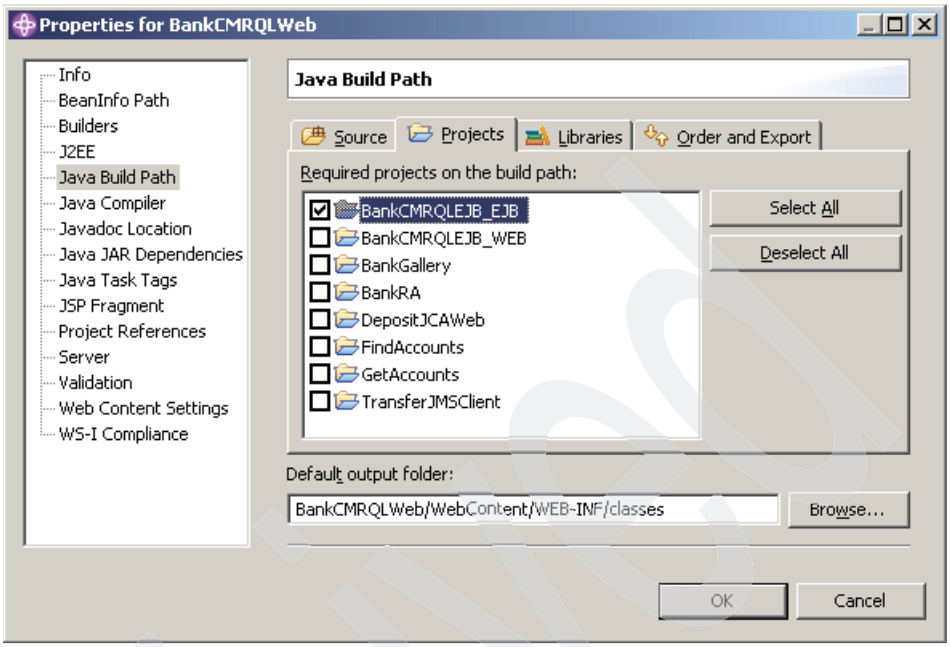


Figure 15-6 Configure Java Build Path for BankCMRQLWeb project

5. Next we need to configure the build path also for the DepositJCAWeb project. Right-click the **DepositJCAWeb** project and select **Properties**.
6. Select the **Java Build Path** section and click the **Projects** tab.
7. Check the **BankRA** project in the list as shown in Figure 15-7 on page 856 and click **OK**.

Archive

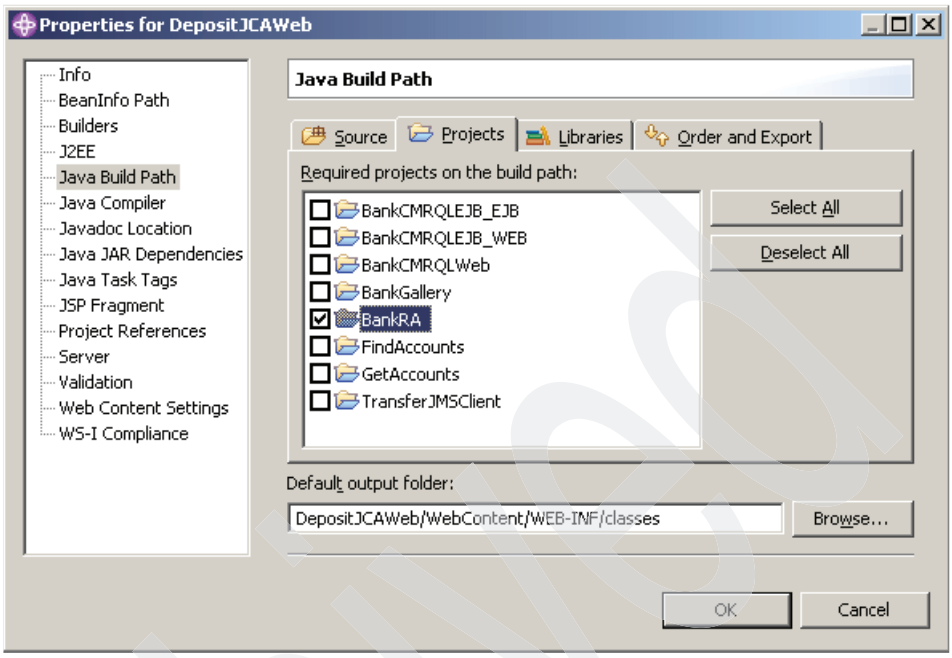


Figure 15-7 Configure Java Build Path for DepositJCAWeb project

This reduces the number of problems to no errors, three warnings and one informational message. We do not need to care about these warnings because the EAR file now builds properly and should work fine for our purposes.

Tip: When using the Application Server Toolkit, keep in mind the following:

- To perform a complete rebuild of your project(s) select **Project** →**Clean** and then select either to clean current or all projects. This will remove all build problems from the Problems view and perform a complete re-build of the selected projects. This sometimes removes errors and warnings in the Problems view.
- As you update and save modules in the toolkit, the contents of the modules are automatically validated and problems are listed in the Tasks view. You can also manually invoke validation of modules by selecting any module and choosing **Run Validation** from the context menu. To verify the settings for validation, select **Window** →**Preferences** and click **Validation**.

15.2.3 Working with deployment descriptors

Information describing a J2EE application and how to deploy it into a J2EE container is stored in XML files called deployment descriptors. An EAR file normally contains multiple deployment descriptors, depending on the modules it contains. Figure 15-8 shows a schematic overview of a J2EE EAR file. In this figure the various deployment descriptors are designated with DD after their name.

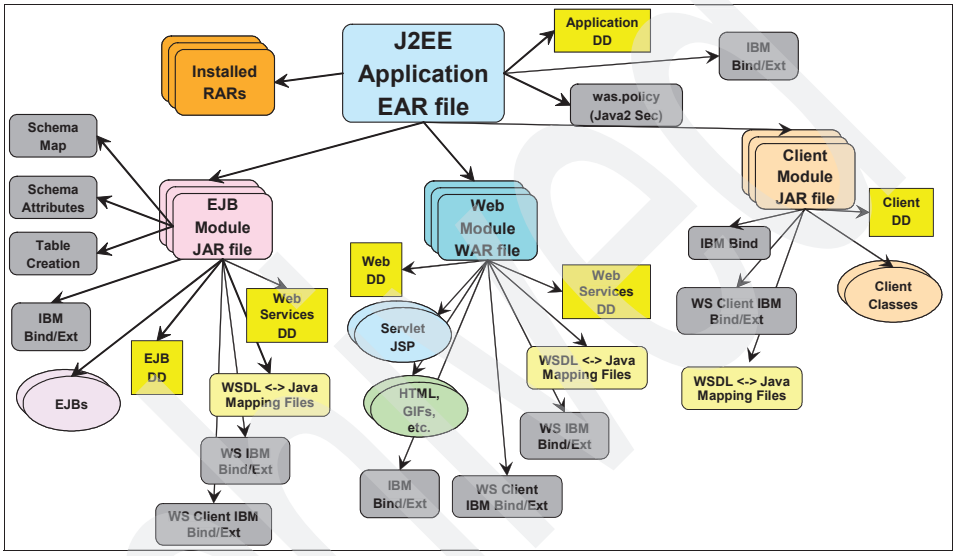


Figure 15-8 J2EE EAR file structure

The deployment descriptor of the EAR file itself is stored in the META-INF directory in the root of the EAR and is called application.xml. It contains information about the modules which make up the application.

The deployment descriptors for each module are stored in the META-INF directory of the module and are called web.xml (for Web modules), ejb-jar.xml (for EJB modules), ra.xml (for resource adapter modules) and application-client.xml (for application client modules). These files describe the contents of a module and allow the J2EE container to configure things like servlet mappings, JNDI names and so forth.

Classpath information specifying which other modules and utility JARs are needed for a particular module to run, is stored in the manifest.mf file also in the META-INF directory of the modules.

In addition to the standard J2EE deployment descriptors, EAR files produced by the Application Server Toolkit can also include additional WebSphere-specific

information used when deploying applications to WebSphere environments. This supplemental information is stored in files called ibm-xxx-xxx-xxx.xmi, also in the META-INF directory of the respective modules. Examples of information in the IBM-specific files are IBM extensions like servlet reloading and EJB access intents.

New in WebSphere Application Server V6 is also the information contained in the Enhanced EAR files. This information, including settings for the resources required by the application, is stored in an ibmconfig subdirectory of the EAR file's META-INF directory. In the ibmconfig directory are the directories for a WebSphere cell configuration.

The Application Server Toolkit has easy-to-use editors for working with all deployment descriptors. The information that goes into the different files are shown on one page in the GUI, eliminating the need to be concerned about what information is put into what file. However, if you are interested, you can click the Source tab of the deployment descriptor editor to see the text version of what is stored in that descriptor. For example, if you open the EJB deployment descriptor, you will see settings that are stored across multiple deployment descriptors for the EJB module, including:

- The EJB deployment descriptor, ejb-jar.xml
- The extensions deployment descriptor, ibm-ejb-jar-ext.xmi
- The bindings file, ibm-ejb-jar-bnd.xmi files
- The access intent settings, ibm-ejb-access-bean.xmi

To work with a deployment descriptor, do the following:

1. Open the J2EE perspective.
2. In the J2EE Project Explorer view, expand the project category (EJB Projects), and then expand the module you want to work with.
3. Double-click the **Deployment Descriptor** to open the editor for it. See Figure 15-9 on page 859.

Archive

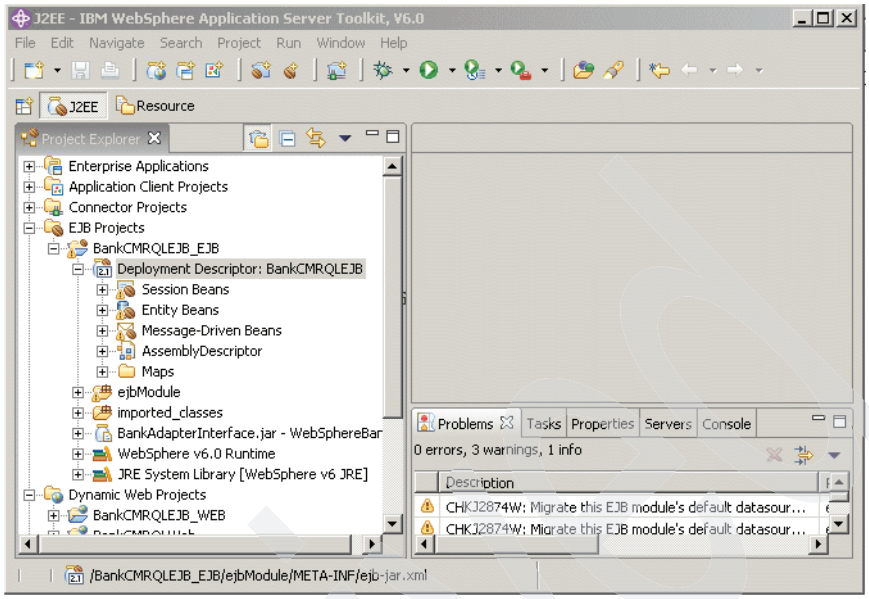


Figure 15-9 Finding the deployment descriptor

4. Figure 15-10 on page 860 shows the deployment descriptor for the WebSphere Bank EJB module, BankCMRQLEJB, open with the deployment descriptor editor.

Archive

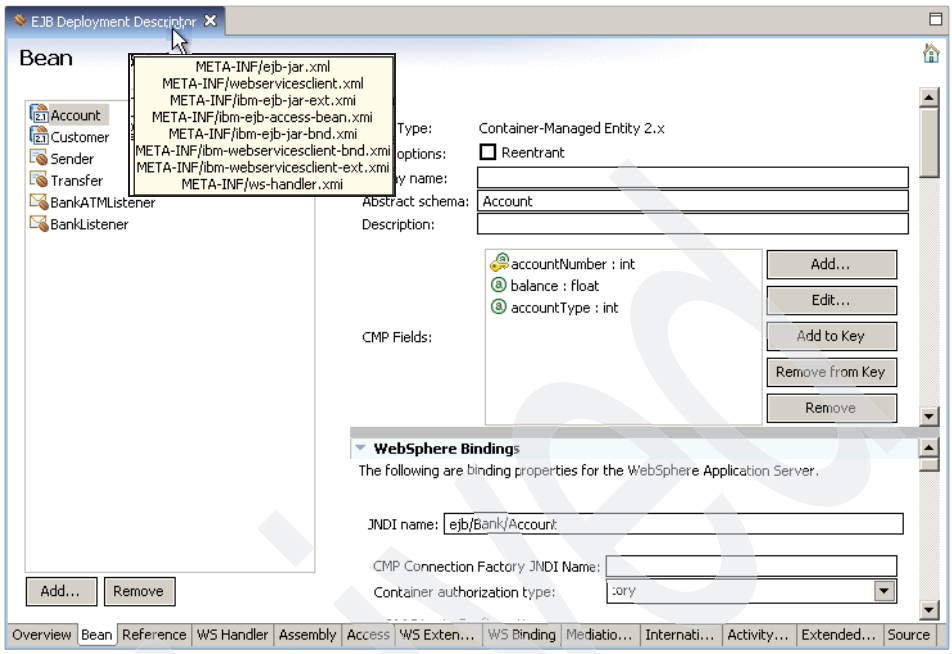


Figure 15-10 EJB deployment descriptor

While the editor shows you information stored in all the relevant deployment descriptor files on the appropriate tabs, the Source tab only shows you the source of the deployment descriptor itself (for example, ejb-jar.xml or web.xml) and not the IBM extensions and bindings stored in the WebSphere-specific deployment descriptor files. If you want to view the results of updates to those files in the source, open each file individually. By hovering over the EJB Deployment Descriptor caption tab you can see the different files that makes up the EJB deployment descriptor you are editing. The descriptor files are kept in the META-INF directory of the module you are editing.

When you have made changes to a deployment descriptor, save it by pressing Ctrl+S and then close it.

15.3 Setting application bindings

At packaging time, you create references to resources. For an application to run, you need to bind these references to the real resources, such as JDBC data sources, created from the administrative console. This needs to be done for EJB references and resource references. You also need to define the enterprise bean's JNDI names, and security roles.

Bindings can be defined at development or deployment time. Most likely, developers will deliver a preconfigured EAR file which will then be modified at deployment time by the deployment team to suit the target environment. Developers use a tool like Rational Application Developer to define bindings, while deployers use the Application Server Toolkit.

All binding definitions are stored in the ibm-xxx-bnd.xmi files, where xxx can be ejb-jar, web, application, or application-client.

In the next steps, you define the following bindings using the Application Server Toolkit:

- EJB JNDI names
- EJB references
- ActivationSpecs for message-driven beans
- Data source for entity beans

All sections below assume that you have started the Application Server Toolkit and opened the WebSphere Bank application EAR file.

15.3.1 Defining EJB JNDI names

For each session and entity enterprise bean, you must specify a JNDI name. This name is used to bind the EJB home object to an entry in the global JNDI name space. The bind happens automatically when the application server starts.

The WebSphere Bank enterprise beans are declared to be bound in the ejb/Bank/ subcontext. For clarity, we recommend that you place all enterprise bean JNDI names for an application in a separate subcontext. You can find the JNDI names for the WebSphere Bank session and entity EJBs in Table 15-1.

Table 15-1 Webbank enterprise bean JNDI names

EJB Name	JNDI Name
Sender session bean	ejb/Bank/Sender
Transfer session bean	ejb/Bank/Transfer
Account entity bean	ejb/Bank/Account
Customer entity bean	ejb/Bank/Customer

Use this table and the instructions below to define a JNDI name for each WebSphere Bank enterprise bean:

1. In the Project Explorer view, expand the EJB Projects section.
2. Expand the **BankCMRQLEJB_EJB** module and then expand the S**ession Beans** and **Entity Beans** sections under Deployment Descriptor.

3. Double-click the **Sender** session bean. The EJB deployment descriptor editor opens to the Bean page, shown in Figure 15-11.
4. Look for the WebSphere Bindings section in the editor.
5. In the JNDI name field, enter `ejb/Bank/Sender`.

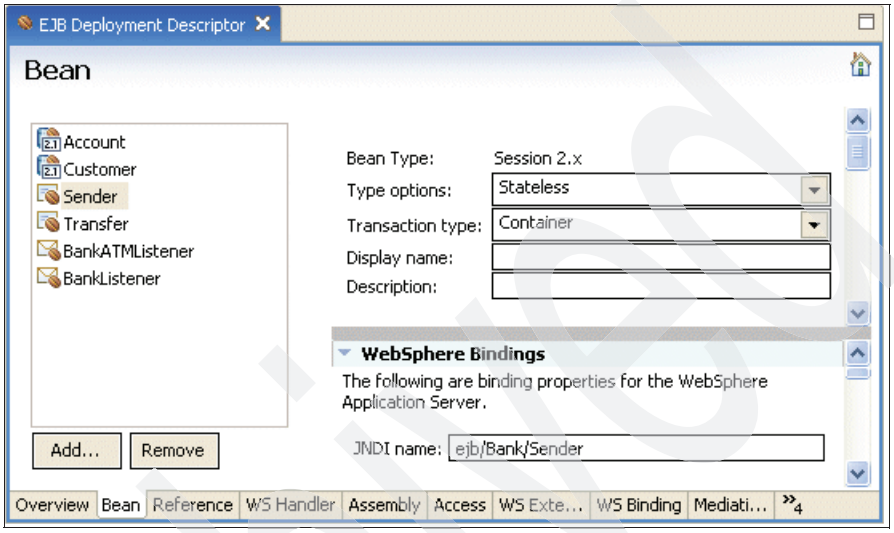


Figure 15-11 Defining EJB JNDI names

6. Repeat these steps for each of the session and entity enterprise beans in the EJB module, that is, Transfer, Account and Customer.
7. Save the deployment descriptor.

15.3.2 Binding EJB and resource references

An EJB client can define *EJB references*, logical names or nicknames, used by the client to find the EJB homes. When using references, the client can hard code the name of the reference. Then, during deployment, the reference is mapped to the real name in the JNDI namespace to which the EJB is bound. A reference to an EJB specifies either the Local or Remote home of the EJB.

For the WebSphere Bank sample, we use the resource references shown in Table 15-2 on page 863.

Table 15-2 EJB and resource references : JNDI names list

EJB/Resource Reference	Corresponding JNDI Name
`ejb/Account`	ejb/Bank/Account
`ejb/Customer`	ejb/Bank/Customer
`ejb/Transfer`	ejb/Bank/Transfer
`ejb/CustomerHome`	ejb/Bank/Customer

The last reference, `ejb/CustomerHome`, is a reference to the remote home interface of the Customer entity EJB. All other are references to their respective EJB's local home.

Follow these steps to bind an EJB local reference to a JNDI name:

1. In the Project Explorer view, expand the **Dynamic Web Projects**.
2. Expand the **BankCMRQLWeb** module and then double-click the deployment descriptor. The editor opens to the overview.
3. Click the **References** tab.
4. Click the **ejb/Account** reference, as shown in Figure 15-12.

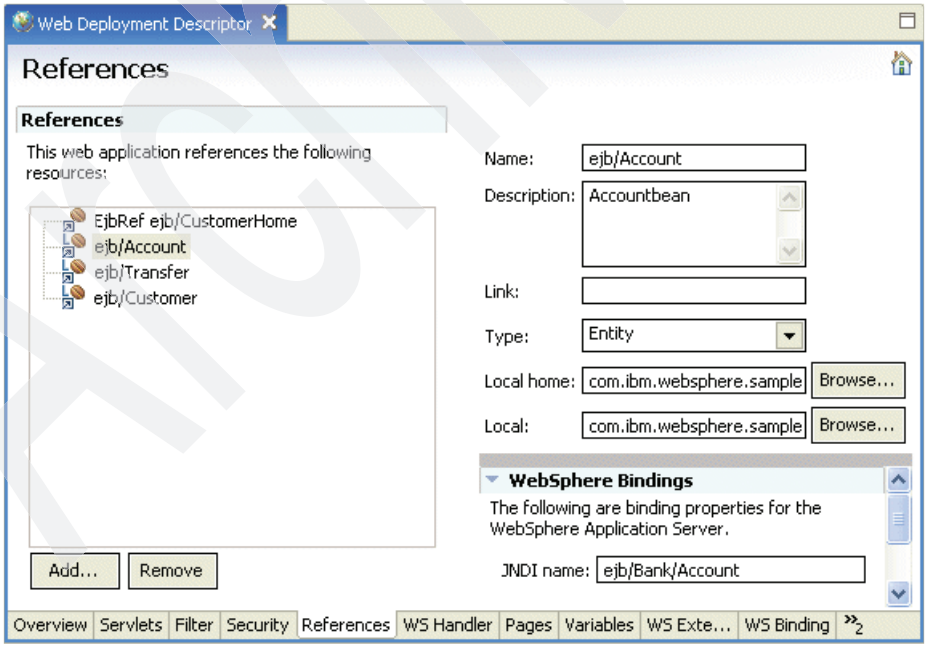


Figure 15-12 Setting EJB References bindings

5. In the WebSphere bindings section, specify `ejb/Bank/Account`.
6. Repeat these steps for all references in this Web module, using Table 15-2 as a guide for JNDI and resources names.
7. Save the deployment descriptor.

Note: EJB reference bindings can be defined or overridden at deployment time in the administrative console for all modules except for application clients, for which you must use the Application Server Toolkit.

15.3.3 Binding the message-driven bean to an ActivationSpec

The WebSphere Bank sample includes two message-driven beans. The BankListener is a JMS-type bean and the BankATMListener is a non-JMS-type bean. These message-driven beans must be bound to ActivationSpecs so they can be activated when an incoming message for them arrives.

To do this, follow these steps:

1. In the Project Explorer view, expand the EJB Projects section.
2. Expand the **BankCMRQLEJB_EJB** module and double-click the deployment descriptor.
3. Click the **Bean** tab at the bottom of the editor and select the **BankListener** bean, as shown in Figure 15-13 on page 865.

Archive

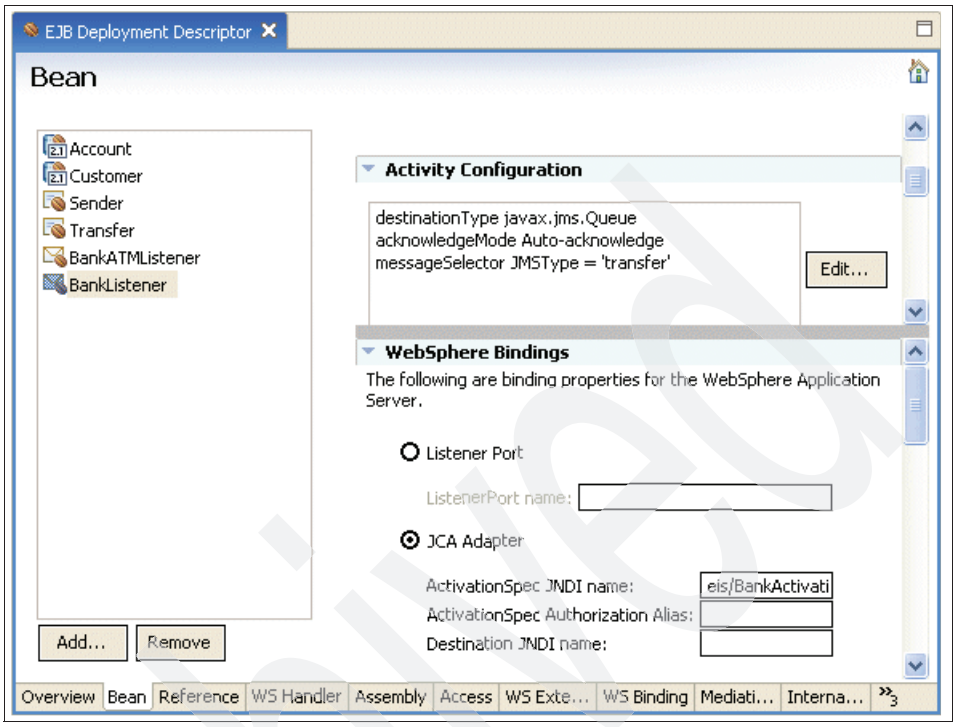


Figure 15-13 Configuring an ActivationSpec for a Message-driven bean

4. In the WebSphere bindings section, set the ActivationSpec JNDI name to `eis/BankActivationSpec`.
5. Repeat steps 3 on page 864 and 4 for the BankATMListener, but use the ActivationSpec JNDI name `eis/com.ibm.websphere.samples.bank.adapter.BankMessageListener`.
6. Save the deployment descriptor file.

All bindings are now complete. Save the ready-to-deploy EAR file.

15.3.4 Defining data sources for entity beans

The entity beans in the WebSphere Bank application are container-managed (CMP) EJBs. The EJB container handles the persistence of the EJB attributes in the underlying persistent store. You must specify which data store to use. This is done by binding an EJB module or an individual EJB to a data source. If you bind the EJB module to a data source, all EJBs in that module use the same data source for persistence. If you specify the data source at the EJB level, then this data source is used instead.

Archived

For the WebSphere Bank application, the data source is bound at the EJB module level. The data source configured for the EJB must match the data source configured in the WebSphere environment. The JNDI name for this data source is `jdbc/Bank`. See “WebSphere Bank resources used” on page 849.

To bind the WebSphere Bank EJB module to this data source, follow these steps:

1. In the Project Explorer view, expand the **EJB Projects** section.
2. Expand the **BankCMRQLEJB_EJB** module and then double-click the deployment descriptor.
3. In the Overview tab, scroll down and find the WebSphere bindings section, as in Figure 15-14. Check the following fields:

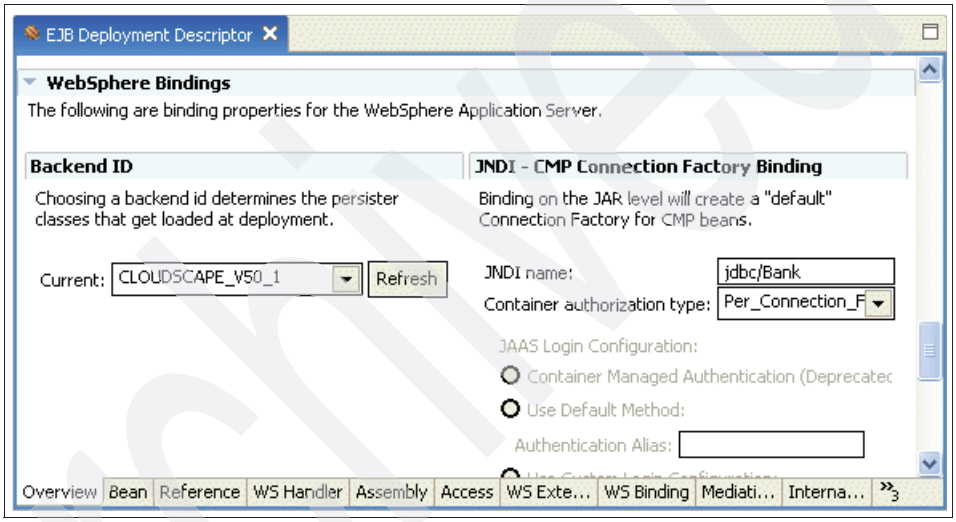


Figure 15-14 Specifying the default CMP data source for the entity EJBs

   – Backend ID

     In EJB 2.x, mapping and schema files make up a back end for EJB 2.x projects. The WebSphere Bank sample ships with a Cloudscape back end defined. Leave this for now, though in “Creating a new database mapping and schema” on page 867, we will create a new backend for DB2 and change this binding to point it.

   – **JNDI name**

     Enter `jdbc/Bank` in the JNDI name field. This is the value the application uses to access the database. Note that this is the same, regardless of which backend ID is used.

– **Container authorization type**

  Select **Per_Connection_Factory** for the Container authorization type.

4. Save the deployment descriptor.

Tip: An EJB JAR can contain database mappings and EJB deployed code for multiple databases. Currently the WebSphere Bank sample application contains a database mapping and EJB deployed code only for Cloudscape 5, but we will also add it for DB2 UDB. You can set which backend ID will be used at runtime in the WebSphere bindings section. This choice can also be overriden at deployment time.

Creating a new database mapping and schema

The WebSphere Bank sample is configured to run against a Cloudscape database. However, for the purpose of showing how to create a new database backend and deployed code, we will configure it so it can also run against a DB2 database.

Creating the database mapping

First you need to create a database mapping using the EJB project. To do this, perform the following steps:

1. Expand the **EJB Projects** section
2. Right-click the **BankCMRQLEJB_EJB** project and select **EJB to RDB Mapping →Generate Map**
3. Select **Create a new backend folder** and click **Next**.
4. Select **Top-down** and click **Next**.
5. On the Top-down mapping options page, select **DB2 Universal Database Express V8.2**, or the corresponding DB2 product and version you are using as the target database. Enter database name `BANK` and leave `NULLID` as the schema name. See Figure 15-15 on page 868.

Archive

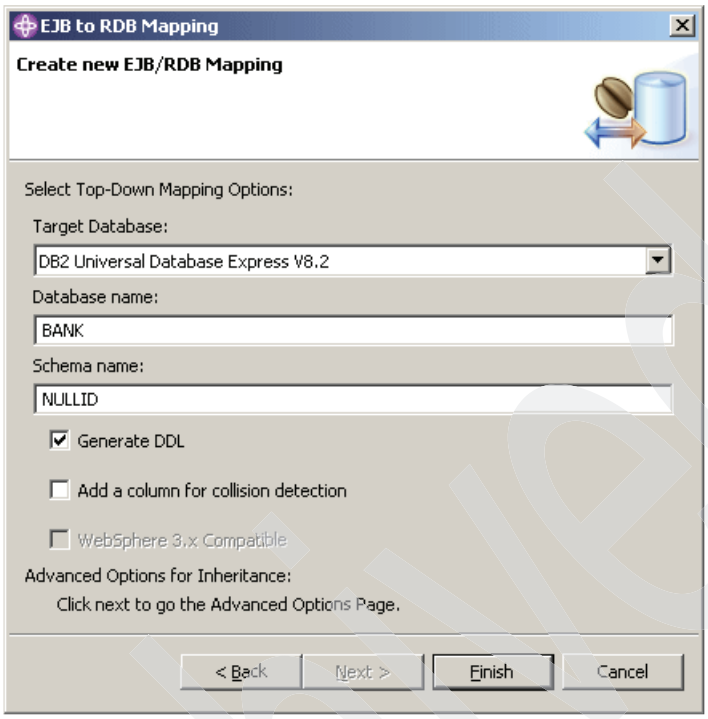


Figure 15-15 Generating a DB2 mapping

6. Click **Finish**.

   A Table.ddl file containing the script to set up the DB2 tables is created in the BankCMRQLEJB_EJB\ejbModule\META-INF\backends\DB2EXPRESS_V82_1 directory of the Application Server Toolkit workspace. You will need this script when creating the DB2 BANK database before deploying WebSphere Bank. The file has the commands shown in Example 15-1.

Example 15-1 WebSphere Bank Table.ddl

```
CREATE TABLE ACCOUNT
  (ACCOUNTNUMBER INTEGER NOT NULL,
   BALANCE REAL NOT NULL,
   ACCOUNTTYPE INTEGER NOT NULL,
   ACCOUNTSCUSTOMERINVERSE_CUSTO2 BIGINT);

ALTER TABLE ACCOUNT
  ADD CONSTRAINT PK_ACCOUNT PRIMARY KEY (ACCOUNTNUMBER);

CREATE TABLE CUSTOMER
  (CUSTOMERNUMBER BIGINT NOT NULL,
   FIRSTNAME VARCHAR(250),
```

```
   TAXID VARCHAR(250),
   LASTNAME VARCHAR(250));

ALTER TABLE CUSTOMER
  ADD CONSTRAINT PK_CUSTOMER PRIMARY KEY (CUSTOMERNUMBER);
```

7. The Application Server Toolkit database mapping editor (Map.mapxmi editor) opens allowing you to make adjustments to the mapping between the fields of the entity EJBs and the database columns. We do not need to do that so **close** the editor.
8. Right-click the **BankCMRQLEJB_EJB** project again and select **Deploy**. This generates the EJB deployed code, as shown in Figure 15-16.

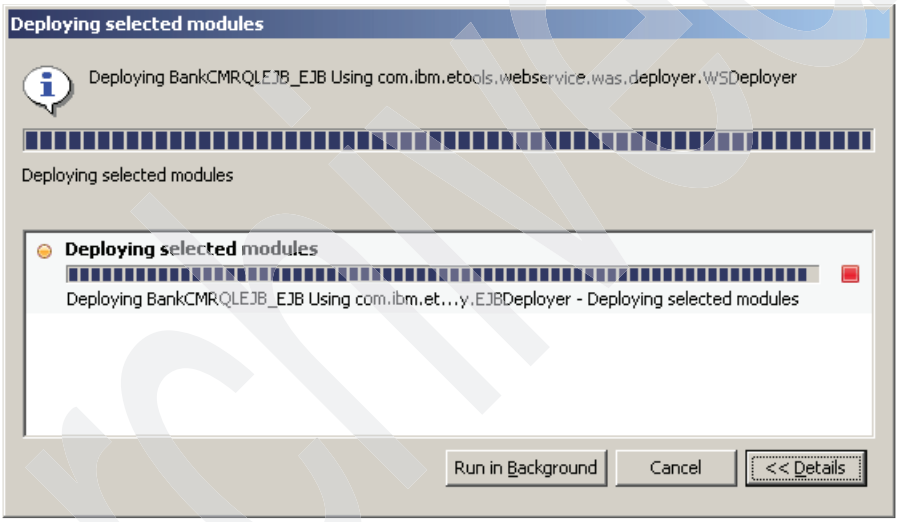


Figure 15-16 Generating EJB deployed code

If you receive the following error during the deploy code generation you can ignore it:

```
Deployment from com.ibm.etools.webservice.was.deployer.WSDeployer had
errors: Deployment error:
```

Change the backend ID

Because we have now created a new database backend map, we can set the default backend map for the EJB to the newly created DB2 map. To map to the new DB2 map, do the following:

Archive

1. Open the deployment descriptor for the EJB module, scroll down do the bottom of the Overview tab and select **DB2EXPRESS_V82_1** as the Current Backend ID as shown in Figure 15-17.

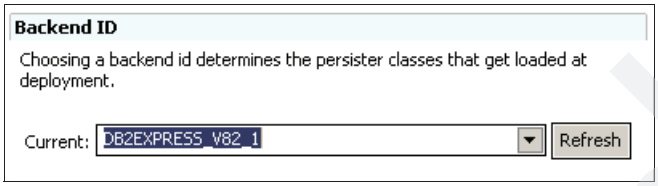


Figure 15-17 Setting default backend id for EAR file

2. Press **Ctrl-S** to save the deployment descriptor.

15.4 IBM EJB extensions: EJB caching options

This section discusses the caching options for entity and stateful session beans.

15.4.1 EJB container caching option for entity beans

The Enterprise JavaBeans specification defines three EJB caching options: options A, B, or C. Those options define how the EJB container handles entity bean instances between transactions. EJB caching options are set at the bean level, and are part of the IBM extensions deployment descriptor.

Caching option A

With caching option A, you assume that the entity bean has *exclusive* access to the underlying persistent store. In other words, between transactions, no one can modify the data. This includes a batch program updating the data, a Java application updating the data, or even the same entity bean running in a different container. This implies option A cannot be used in a clustered environment (WLM). Note that it is your responsibility to ensure no other application will modify the data, as the EJB container has no way to control write access to the underlying database from other servers.

When caching option A is used, the entity bean instance is kept in a memory cache across transactions. At transaction commit, the entity bean attributes are synchronized with the underlying persistent store, and the bean instance remains cached in memory.

If you were tracing the calls made by the container, you would see something similar to Example 15-2 on page 871. The first time the entity bean is used, its runtime context is set (step 1), a bean is taken from the entity beans instance

pool (step 2), the bean instance attributes are synchronized with the underlying data store (step 3), the method setBalance is invoked on the bean (step 4), and, finally, the bean attributes are saved back to the database (step 5). The bean is not returned to the pool. On subsequent calls, the setBalance method is invoked directly on the cached bean instance, and the bean attributes are synchronized with the underlying persistent data store.

Example 15-2 Entity beans call trace with option A caching

```
Transaction 1 (Begin)
Step 1: 1c9585f1 BranchAccount E called setEntityContext() method
Step 2: 1c9585f1 BranchAccount E called ejbActivate() method
Step 3: 1c9585f1 BranchAccount E called ejbLoad() method
Step 4: 1c9585f1 BranchAccount E called setBalance() method
Step 5: 1c9585f1 BranchAccount E called ejbStore() method
Transaction 1 (Commit)

Transaction 2 (Begin)
Step 1: 284485f1 BranchAccount E called setBalance() method
Step 2: 284485f1 BranchAccount E called ejbStore() method
Transaction 2 (Commit)
```

Using caching option A can provide some performance enhancements at the expense of higher memory usage. You should only use it if you do not intend to use WebSphere clustering capabilities and you mostly access data in read mode.

Caching option B

With caching option B, you assume that you have *shared* access to the underlying database. This means the data could be changed by another application between transactions. When option B is used, the bean instance attributes are always synchronized with the underlying back-end data store at the beginning of every transaction. Similar to Option A, the bean is kept in the cache between transactions. Therefore, if you were tracing the different calls made in Option B, you would obtain the trace shown in Example 15-3.

Example 15-3 Entity beans call trace with option B caching

```
Transaction 1 (Begin)
Step 1: 1c9585f1 BranchAccount E called setEntityContext() method
Step 2: 1c9585f1 BranchAccount E called ejbActivate() method
Step 3: 1c9585f1 BranchAccount E called ejbLoad() method
Step 4: 1c9585f1 BranchAccount E called setBalance() method
Step 5: 1c9585f1 BranchAccount E called ejbStore() method
Transaction 1 (Commit)

Transaction 2 (Begin)
```

```
Step 1: 284485f1 BranchAccount E called ejbLoad() method
Step 2: 284485f1 BranchAccount E called setBalance() method
Step 3: 284485f1 BranchAccount E called ejbStore() method
Transaction 2(Commit)
```

Caching option B can be safely used in a clustered environment, or when you are not sure if you have exclusive access to data. You are assured that you always work with the last committed data. Option B memory usage is the same as for option A. The performance of both options can slightly differ depending on the nature of your application.

Caching option C

Similar to option B, caching option C assumes *shared* access to the database. Unlike option B or A, the bean instance is returned to the entity beans pool at the end of the transaction. A new bean instance is used at the beginning of every transaction. Each transaction results in the sequence of calls shown in Example 15-4.

Example 15-4 Entity beans call trace with option C caching

```
Transaction (Begin)
Step 1: 1c9585f1 BranchAccount E called setEntityContext() method
Step 2: 1c9585f1 BranchAccount E called ejbActivate() method
Step 3: 1c9585f1 BranchAccount E called ejbLoad() method
Step 4: 1c9585f1 BranchAccount E called setBalance() method
Step 5: 1c9585f1 BranchAccount E called ejbStore() method
Step 6: 1c9585f1 BranchAccount E called ejbPassivate() method
Step 7: 1c9585f1 BranchAccount E called unsetEntityContext() method
Transaction (Commit)
```

Caching option C has the best memory usage at the expense of a larger number of methods calls. This is the default behavior.

How to set the EJB caching option

You must combine the *Activate at* and *Load at* options to set the EJB caching option to A, B, or C. Use Table 15-3 to choose the right combination.

Table 15-3 Setting entity EJB caching properties

Option	Activate at must be set to	Load at must be set to
Option A	Once	Activation
Option B	Once	Transaction
Option C (default)	Transaction	Transaction

To set the EJB caching option, do the following:

1. Open the EJB deployment descriptor.
2. Switch to the **Bean** tab.
3. Select the entity bean in the window to the left, then scroll down the options at right until you see the Bean Cache settings under the WebSphere extensions section, as in Figure 15-18.

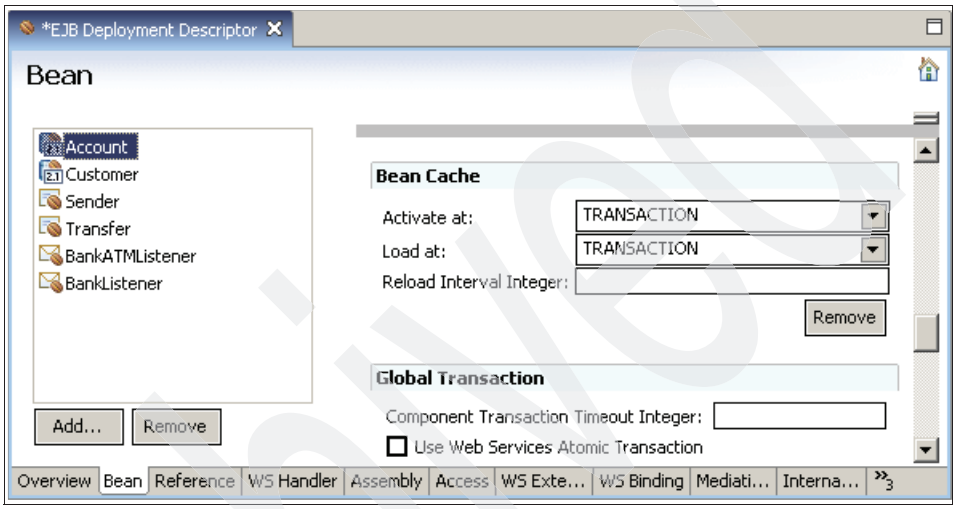


Figure 15-18 Setting the activate and load settings for entity beans

4. Select the Activate at and Load at options according to Table 15-3 on page 872.
5. Close and save the deployment descriptor.

The settings are saved in the ibm-ejb-jar-ext.xmi file. They correspond to the following line:

```
<beanCache xmi:id="BeanCache_1" activateAt="ONCE" loadAt="TRANSACTION"/>
```

There is one line for each entity bean for which you have set this option.

15.4.2 EJB container caching option for stateful session beans

Similarly to entity beans, you can specify which caching strategy to use for stateful session beans. This caching option specifies the point at which an

Archived

enterprise bean is activated and placed in the cache. Removal from the cache and passivation are also governed by this setting. Valid values are:

- **Once** (default)

  Choosing Once indicates that the bean is activated when it is first accessed in the server process. It is passivated and removed from the cache at the discretion of the container, for example, when the cache becomes full.

- **Transaction**

  Choosing Transaction indicates that the bean is activated at the start of a transaction. It is passivated and removed from the cache at the end of the transaction.

You can set this caching option by opening the EJB deployment descriptor for the EJB module. The **Activate at** setting is found on the Bean tab (Figure 15-19). Select the bean and scroll down to the Bean Cache category.

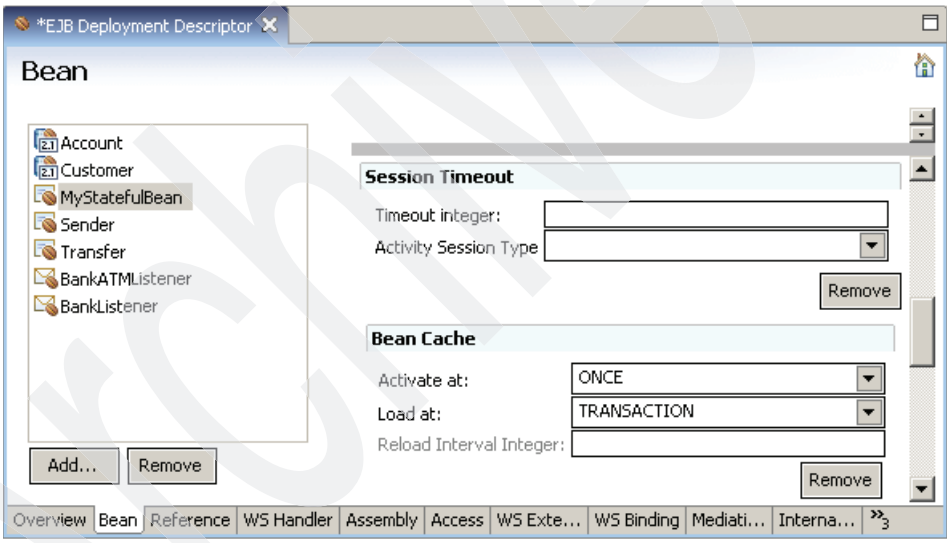


Figure 15-19 Activate settings for stateful session beans

15.4.3 Stateful EJB timeout option

Additionally, you can specify a timeout value for stateful session beans. A bean can time out in the METHOD_READY or in the PASSIVATED state. If you try to access a bean that has timed out, you see an exception similar to that in Example 15-5 on page 875:

Example 15-5 Stateful EJB timed out exception

```
com.ibm.ejs.container.SessionBeanTimeoutException: Stateful bean
StatefulBean0(BeanId(Webbank#webbankEJBs.jar#Transfer, ebf64d846a), state =
METHOD_READY) timed out.
```

Session beans that have timed out can be removed by the container, for example if it needs to free memory. However, a well-written application should not rely on beans to time out to free memory. Instead, it is important that the developer explicitly calls remove() on a bean when this stateful bean is not needed anymore.

The default timeout is 600 seconds. You can set the timeout integer value as a parameter of a stateful session bean by opening the EJB deployment descriptor and selecting the **Bean** tab as in Figure 15-19 on page 874. By specifying a value of 0, you set the bean to never expire.

Setting this timeout inserts the following property in the ejbExtensions tag of the IBM bindings file:

```
<ejbExtensions xmi:type="ejbext:SessionExtension"
   xmi:id="Session_1_Ext" timeout="120">
```

Note: If a bean times out in the METHOD_READY state and is consequently removed by the container, the ejbRemove() method is called on the bean instance. If a bean times out in the passivated state, ejbRemove() is not called, according to the EJB specification.

15.5 IBM EJB extensions: EJB access intents

Access intents are used to optimize the access to relational data. Access intents are only applicable to EJB 2.x beans. For EJB 1.1 beans, you still use the old method of setting the transaction isolation level at the method level, as well as mark methods as read-only. In this section we only cover access intents for EJB 2.x beans.

Access intent policies are specifically designed to supplant the use of isolation level and read-only, method-level modifiers found in the extended deployment descriptor for EJB Version 1.1 enterprise beans. You cannot specify isolation level and read-only modifiers for EJB Version 2.x enterprise beans. The WebSphere persistence manager uses access intent hints to make decisions about isolation level, cursor management, and more.

Archived

15.5.1 Transaction isolation levels overview

Transaction isolation levels provide a trade-off between accuracy of reads versus concurrent readers. The levels can best be described by the types of read anomalies they permit and forbid. Consider the read anomalies that can occur with two concurrent transactions, T1 and T2:

- Dirty read

  T1 reads data that has been modified by T2, before T2 commits.

- Non-repeatable read

  Non-repeatable read is caused by fine-grained locks.

  – T1 reads a record and drops its lock.
  – T2 updates.
  – T1 re-reads different data.

- Phantom read

  This is a non-repeatable read involving a range of data and inserts or deletes on the range.

  – T1 reads a set of records that match some criterion.
  – T2 inserts a record that matches the criterion.
  – T1 continues processing the set, which now includes records that were not part of the original matching set.

There are four possible settings for the transaction isolation level:

- Repeatable read (`TRANSACTION_REPEATABLE_READ`)

  This setting permits phantom reads and forbids both dirty and unrepeatable reads.

- Read committed (`TRANSACTION_READ_COMMITTED`)

  This setting permits non-repeatable and phantom reads and forbids dirty reads.

- Read uncommitted (`TRANSACTION_READ_UNCOMMITTED`)

  This setting permits all the read anomalies including dirty reads, non-repeatable reads, and phantom reads.

- Serializable (`TRANSACTION_SERIALIZABLE`)

  This setting forbids all the read anomalies.

The container applies the isolation level as follows:

- For entity beans with Container Managed Persistence (CMP), the container generates code that assures the desired level of isolation for each database access.

Archived

- For session beans and Bean-Managed Persistence (BMP) entity beans, the container sets the isolation level at the start of each transaction, for each database connection.

The transaction isolation level is tied to a database connection. The connection uses the isolation level specified in the first bean that uses the connection. The container throws an IsolationLevelChangeException whenever the connection is used by another bean method that has a different isolation level.

Not all databases support all JDBC isolation levels. Moreover, JDBC definitions for isolation levels might not match the database definition of isolation levels. As an example, DB2 definitions for isolation levels follow the naming conventions used in Jim Gray's classic book on transaction processing, *Transaction Processing: Concepts and Techniques*. Table 15-4 shows a mapping between EJB and DB2 isolation levels.

Table 15-4 Mapping JDBC isolation levels to DB2 isolation levels

JDBC isolation level	DB2 isolation level
TRANSACTION_SERIALIZABLE	Repeatable Read
TRANSACTION_REPEATABLE_READ	Read Stability
TRANSACTION_READ_COMMITTED	Cursor Stability
TRANSACTION_READ_UNCOMMITTED	Uncommitted Read

To learn more, refer to the documentation provided by your database product.

15.5.2 Concurrency control

Concurrency control is the management of contention for data resources. A concurrency control scheme is considered *pessimistic* when it locks a given resource early in the data-access transaction and does not release it until the transaction is closed. A concurrency control scheme is considered *optimistic* when locks are acquired and released over a very short period of time at the end of a transaction.

The objective of optimistic concurrency is to minimize the time over which a given resource is unavailable for use by other transactions. This is especially important with long-running transactions, which under a pessimistic scheme would lock up a resource for unacceptably long periods of time.

WebSphere uses *an overqualified update scheme* to test whether the underlying data source has been updated by another transaction since the beginning of the current transaction. With this scheme, the columns marked for update and their

original values are added explicitly through a WHERE clause in the UPDATE statement so that the statement fails if the underlying column values have been changed. As a result, this scheme can provide column-level concurrency control; pessimistic schemes can control concurrency at the row level only.

Optimistic schemes typically perform this type of test only at the end of a transaction. If the underlying columns have not been updated since the beginning of the transaction, pending updates to container-managed persistence fields are committed and the locks are released. If locks cannot be acquired or if some other transaction has updated the columns since the beginning of the current transaction, the transaction is rolled back and all work performed within the transaction is lost.

Pessimistic and optimistic concurrency schemes require different transaction isolation levels. Enterprise beans that participate in the same transaction and require different concurrency control schemes cannot operate on the same underlying data connection unless the connection is able to change its isolation level on an individual-query basis. Some, but not all, JDBC drivers can do this. For those JDBC drivers that cannot, mixing concurrency controls requires the use of multiple connections within a transaction.

Whether or not to use optimistic concurrency depends on the type of transaction. Transactions with a high penalty for failure might be better managed with a pessimistic scheme. For low-penalty transactions, it is often worth the risk of failure to gain efficiency through the use of an optimistic scheme.

15.5.3 Using EJB 2.x access intents

Access intents policies let you define, in a very flexible and powerful way, how relational data will be accessed, if you use BMP or CMP entity beans.

Important: Although you can set access intents on a BMP, you are responsible for reading the access intent metadata from your code and applying the corresponding change to the isolation levels yourself by calling `connection.setTransactionLevel()`. The EJB container has no control over your persistence strategy, and therefore cannot perform the same tasks as it can for CMPs. Access intent data is valid in the WebSphere naming service for the time of the transaction. See the Information Center for more coding examples.

Access intents policies

Seven access intent policies are available. They cover a wide variety of ways to access data. They are summarized in Table 15-5 on page 879.

Table 15-5 Access intent policies

Access Intent Policy	Concurrency control	Used for update	Transaction isolation level	Notes
wsPessimisticRead	pessimistic	No	read committed	Read locks are held for the duration of the transaction. Updates are not permitted; the generated SELECT query does not include FOR UPDATE
wsPessimisticUpdate	pessimistic	Yes	For Oracle, read committed. Otherwise, repeatable read	The generated SELECT FOR UPDATE query grabs locks at the beginning of the transaction
wsPessimisticUpdate-Exclusive	pessimistic	Yes	serializable	SELECT FOR UPDATE is generated; locks are held for the duration of the transaction
wsPessimisticUpdate-NoCollision	pessimistic	No	read committed	The generated SELECT query does not include FOR UPDATE. *No locks are held, but updates are permitted.*
wsPessimisticUpdate-WeakestLockAtLoad (DEFAULT VALUE)	pessimistic	No (Oracle, yes)	For Oracle, read committed, otherwise repeatable read	For Oracle, this is the same as wsPessimisticUpdate. Otherwise, the generated SELECT query does not include FOR UPDATE; locks are escalated by the persistent store at storage time if updates were made.
wsOptimisticRead	optimistic	No	read committed	
wsOptimisticUpdate	optimistic	No	read committed	Generated overqualified-update query forces failure if CMP column values have changed since the beginning of the transaction

Archive

There are two critical questions to ask yourself when using access intents:

- At which point in my transaction do I access data? This is critical in selecting on which method you must set the access intent.
- How do I want to access data? This is critical to selecting the best access intent to apply on the method.

Choosing where to apply the access intent

This is critical because it is at this point that the WebSphere persistence manager decides which access intent to use. To illustrate this point, we will use the following example.

The Consultation session bean obtains the CustomerAccount balance using getBranchAccountBalance, as in Example 15-6:

Example 15-6 Obtaining CustomerAccount balance using getBranchAccountBalance

```
int getCustomerAccountBalance () {
...
custAcct = (CustomerAccountLocal) custAcctHome.findByPrimaryKey(custAcctKey);
return custAcct.getBranchBalance();
}
```

Imagine you have applied AccessWriteIntent1 on the findByPrimarykey() method of the CustomerAccount bean, and AccessReadIntent2 on the getBranchBalance() method. Because the first access to the database in the transaction started by the call to getCustomerAccountBalance() is done by the findByPrimaryKey method, then AccessWriteIntent1 is used for all calls within the transaction. AccessReadIntent2 will be ignored by the persistence manager and therefore useless in this case.

This might be satisfactory or not, depending on what you want to achieve. The critical point here is that you can use the findByPrimaryKey method in read and update transactions. If you use it in an update transaction, you probably want to execute it with, for example, a PessimisticUpdate intent. If you access data only for reading it, this would be more than you need.

There are two main solutions you can adopt for this problem. The simplest one is to have two or more versions of your finder methods, specialized by access intent. You could use the standard findByPrimaryKey in update scenarios and add another finder method such as findByPrimaryKeyForRead and use it in read-only scenarios. You would set the access intent of the findByPrimaryKeyForRead finder, say to wsOptimisticRead. The default access intent (wsPessimisticUpdate-WeakestLockAtLoad) is fine in most cases for the findByPrimaryKey() method, as well as for other finder or non-finder methods.

Important: This solution is also well adapted to BMPs. By having a different findByPrimaryKey method for read and write transactions, you can easily set a different isolation level in the code for each of them. You can also define a different SELECT query, one with a FOR UPDATE clause and one without, then call them from those different methods.

Another solution is to run findByPrimaryKey, or another finder, in its own transaction. Do this by applying a RequiresNew transaction flag on it. Take another look at the previous sample:

1. The getCustomerAccountBalance method starts a new transaction.
2. findByPrimaryKey is called. The current transaction is paused. The findByPrimary method executes within its own transaction and, therefore, own access intent. The call to findByPrimaryKey() returns an *unhydrated* instance, which means it has not been activated nor loaded.
3. The transaction initiated by the session bean resumes.
4. getBalance() is called on the instance returned by findByPrimaryKey. The instance is hydrated and the access intent specified for this method is used. Any other method calls within the transaction will execute with the same access intent.

Note: WebSphere Application Server V6 also includes a new feature (inherited from WebSphere Application Server V5 Enterprise Edition) that provides an extension to access intents called Application Profiles, which handles the problem mentioned above in a powerful way. Application profiles let you externally specify a set of tasks (that is, a flow of calls in your code), and specify which access intent should be used for a specific task. For information about Application Profiles, please refer to the WebSphere Information Center.

Choosing the right access intent

The main rule is: keep it simple.

Start with the default setting (wsPessimisticUpdate-WeakestLockAtLoad), and work from there. Specifying access intents on all your business methods could lead to a configuration, debugging, and maintenance nightmare. Specify access intents on a selected number of methods. Also, choose access intents wisely.

- Access intents can be applied to your business methods, to the findByPrimaryKey() method, as well as the create and remove method. As much as possible, avoid other methods.

- Make sure that no method is configured with more than one access intent policy. Applications that are misconfigured in this way will not be runnable until the configuration errors are fixed.
- For entity beans backed by tables with nullable columns, use optimistic policies with caution. Nullable columns are automatically excluded from overqualified updates at deployment time. This means that at commit time, those columns will not be used in the update statement to check whether the data has changed or not. Therefore, concurrent changes to a nullable field might result in lost updates. Using the Application Server Toolkit, you can set a property on each enterprise bean attribute called *OptimisticPredicate*, as shown in Figure 15-20 on page 883. You can change this property by editing the data mappings of your EJBs. When this property is set, the column, even if it is nullable, will be reflected in the overqualified update statement that is generated in the deployment code to support optimistic policies.

  In V6, support has been added for a new optimistic concurrency control scheme for EJB 2.x CMP entity beans. This new support allows you to add a column for collision detection in your relational database table. This column is reserved to determine if a record has been updated. When using a collision detection column, the overqualified UPDATE statement only needs the collision detection column and the primary key. To manage the collision detection column, provide your own database trigger implementation. Using the collision detection column overcomes the nullable column limitation and the unsupported optimistic concurrency control data types such as BLOBs and CLOBs.

Tip: If you want to check which SQL code is executed for an optimistic update, check the storeUsingOCC method in the <beanname>FunctionSet generated class.

Archive

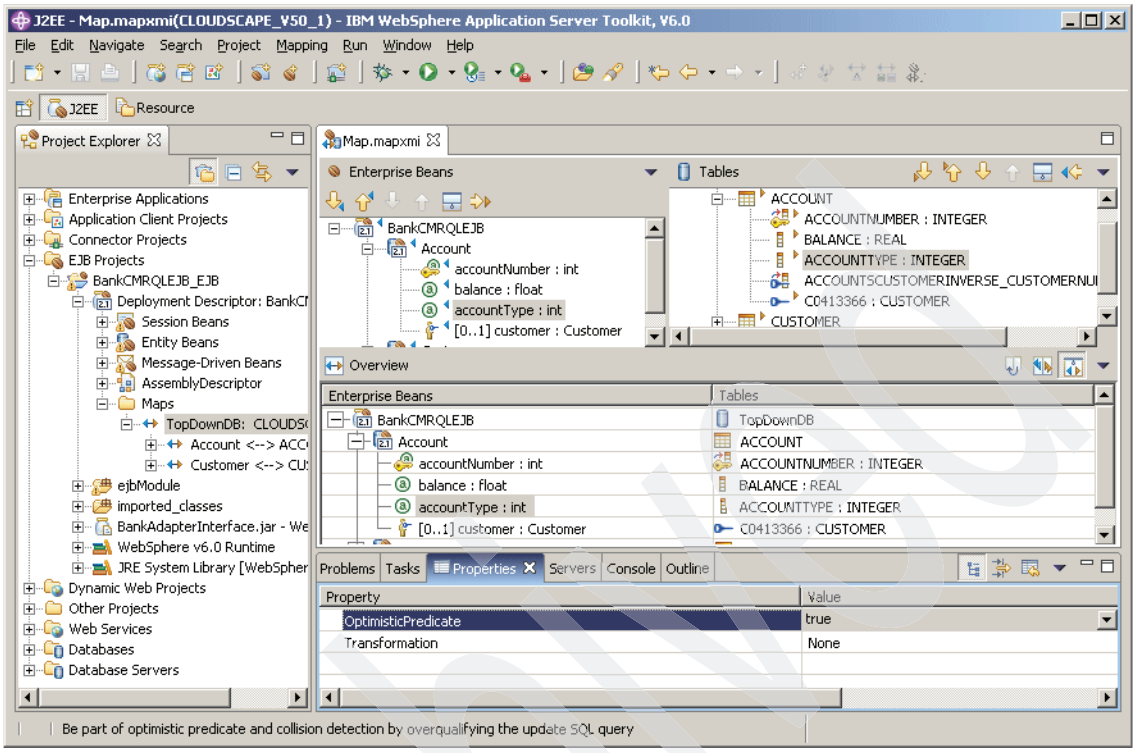


Figure 15-20 Optimistic predicate property

If a bean is loaded for read intent and an update is attempted during that transaction, then the persistence manager will raise an UpdateCannotProceedWithIntegrity exception. In other words, if you call findByPrimaryKeyForRead() and an update is attempted, it will fail.

Important: The behavior described above is true for all access intents except wsPessimisticUpdate-NoCollision. This access intent will not flag updates, even if no locks are held. Our recommendation is that you avoid this access intent in production.

15.5.4 Using read-ahead hints

Read-ahead schemes enable applications to minimize the number of database round trips by retrieving a working set of CMP beans for the transaction within one query. Read-ahead involves activating the requested CMP beans and caching the data for their related beans, which ensures that data is present for the beans that are most likely to be needed next by an application.

A read-ahead hint is a canonical representation of the related beans to be read. It is associated with a finder method for the requested bean type, which must be an EJB 2.x-compliant CMP entity bean. Currently, only findByPrimaryKey methods can have read-ahead hints. Only beans related to the requested beans by a container-managed relationship (CMR), either directly or indirectly through other beans, can be read ahead.

To set Read-ahead hints, do the following:

1. Open the EJB deployment descriptor editor.
2. Select the **Access** tab and scroll down to the WebSphere Extensions section.
3. Click the **Add** button to the right of the Access Intent for Entities 2.x (Method Level) field. See Figure 15-21.

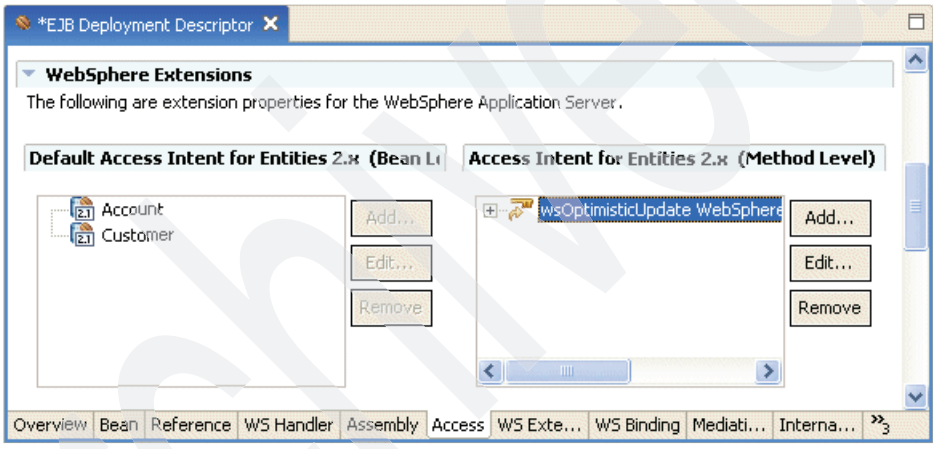


Figure 15-21 Adding a read ahead hint

In the wizard, the Read Ahead Hint check box is enabled only with access intent policies with optimistic concurrency. Read-ahead is limited to optimistic policies because locking persistent data store for all beans represented in the hint would be more likely to cause lock conflicts, and optimistic policies do not obtain locks until immediately before the database operation.

See Figure 15-22 on page 885.

Archived

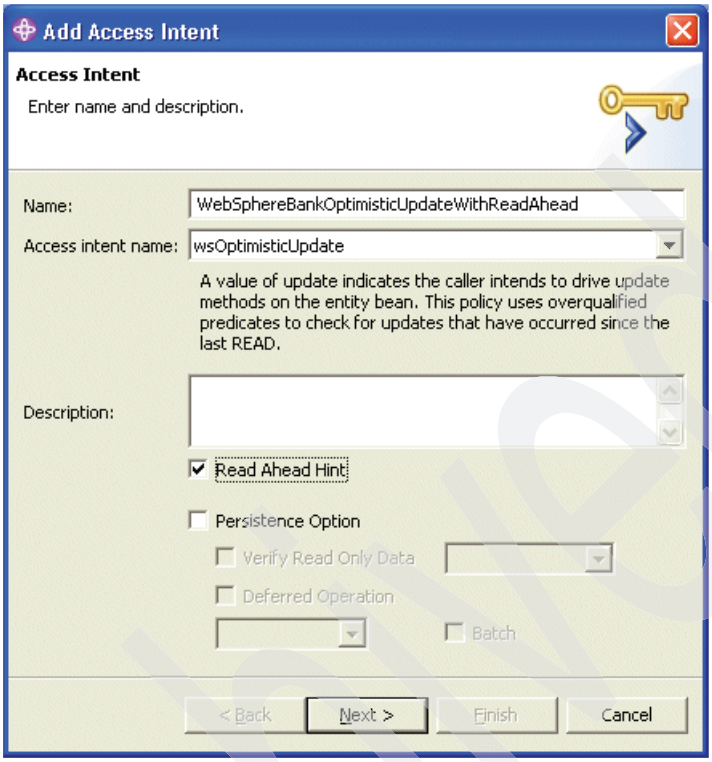


Figure 15-22 Specifying read-ahead hint

4. Follow the panels in the wizard to select the beans and methods, then click **Finish**.

15.5.5 Tracing access intents behavior

You can obtain a very detailed trace of the EJB persistence manager behavior by specifying the following trace specification for an application server:

```
com.ibm.ejs.container.*=all=enabled:com.ibm.ejs.persistence.*=all=enabled:
com.ibm.ws.appprofile.*=all=enabled.
```

15.6 IBM EJB extensions: Inheritance relationships

Support for entity beans inheritance, which is not part of the EJB 1.1 nor EJB 2.x specifications, is also available in the toolkit. Support for enterprise entity beans relationships for EJB 1.1, although not standard, is also available using this tool. Refer to the toolkit documentation for more details.

15.7 IBM Web module extensions

WebSphere Application Server V6 also provides multiple extensions for Web modules. To work with these extensions, open the Web deployment descriptor by double-clicking the Web module in the J2EE Hierarchy view. To see the IBM Web module extensions, select the **Extensions** tab, as in Figure 15-23.

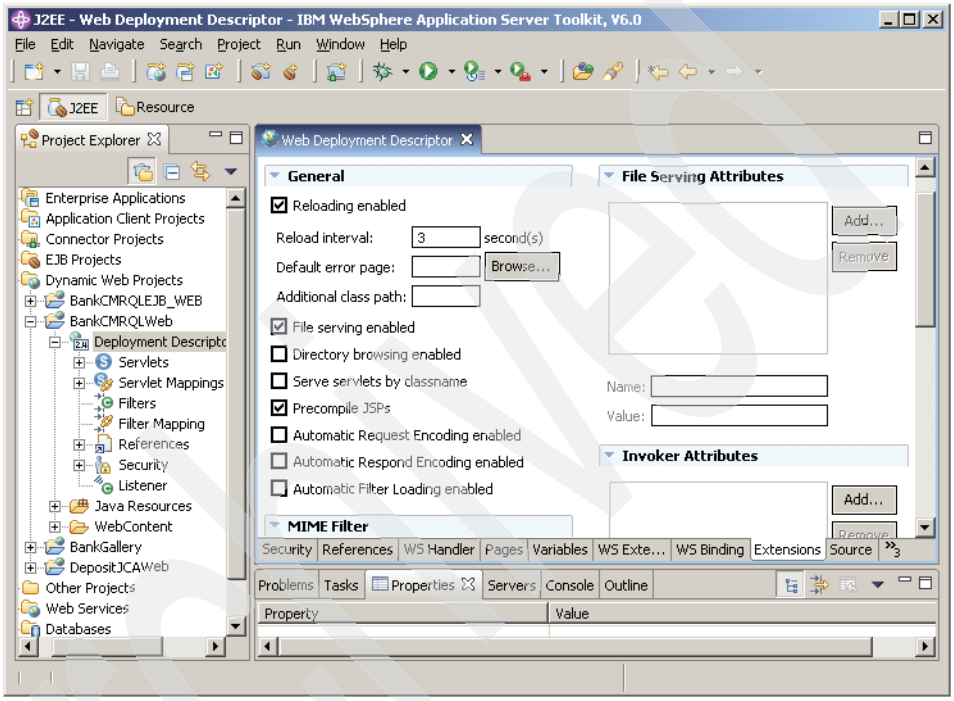


Figure 15-23 Web module extensions

15.7.1 File serving servlet

When dealing with static content (HTML pages, images, style sheets and so on), you can choose to have these resources served by WebSphere, or have them served by the HTTP server itself.

If you want WebSphere to serve the static content of your application, you must enable file servlet, also known as the file serving servlet or file serving enabler. This servlet serves up any resource file packaged in the WAR file. The File serving enabled attribute is set to `true` by default. By changing it to `false`, WebSphere's HTTP plug-in will not send requests for static content to WebSphere, but leave it up to the HTTP server to serve them.

If you want the HTTP server to serve static content, you can experience better performance than using WebSphere in this instance, because the HTTP server is serving the content directly. Moreover, an HTTP server has much more customization options than the file servlet can offer.

However, using the WebSphere file serving servlet has the advantage of keeping the static content organized in a single, deployable unit with the rest of the application. Additionally, this allows you to protect the static pages using WebSphere security.

To enable this option, check the **File serving enabled** box. Enter attributes used by the file serving servlet in the File Serving Attributes section.

15.7.2 Web application auto reload

If you check the **Reloading enabled** option, the class path of the Web application is monitored and all components, JAR or class files, are reloaded whenever a component update is detected. The Web module's class loader is shut down and restarted. The reload interval is the interval between reloads of the Web application. It is set in seconds.

The auto reload feature plays a critical role in hot deployment and dynamic reload of your application.

Important: You must set the Reloading enabled option to `true` for JSP files to be reloaded when they are changed on the file system. Reloading a JSP does not trigger the reload of the Web module, because separate class loaders are used for servlets and JSP.

This option is set to true by default, with the reload interval set to three (3) seconds. In production mode, you might consider making the reload interval much higher.

15.7.3 Serve servlets by class name

The invoker servlet can be used to invoke servlets by class name. Note there is a potential security risk with leaving this option set in production. It should be seen as more of a development-time feature, for quickly testing your servlets.

This option is turned off by default.

15.7.4 Default error page

This page will be invoked to handle errors if no error page has been defined, or if none of the defined error pages matches the current error.

15.7.5 Directory browsing

This boolean defines whether it is possible to browse the directory if no default page has been found.

This option is turned off by default.

15.7.6 JSP attributes

The following options can be set for the JSP compiler:

- keepgenerated

  If this boolean is set to `true`, the source code of the servlet created by compilation of a JSP page is kept on the file system. Otherwise, it is deleted as soon as the servlet code has been compiled, only the .class file is available.

- scratchdir

  This string represents the directory in which servlets code will be generated. If this string is not set, code is created under:

  <was_home>\temp*<hostname>**<application_server_name>**<applicationname>**<web modulename>*.

15.7.7 Automatic HTTP request and response encoding

The Web container no longer automatically sets request and response encodings and response content types. The programmer is expected to set these values using the methods available in the Servlet 2.4 API. If you want the application server to attempt to set these values automatically, check the **Automatic Request Encoding enabled** option in order to have the request encoding value set. Similarly, you can check the **Automatic Response Encoding enabled** in order to have the response encoding and content type set.

The default value of the autoRequestEncoding and autoResponseEncoding extensions is false, which means that both the request and response character encoding is set to the Servlet 2.4 specification default of ISO-8859-1. Different character encodings are possible if the client defines character encoding in the request header, or if the code uses the setCharacterEncoding(String encoding) method.

If the autoRequestEncoding value is set to `true`, and the client did not specify character encoding in the request header, and the code does not include the setCharacterEncoding(String encoding) method, the Web container tries to determine the correct character encoding for the request parameters and data.

The Web container performs each step in the following list until a match is found:

- Looks at the character set (charset) in the Content-Type header
- Attempts to map the server's locale to a character set using defined properties
- Attempts to use the DEFAULT_CLIENT_ENCODING system property, if one is set
- Uses the ISO-8859-1 character encoding as the default

If you set the autoResponseEncoding value to `true` and the client:

- The client did not specify character encoding in the request header.
- The code does not include the setCharacterEncoding(String encoding) method,

The Web container does the following:

- Attempts to determine the response content type and character encoding from information in the request header.
- Uses the ISO-8859-1 character encoding as the default.

15.8 IBM EAR extensions: Sharing session context

In accordance with the servlet 2.4 API specification, the session manager supports session scoping by Web module only. Only servlets in the same Web module can access the data associated with a particular session. WebSphere Application Server provides an option that you can use to extend the scope of the session attributes to an enterprise application. Therefore, you can share session attributes across all the Web modules in an enterprise application.

This option can be set in the toolkit in the enterprise application deployment descriptor.

1. Open the deployment descriptor by double-clicking the enterprise application.
1. Check the **Shared httpsession context** option in the WebSphere Extensions section. See Figure 15-24 on page 890.

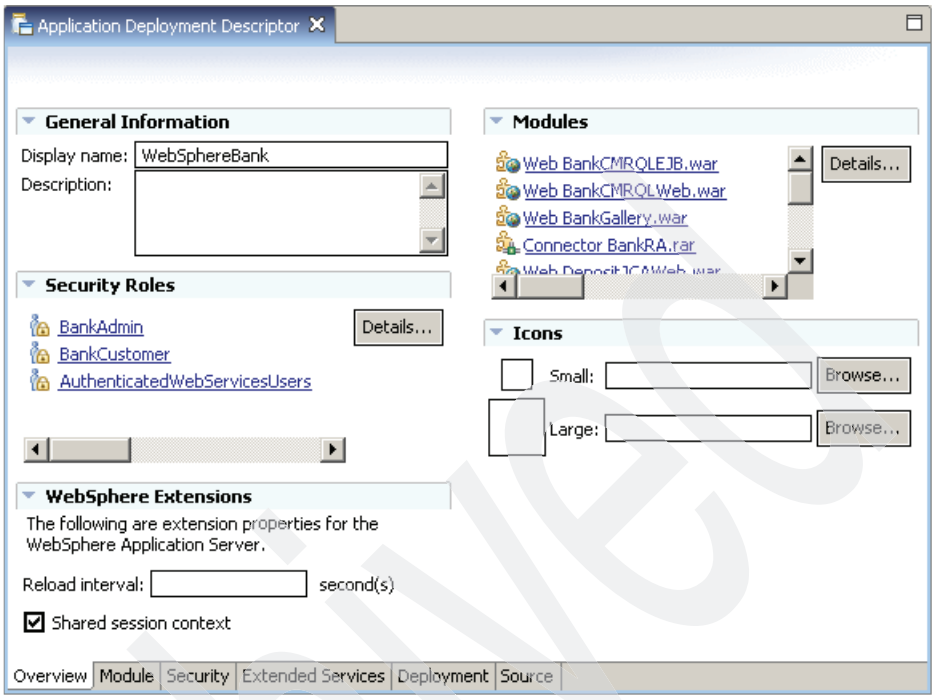


Figure 15-24 EAR deployment descriptor

Important: To use this option, you must install all the Web modules in the enterprise application in the same application server. You cannot use this option when one Web module is installed in one server and the second Web module is installed in a different server.

In such split installations, applications might share session attributes across Web modules using distributed sessions. However, session data integrity is compromised when concurrent access to a session is made in different Web modules.

Sharing HTTP session context also severely restricts the use of some session management features, like time-based writes. For enterprise applications on which this option is enabled, the session management configuration set at the Web module level is ignored. Instead, the session management configuration defined at the enterprise application level is used.

15.9 Exporting WebSphere Bank EAR file

Once you have made all the changes to your application and are ready to deploy, you need to export the EAR file to a location where it can be picked up for deployment by the application server.

To export the WebSphere Bank sample application, do the following:

1. Select **File** → **Export.**
2. Select **EAR file** as export target and click **Next**.
3. Select to export the **WebSphereBank EAR** project and enter a suitable destination for the EAR file, such as `C:\WebSphereBank.ear`. Then click **Finish**. See Figure 15-25.

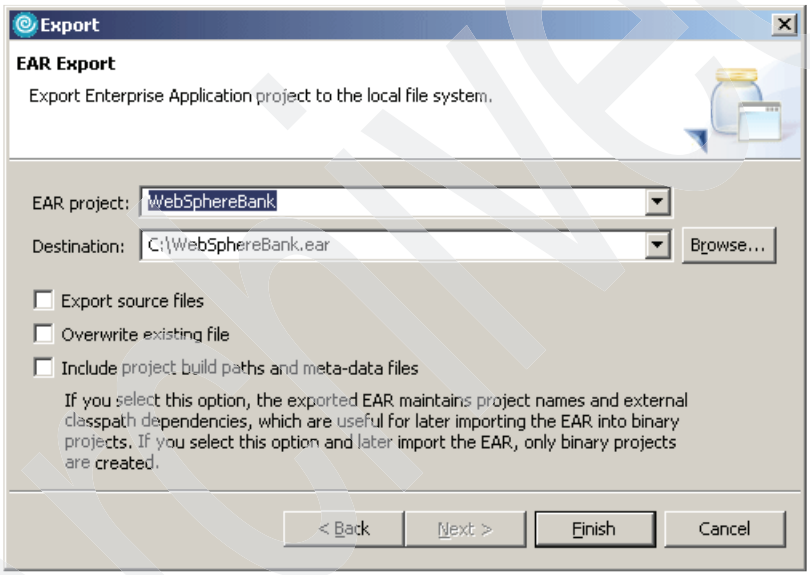


Figure 15-25 Exporting WebSphere Bank EAR file

15.10 WebSphere Enhanced EAR

The Enhanced EAR, introduced in WebSphere Application Server V6, is a normal J2EE EAR file, but with additional configuration information for resources required by J2EE applications. While adding this extra configuration information at packaging time is not mandatory, it can simplify deployment of J2EE applications to WebSphere if the environments where the application is to be deployed is similar.

Table 15-6 shows the resources supported by the Enhanced EAR and the scope in which they are created:

Table 15-6 Scope for resources in WebSphere Enhanced EAR file

Resource	Scope
JDBC providers	Application
Data sources	Application
Substitution variables	Application
Class loader policies	Application
Shared libraries	Server
JAAS authentication aliases	Cell
Virtual hosts	Cell

When an Enhanced EAR is deployed to a WebSphere Application Server V6 server, WebSphere can automatically configure the resources specified in the Enhanced EAR. This reduces the number of configuration steps required to set up the WebSphere environment to host the application.

When an Enhanced EAR is uninstalled, the resources that are defined at the application level scope are removed as well. However, resources defined at a scope other than application level are not removed because they might be in use by other applications.

Note: Resources created at Application level scope are limited in visibility to only that application. Also, these resources are not visible from the WebSphere administrative console. To modify these resources once the application is deployed in WebSphere, you have to use wsadmin. And example of this is changing the connection pool settings for a data source.

Worth noticing is that in WebSphere Application Server V6 the Enhanced EAR file does not support configuration information about, for example, JMS queues. These resources will still have to be configured using the WebSphere administrative console or wsadmin before the application is deployed.

15.10.1 Configuring a WebSphere Enhanced EAR

The supplemental information in an Enhanced EAR is modified by using the WebSphere Enhanced EAR editor, the Deployment tab of the application deployment descriptor in the Application Server Toolkit.

Note: Before adding or removing J2EE modules using the Module page in the Application Deployment Descriptor editor, do the following:

1. Click the **Deployment** tab to activate the functions in the deployment page.
2. Add your modules to the Module page.

Complete this task for each Application Deployment Descriptor editor session that you want to add or remove modules from the Module page.

To access the Enhanced EAR deployment options, do the following:

1. In the J2EE Project Explorer view expand **Enterprise Applications**, then the application.
2. Double-click **Deployment Descriptor** and select the **Deployment** tab. This opens up the Enhanced EAR editor as shown Figure 15-26.

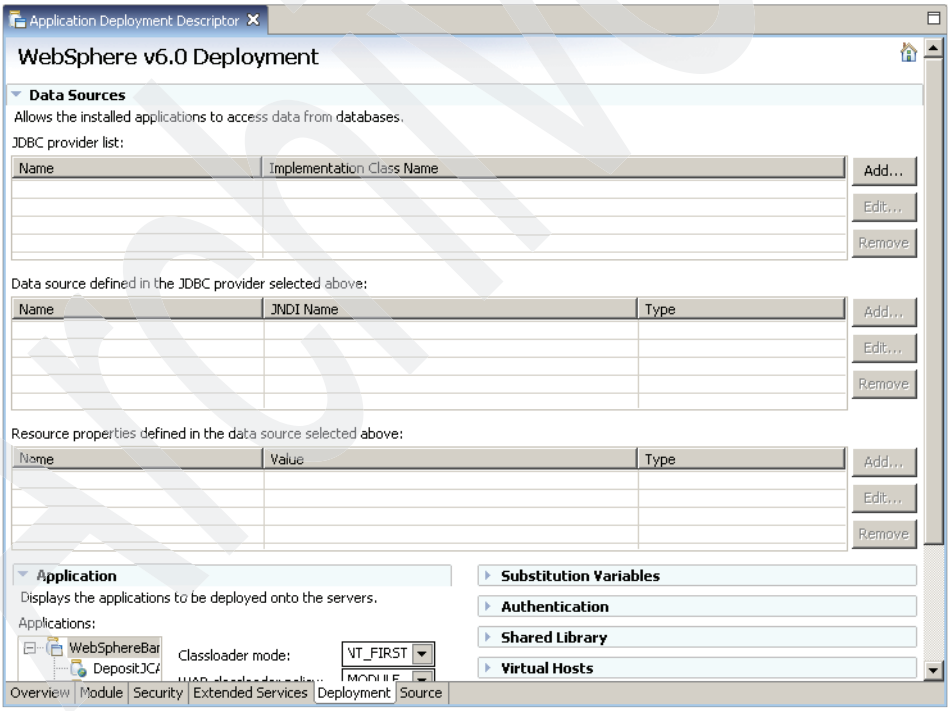


Figure 15-26 WebSphere Enhanced EAR editor

In the Application section in Figure 15-27, you can see the class loader policies and class loader mode configured for each of the containing module. WebSphere Bank runs fine with the default policies and modes so they do not need to be changed.

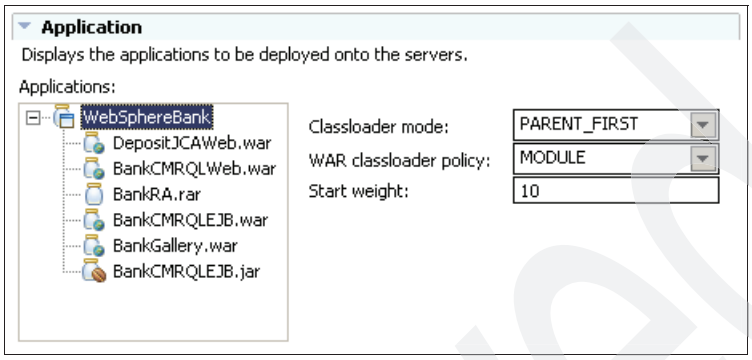


Figure 15-27 Configuring class loader mode and class loader policies

In the Enhanced EAR editor you also configure resources such as JDBC providers, datasources, class loader policies, JAAS authentication aliases, virtual hosts and so forth.

To configure the WebSphere Bank application, we need to add the following:

- JAAS authentication alias
- JDBC provider for DB2
- Data source for DB2 database

Just to show the editor, we will also configure a new virtual host for a domain called www.webspherebank.com.

Configuring a JAAS authentication alias

To configure the the JAAS authentication alias, do the following:

1. In the Deployment tab, expand the **Authentication** section.
2. Click the **Add** button. See Figure 15-28 on page 895.

Archived

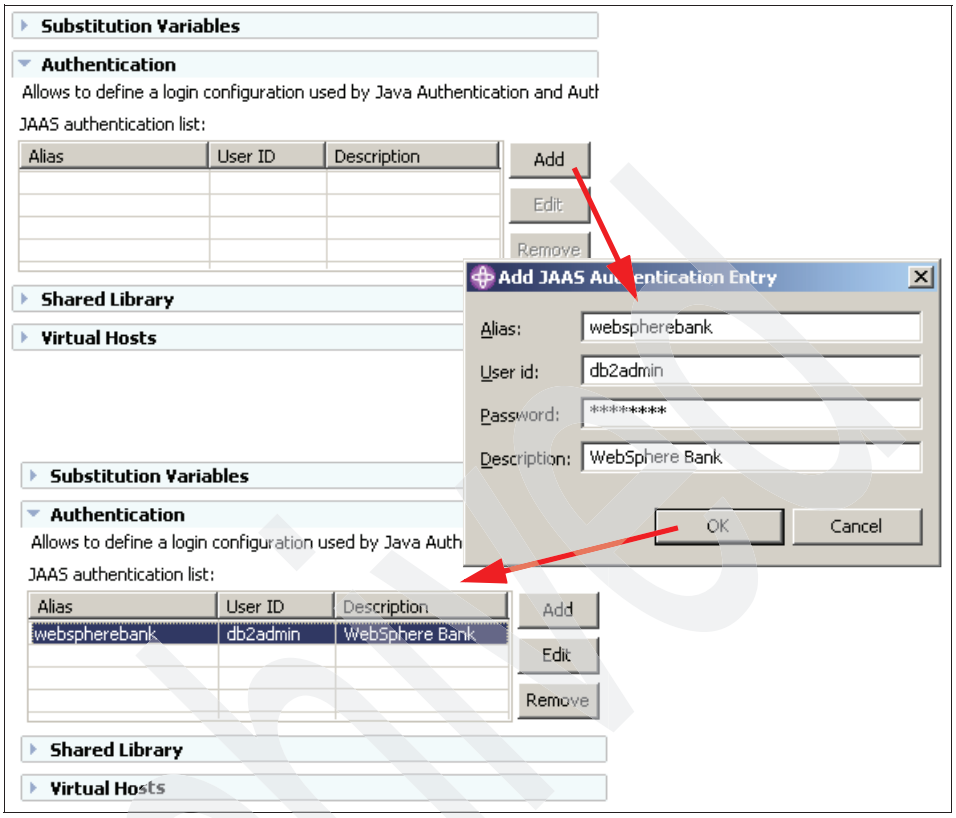


Figure 15-28 Configuring JAAS authentication alias for WebSphere Bank

3. In the dialog box that displays, enter:
   - `websphere bank` as the alias
   - A user ID with access to the BANK database
   - The password for the user ID.
   - `WebSphere Bank` as the description
4. Click **OK**.

Configuring a DB2 JDBC provider

To configure the DB2 JDBC provider, do the following:

1. Click the **Add** button next to the JDBC provider list in the **Data Sources** section. See Figure 15-29 on page 896.

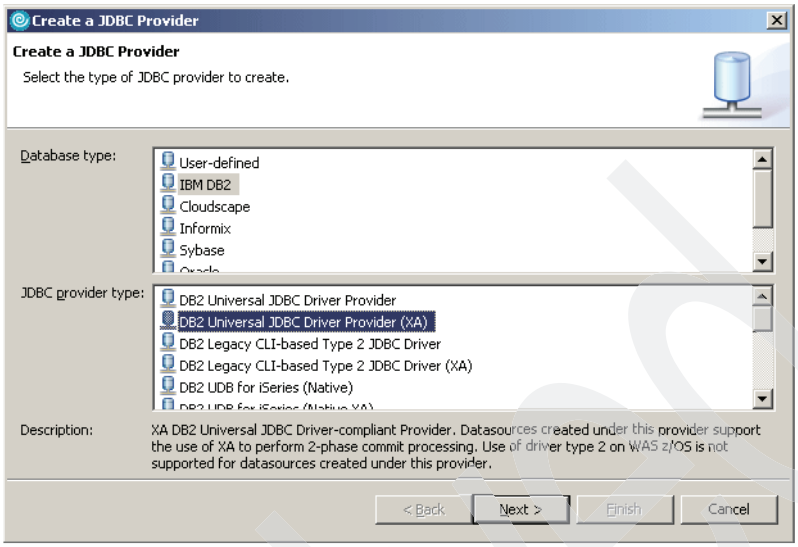


Figure 15-29 Creating a DB2 JDBC Provider

2. In the dialog box:
   - Select **IBM DB2** as the Database type
   - Select **DB2 Universal JDBC Driver Provider (XA)** as the JDBC provider type

   Click **Next**.
3. In the next dialog box, enter a name for the JDBC provider (for administration purposes only) and leave the other properties as the default values. See Figure 15-30 on page 897.

Archived

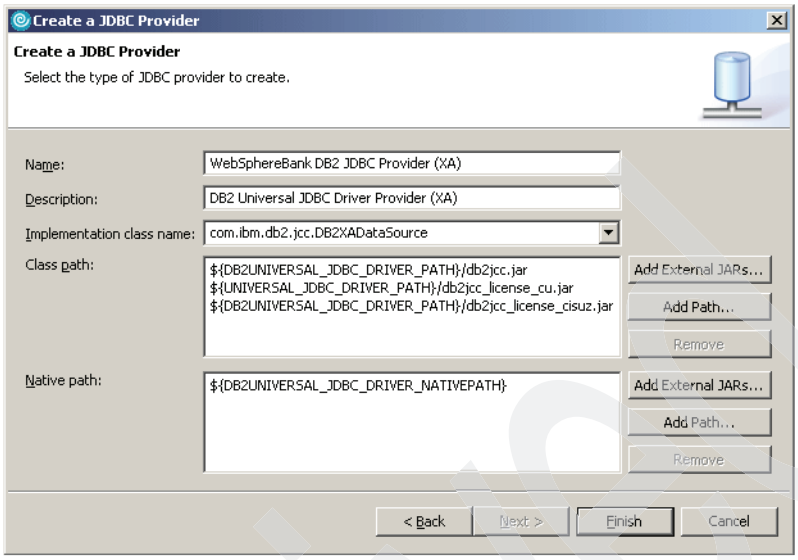


Figure 15-30 Creating a DB2 JDBC provider

Click **Finish**.

4. Select the **WebSphereBank DB2 JDBC Provider (XA)** you just created and click the **Add** button next to the Data source list, as in Figure 15-31.

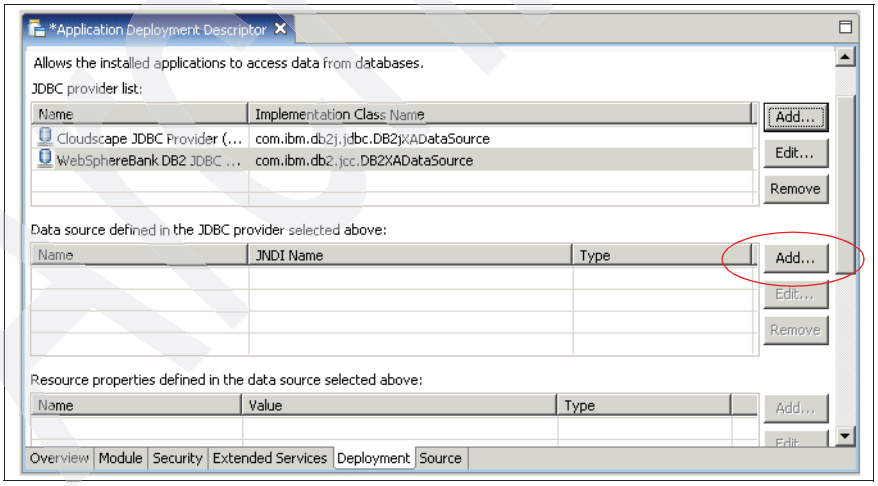


Figure 15-31 Add a data source

5. In the Create a Data Source dialog box, select **DB2 Universal JDBC Driver Provider (XA)** as the JDBC provider type and **Version 5.0 data source** as the data source type, as in Figure 15-32.

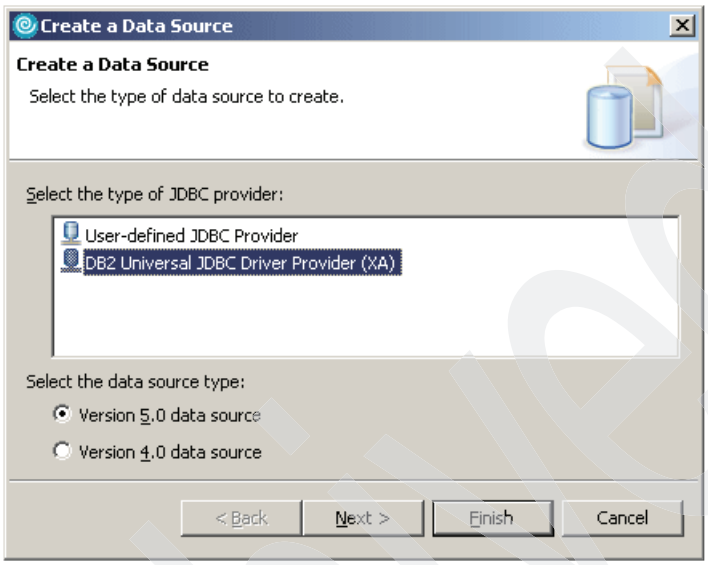


Figure 15-32 Creating a DB2 data source

Click **Next.**

6. In the dialog box displayed enter the appropriate values for the DB2 data source. See Figure 15-33 on page 899.

Archived

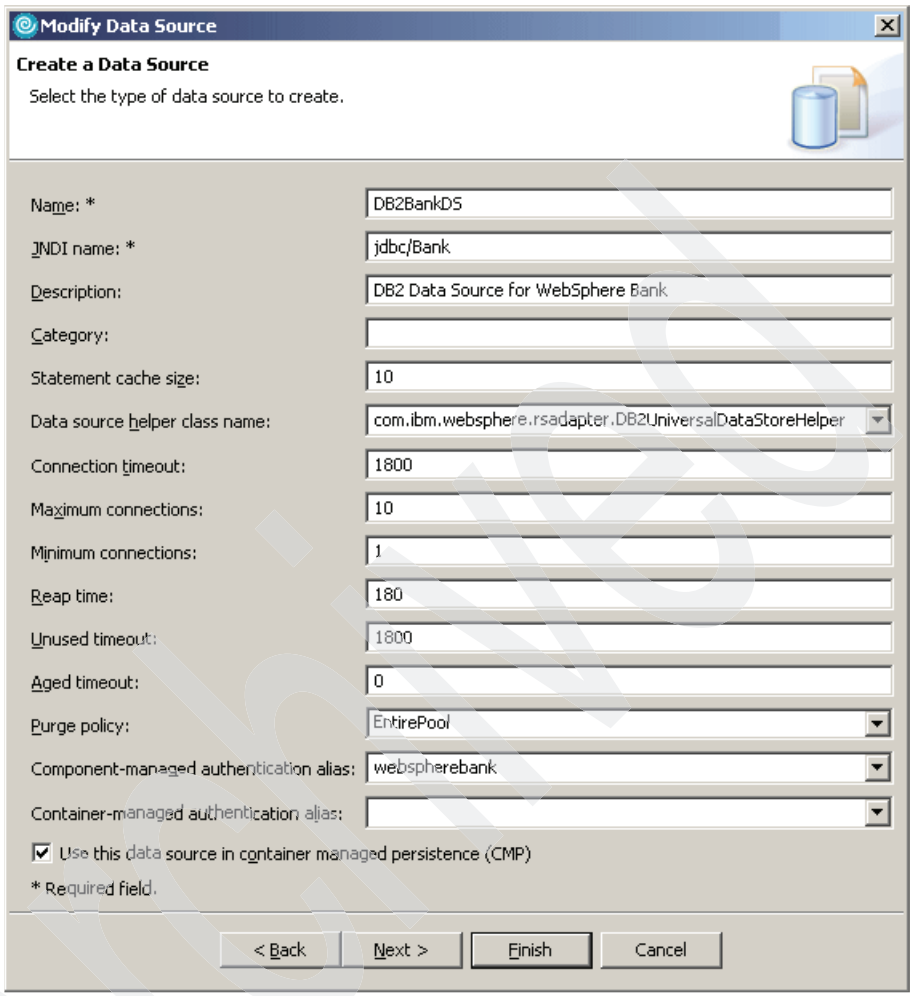


Figure 15-33 Creating a DB2 data source

- Enter `DB2BankDS` as the name.
- Enter `jdbc/Bank` as the JNDI name.
- Enter `DB2 Data Source for WebSphere Bank` as the description.
- Select **websphererebank** as the Component-managed authentication alias.
- Check **Use this data source in container manager persistence (CMP)**.

Click **Next**.

7. In the Create Resource Properties dialog box, select **databaseName** and enter `BANK` as the value. See Figure 15-34 on page 900.

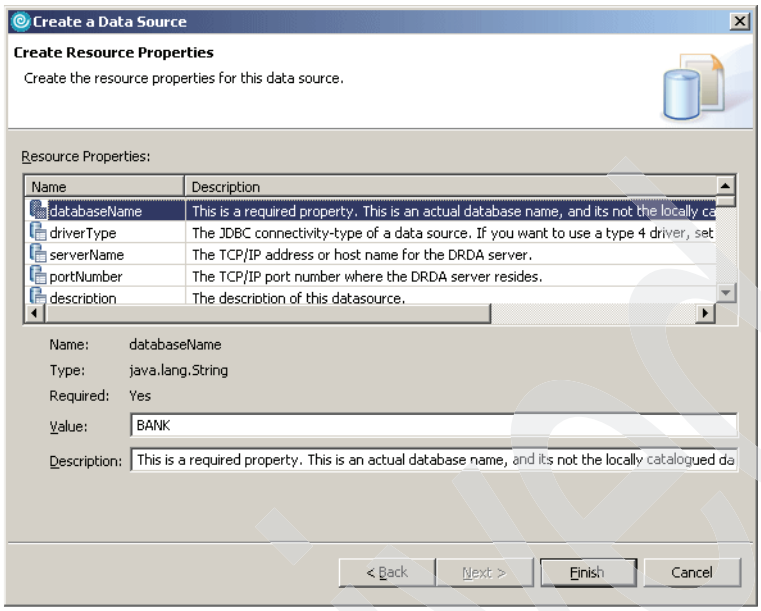


Figure 15-34 Setting database properties for DB2 data source

Click **Finish**.

When you are finished, your data source configuration should look like Figure 15-35 on page 901.

Archive

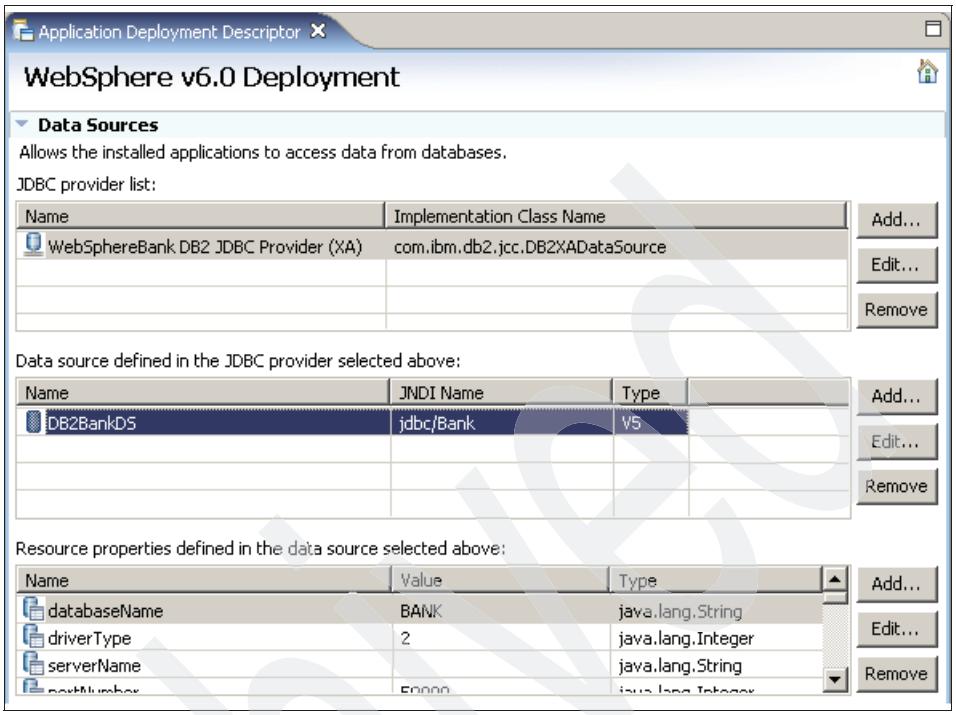


Figure 15-35 DB2 data source configured

Adding a virtual host

To configure a virtual host, do the following:

1. Expand the **Virtual Hosts** section of the Deployment tab and click the **Add** button next to the Virtual host name list.
2. In the Add Host Name Entry dialog box, enter `webspherebank_host` and click **OK**.
3. Click the **Add** button next to the Host aliases list.
4. In the Add Host Alias Entry dialog box, enter `www.webspherebank.com` for the host name and `80` for the port number. Click **OK**.

   Repeat the procedure to add port numbers `9085` and `443` as well. We will use these port numbers when we deploy the application later.

   The flow is shown in Figure 15-36 on page 902.

Archived

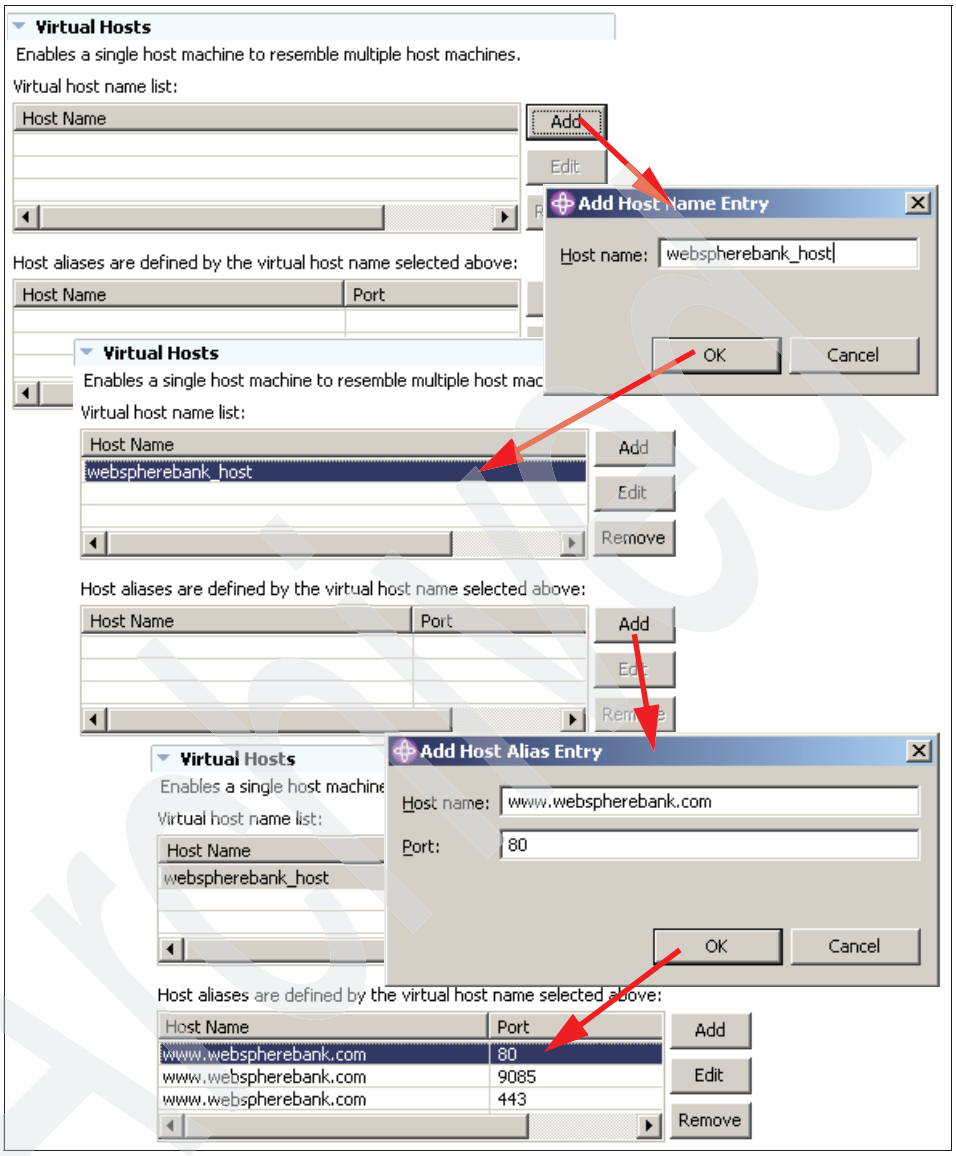


Figure 15-36 Configuring the virtual host for WebSphere Bank

5. When you are finished, press **Ctrl-S** to save the deployment descriptor editor.

Setting default virtual host for Web modules

Just because we have configured a new virtual host, webspherebank_host, in the Enhanced EAR file does not mean that all our Web modules automatically use it.

The default virtual host for a Web module created in the Application Server Toolkit or Rational Application Developer is default_host, and so is the case also for the Web modules of the WebSphere Bank application. This setting can be found in the ibm-web-bdn.xmi file in the /WEB-INF directory of each Web module.

To configure the Web modules to default to the webspherebank_host instead, do the following:

1. Expand **Dynamic Web Project** in the Project Explorer view.
2. Expand the **BankCMRQLEJB_WEB** project and double-click **Deployment Descriptor.**
3. Scroll to the bottom of the Overview page and replace default_host with `webspherebank_host` as shown in Figure 15-37.



Figure 15-37 Setting default virtual host for a Web module

4. Save the deployment descriptor by pressing **Ctrl-S** and then **close** it.
5. Repeat these steps for the other Web modules (BankCMRQLWeb, BankGallery and DepositJCAWeb) as well. Remember to save each Web module's deployment descriptor.

Examining the WebSphere Enhanced EAR file

The information about the resources configured is stored in the ibmconfig subdirectory of the EAR file's META-INF directory. Expanding this directory reveals the well-known directory structure for a WebSphere cell configuration, as seen in Figure 15-38 on page 904. You can also see the scope level where each resource is configured.

Archived

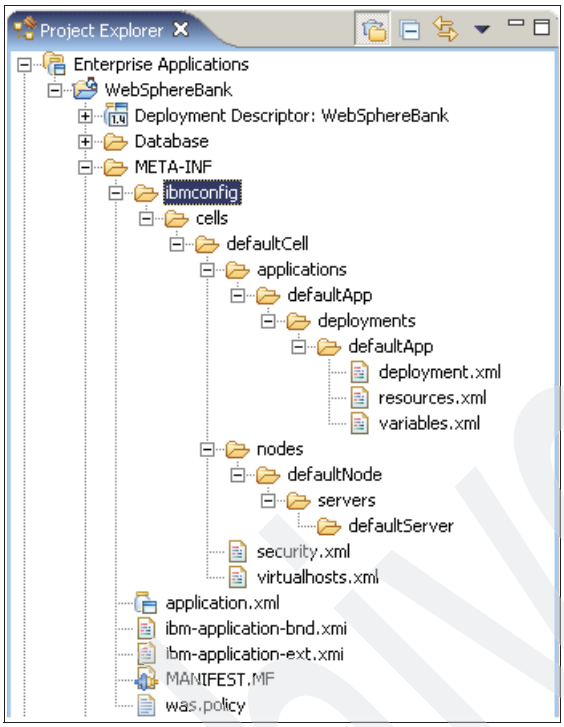


Figure 15-38 Enhanced EAR file contents

After you have re-packaged the application into an Enhanced EAR, export it as explained in "Exporting WebSphere Bank EAR file" on page 891.

At deployment time, WebSphere Application Server V6 uses this information to automatically create the resources.

15.11 Packaging recommendations

Here are some basic rules to consider when packaging an enterprise application:

- The EJB JAR modules and Web WAR modules comprising an application should be packaged together in the same EAR module.
- When a Web module accesses an EJB module, you should not package the EJB interfaces and stubs in the WAR modules. Thanks to the class loading architecture, EJB stubs and interfaces are visible by default to WAR modules.
- Utility classes used by a single Web module should be placed within its WEB-INF/lib folder.

Archived

- Utility classes used by multiple modules within an application should be placed at the root of the EAR file. This is what is done for the BankAdapterInterface.jar and WsaEJBDeployUtility.jar, which are used both by servlets and EJBs.
- Utility classes used by multiple applications can be placed on a directory referenced through a shared library definition.

See 14.4, “Learning class loaders by example” on page 835 for more details on how WebSphere finds and loads classes.

Archived

Archived

16

Deploying applications

In Chapter 15, “Packaging applications” on page 847, we discussed how to use the Application Server Toolkit to perform common tasks for packaging an application. In this chapter we show you how to deploy the application. We take you through setting up the environment for the application, and then deploying the application itself. Next, we explain how to deploy the client part of the application.

The deployment tasks in this chapter can also be automated using command-line tools as explained Chapter 6, “Administration with scripting” on page 267. You might also want to take a look at Chapter 17, “WebSphere Rapid Deployment” on page 957 to see if using the new automatic application installation mode of the WebSphere Rapid Deployment features is appropriate for your environment.

WebSphere Application Server V6 supports the J2EE Deployment API Specification (JSR-88), which defines standard APIs to enable deployment of J2EE applications and standalone modules to J2EE application servers. For more information about how to use this API, see the WebSphere Information Center by searching for JSR-88 and browse to the section discussing Installing J2EE modules with JSR-88.

© Copyright IBM Corp. 2005. All rights reserved.

16.1 Preparing the environment

In this chapter we show you how to set up a fairly complete environment for the WebSphere Bank application and deploy the EAR file. Many times, however, you might not need or want to customize the environment as far as we do in this chapter. Some steps are optional. If all you want to do is deploy your application quickly, using the WebSphere defaults for directory names, log files, and so forth, skip to 16.3, “Deploying the application” on page 930.

The steps in this section are performed typically by the application deployer. To deploy the WebSphere Bank application, do the following:

1. Create the DB2 database for WebSphere Bank. This step is required.
2. Create an environment variable for WebSphere Bank server. This step is optional.
3. Create an application server to host the application. This step is optional.
4. Customize the IBM HTTP Server configuration. This step is optional.
5. Define a JDBC provider, data source and authentication alias. This step is required if you are not using an Enhanced EAR.
6. Define virtual hosts. This step is optional and not required if you are using an Enhanced EAR.
7. Configure WebSphere messaging. This step is required.

If the application to be deployed is a WebSphere Enhanced EAR file, the resources configured in the Enhanced EAR file are created automatically when the application is deployed. If the application to be deployed is a WebSphere Enhanced EAR file, the resources configured in the Enhanced EAR file are created automatically when the application is deployed.

16.1.1 Creating the WebSphere Bank DB2 database

The WebSphere samples by default use Cloudscape as the database. However, in this chapter we will configure WebSphere Bank to use a DB2 UDB 8.2 database instead.

To set up the DB2 database, make sure you have DB2 installed and running. Then run the following commands:

1. Select **Start** →**Programs** → **IBM DB2** → **Command Line Tools** → **Command Window**.
2. Create the database using the commands in Example 16-1 on page 909.

Example 16-1 Creating the DB2 database

```
db2 create database bank
db2 connect to bank user <user_id<> using <password>
db2 -tvf Table.ddl
db2 connect reset
```

The Table.ddl file is located in the `BankCMRQLEJB_EJB\ejbModule\META-INF\backends\DB2EXPRESS_V82_1` directory of your Application Server Toolkit workspace. See “Creating a new database mapping and schema” on page 867.

16.1.2 Creating a WEBSPHEREBANK_ROOT environment variable

It is recommended that you use WebSphere environment variables, rather than hard-coded paths when deploying an application. In the following steps, we assume you have declared a WEBSPHEREBANK_ROOT variable. You will use it when specifying, for example, the JVM log’s location.

Be certain you declare this variable at the right scope. For example, if you define this variable at the application server scope, it will only be known at that level. As long as you work with the WebSphere Application Server Base or Express editions, this is fine. But if you later decide to use the Network Deployment edition and you create a cluster of application servers, the WEBSPHEREBANK_ROOT variable will need to be defined at the node level if all members of a cluster are on the same node, and at cell level if the cluster spans multiple nodes. Use the steps in 5.1.10, “Using variables” on page 179 to create a WEBSPHEREBANK_ROOT variable with a value of `C:\apps\WebSphereBank`.

There are several ways to organize WebSphere applications. Some companies prefer to create a directory for each application, as we do in our example, such as C:\apps\<*application_name*>, and keep all resources and directories required by the application in subdirectories under this directory. This strategy works well when deploying only one application per application server, again as we do in our example, because the application server’s log files could then all be changed to point to c:\apps\<*application_name*>\logs.

Other companies prefer to organize resources by resource type, and so create directories such as c:\apps\logs\<*application_name.log*>, c:\apps\properties\<*application_name.properties*> etc. And some companies prefer to stick with the vendor defaults as far as possible. For WebSphere, that means that the applications are installed in the <*was_home*>/installedApps directory and the logs files are written to the <*was_home*>/logs/<*server_name*> directory. Which option you choose is a matter of personal preferences and corporate guidelines.

Note: Make sure you create the target directory you specify for the WEBSPHEREBANK_ROOT variable before proceeding. If the directory is not created, the application server will not start.

16.1.3 Creating the WebSphere Bank application server

In a distributed server environment, you have the option of using a single application server, or creating multiple application servers or clusters.

The advantages of deploying multiple applications to a single application server is that it consumes less resources. There is no overhead for any extra application server processes. Another benefit is that applications can make in-process calls to each other. For example, servlets in one EAR file could access Local interfaces of EJBs in another EAR file.

One alternative to using a single application server is to deploy each application to its own server. The advantages of deploying only one application on an application server, is that it gives you greater control over the environment. The JVM heap sizes and environment variables are set at application server level, so all applications running in an application server share the JVM memory given to the application server and they would all see the same environment variables. Running each application in its own application server could also make it easier to perform problem determination. For example, if an application runs amok and consumes a lot of CPU, you could see which application it is by looking at the process ID of the application server.

In our example, we create a unique application server on which to run the WebSphere Bank sample application.

Note: For a full discussion of application server properties, see 5.4, “Working with application servers” on page 190.

To create an application server, do the following:

1. Select **Servers** → **Application Servers**.
2. Click the **New** button and provide the information as shown in Figure 16-1 on page 911.

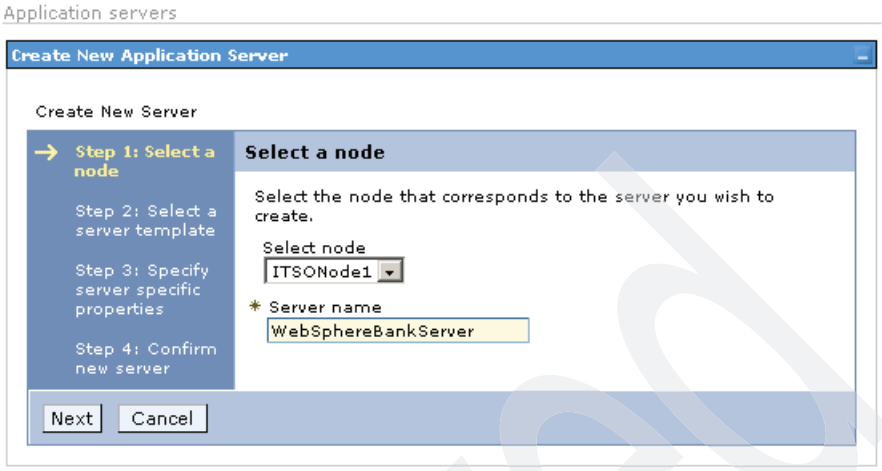


Figure 16-1 Creating the WebSphere Bank application server

- **Node**

  Select the node on which the application server will be created.

- **Server name**

  Enter the application server name, such as `WebSphereBankServer`.

Click **Next**.

3. In Step 2, select which server template to use as the base for this new application server. If you have not created any templates, your only choice is to select the WebSphere **default**. Otherwise, select the server template you want to use and click **Next**.
4. In step 3 you can select if you want WebSphere to generate a unique set of port numbers for this application server. This ensures the ports defined for this server does not conflict with another server currently configured on this node. Check the **Generate Unique Http Ports** box and click **Next**.
5. On the Summary page, click **Finish**.

Changing the working directory

The next thing we want to do is to change the working directory for the application server process. This directory is the relative root for searching files. For example, if you do a File.open("foo.gif"), foo.gif must be present in the working directory. This directory will be created by WebSphere if it does not exist. We recommend that you create a specific working directory for each application server.

Archived

1. Select the server, **WebSphereBankServer**, you just created.
2. Expand the **Java and Process Management** in the Server Infrastructure section and select **Process Definition**.
3. Scroll down the page and change the working directory from `${USER_INSTALL_ROOT}` to `${WEBSPHEREBANK_ROOT}/workingDir`.
4. Click **OK**.

Note: The working directory will not be created if you use a composed path, such as C:/apps/WebSphereBank/workingDir. If you want to use such a path, create it before starting the application server, or the startup sequence fails.

Changing the logging and tracing options

Next, we want to customize the logging and tracing properties for the new application server. These properties are discussed in detail in Chapter 9, "Problem determination" on page 417. There are several ways to access the logging and tracing properties for an application server:

- Select **Troubleshooting** → **Logs and Trace** in the navigation bar, then select a server.
- Select **Servers** → **Application Servers**, select a server, and then select **Logging and Tracing** from the Troubleshooting section.
- Select **Servers** → **Application Servers**, select a server, select **Process definition** from the Java and Process Management section. Select **Logging and Tracing** from the Additional Properties section.

Because we have just finished updating the application server process definition, we will take the third navigation path to customize the location of the JVM logs, the diagnostic trace logs, and the process logs.

1. Select **Logging and Tracing.**
2. Select **JVM Logs**.

   This allows you to change the JVM standard output and error file properties. Both are rotating files. You can choose to save the current file and create a new one, either when it reaches a certain size, or at a specific moment during the day. You can also choose to disable the output of calls to System.out.print() or System.err.print().

   We recommend that you specify a new file name, using an environment variable to specify it, such as:

   ```
   ${WEBSPHEREBANK_ROOT}/logs/SystemOut.log
   ${WEBSPHEREBANK_ROOT}/logs/SystemErr.log
   ```

   Click **OK**.

3. Select **Diagnostic Trace.**

   Each component of the WebSphere Application Server is enabled for tracing with the JRas interface. This trace can be changed dynamically while the process is running from the Runtime tab, or added to the application server definition from the Configuration tab. As shown in Figure 16-2, the trace output can be either directed to memory or to a rotating trace file.

   Change the trace output file name so the trace is stored in a specific location for the server using the WEBSPHEREBANK_ROOT variable and select the **Log Analyzer** format.

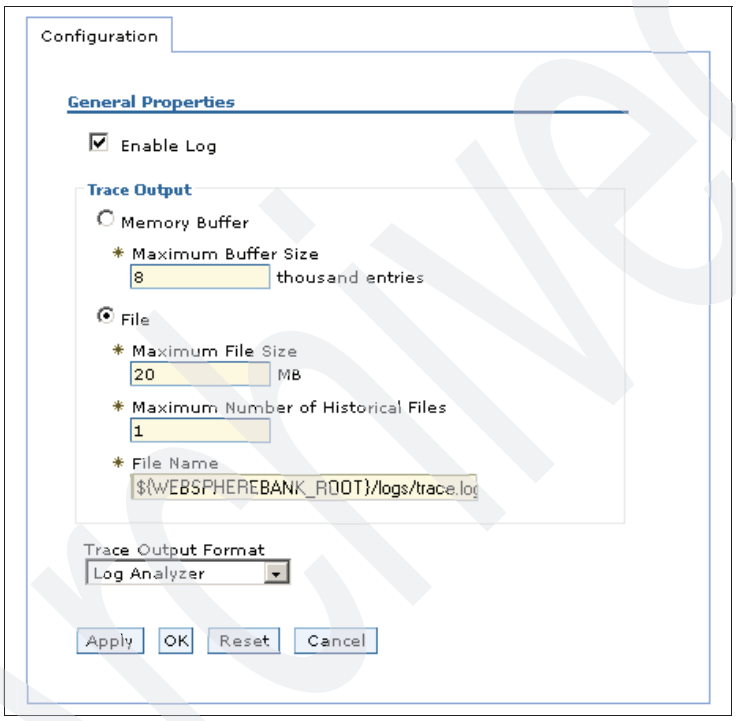


Figure 16-2 Specifying diagnostic trace service options

   Click **OK**.

4. Select the **Process Logs**.

   Messages written by native code (JNI) to standard out and standard error streams are redirected by WebSphere to process logs, usually called native_stdout.log and native_stderr.log. Change the native process logs to:

```
${WEBSPHEREBANK_ROOT}/logs/native_stdout.log
${WEBSPHEREBANK_ROOT}/logs/native_stderr.log
```

   Click **OK**.

Archived

5. All log files produced by the application server are now redirected to the `${WEBSPHEREBANK_ROOT}/logs` directory. **Save** the configuration.

Note: The rest of this example assumes a default HTTP port of 9085 for the Web container. Before proceeding, check the application server you created to determine the port you should use:

1. Select **Servers** → **Application Servers**.
2. Select the **WebSphereBankServer**.
3. Select **Ports** in the **Communications** section.
4. Scroll down the page and note the port listed for **WC_defaulthost**.

16.1.4 Defining the WebSphere Bank virtual host

Enhanced EAR file users: If you are using an Enhanced EAR file, the virtual host can be defined at packaging time. See "Adding a virtual host" on page 901.

Web modules need to be bound to a specific virtual host. For our sample, we chose to bind the WebSphere Bank Web module to a specific virtual host called websphere bank_host. This virtual host has the following host aliases:

- www.websphere bank.com:80
- www.websphere bank.com:9085
- www.websphere bank.com:443 (for SSL access)

Any request starting with `<websphere bank_host_alias>/WebSphereBank`, such as `http://www.websphere bank.com:9085/WebSphereBank`, is served by the WebSphere Bank application.

Tip: You can restrict the list of hosts used to access the WebSphere Bank Web application by removing hosts from the virtual host definition.

Imagine you want to prevent users from directly accessing the WebSphere Bank application from the WebSphere internal HTTP server when they invoke `http://www.websphere bank.com:9085/WebSphereBank.` In other words, you want to force all requests to go through the Web server plug-in. You can achieve this by removing `www.websphere bank.com:9085` from the virtual host aliases list. The application server will still listen on that port, and that port will still be used in the plug-in configuration file. However, the application server will prevent any direct call to this port.

To create the webspherebank_host virtual host, do the following:

1. Select the **Environment** → **Virtual Hosts** entry in the navigation pane.
2. Click **New**.
3. Enter the virtual host name, `webspherebank_host`.
4. Click **Apply**.
5. Select **Host Aliases** in the Additional Properties section.
6. Add the three aliases shown in Figure 16-3 by clicking **New**, entering the values, and clicking **OK**.

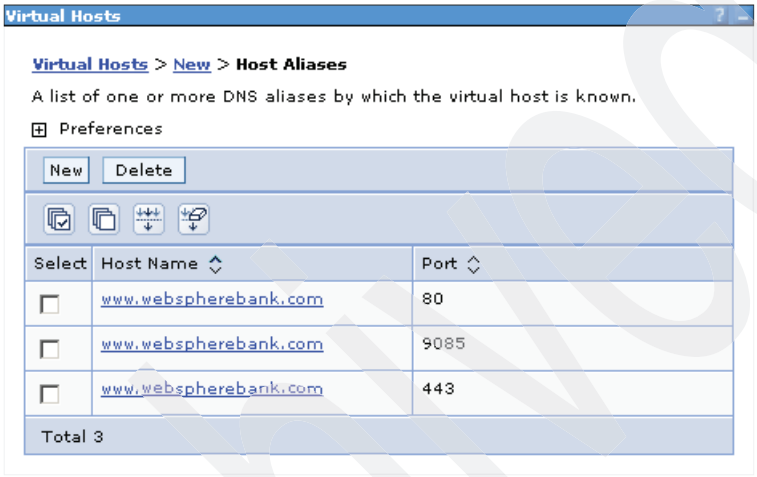


Figure 16-3 WebSphere Bank virtual host aliases

7. Click **OK**.
8. Save the configuration.

16.1.5 Creating the virtual host for IBM HTTP Server and Apache

Now that we have defined a webspherebank_host virtual host, we need to configure the Web server to serve the host aliases in the virtual host. The steps below are valid for both the IBM HTTP Server V6 and Apache 2.0.

Configuring virtual hosting

Note: It is not necessary to create a virtual host in httpd.conf. It is required only if you want to customize the configuration, for example, by separating the logs for each virtual host. This is not normally done.

Creating virtual hosts is done using the VirtualHost directive, as in Example 16-2.

Example 16-2 Using VirtualHost

```
<VirtualHost www.webspherebank.com:80>
   ServerAdmin webmaster@webspherebank.com
   ServerName www.webspherebank.com
   DocumentRoot "C:\WebSphere\IBMHTTPServer\htdocs\webspherebank"
   ErrorLog logs/webspherebank_error.log
   TransferLog logs/webspherebank_access.log
</VirtualHost>
```

:If you want to have multiple virtual hosts for the same IP address, you must use the NameVirtualHost directive. See Example 16-3.

Example 16-3 Using the NameVirtualHost and VirtualHost directives

```
NameVirtualHost 9.42.171.190:80

<VirtualHost itso_server:80>
   ServerAdmin webmaster@itso_server.com
   ServerName itso_server
   DocumentRoot "C:\WebSphere\IBMHTTPServer\htdocs\itso_server"
   ErrorLog logs/itso_server_error.log
   TransferLog logs/itso_server_access.log
</VirtualHost>

<VirtualHost www.webspherebank.com:80>
   ServerAdmin webmaster@webspherebank.com
   ServerName www.webspherebank.com
   DocumentRoot "C:\WebSphere\IBMHTTPServer\htdocs\webspherebank"
   ErrorLog logs/webspherebank_error.log
   TransferLog logs/webspherebank_access.log
</VirtualHost>
```

The www.webspherebank.com and the itso_server hosts have the same IP address, 9.42.171.190. We have set this by inserting the following line in the machine hosts file, located in %windir%\system32\drivers\etc or in /etc on UNIX systems):

```
9.42.171.190 www.webspherebank.com itso_server
```

In a real-life environment, this would probably be achieved by creating aliases at the DNS level. In any event, you must be able to ping the host you have defined, using commands such as **`ping www.webspherebank.com`**.

Archived

As you can see in Example 16-3, each virtual host has a different document root. Make sure that the directory you specify exists before you start the HTTP server. We recommend that you place an index.html file at the document root stating which virtual host is being called. This lets you easily test which virtual host is being used.

You must restart the IBM HTTP Server to apply these changes. If you are running a Windows system, we recommend that you try to start the server by running **apache.exe** from the command line rather than from the Services window. This allows you to spot error messages thrown at server startup.

If your virtual hosts are correctly configured, invoking `http://www.webspherebank.com` or `http://itso_server` returns different HTML pages.

Configuring listening ports

If you want to use a different port rather than the default port 80, use the `Listen` directive to tell the Web server to listen to requests on that specific port:

1. Make a backup of the <*ihs_home*>\conf\httpd.conf file.
1. Start your favorite editor and edit this file.
2. Search for the line containing the `# Listen` string, and edit it as follows to make it listen on, for example, port 90:

   ```
   Listen 90
   ```

3. Save the httpd.conf file.
4. Restart the HTTP server.

16.1.6 Creating a DB2 JDBC provider and data source

Enhanced EAR file users: If you are using an Enhanced EAR file, the JDBC provider, data source, and J2C authentication entry can be defined at packaging time. See “Configuring a DB2 JDBC provider” on page 895.

The WebSphere Bank sample application uses a relational database, via entity beans, to store account information. To access this database, define a data source, which you then associate with the entity beans. The WebSphere Bank sample application, just like the other sample applications, is configured for Cloudscape by default. In Chapter 15, “Packaging applications” on page 847,

however, we also added a backend and EJB deployed code for the WebSphere Bank application to run against a DB2 database. We will now create the DB2 JDBC provider, datasource and JAAS authentication alias required to run against DB2.

For detailed information about JDBC providers and data sources, refer to 7.2, “JDBC resources” on page 321.

Configuring environment variables for DB2 JDBC driver

For the DB2 Universal JDBC Provider to find its classes the `DB2UNIVERSAL_JDBC_DRIVER_PATH` and `DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH` environment variables must be setup. To set up these variables, do the following:

1. Select **Environment** → **WebSphere Variables**.
2. Locate and click the **DB2UNIVERSAL_JDBC_DRIVER_PATH** entry.
3. In the value field, enter the path to where the DB2 JDBC driver is located. For example, for DB2, the location is likely to be:

   ```
   C:\Program Files\IBM\SQLLIB\java
   ```

   See Figure 16-4.

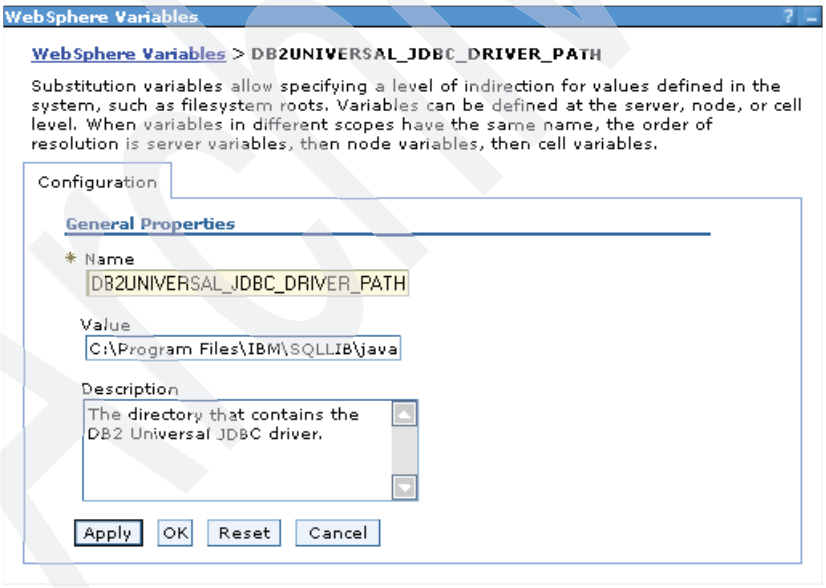


Figure 16-4 Configuring DB2 Driver Path

Click **OK**.

4. Repeat the process for the `DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH` variable. For DB2, it should use the same path, `C:\Program Files\IBM\SQLLIB\java`.

Configuring J2C authentication data

The user ID and password required to access the database are specified in a J2C authentication data entry.

1. Select **Security** → **Global Security.** Expand the JAAS Configuration section and select **J2C Authentication Data**.
2. Click **New**, and specify the following information to create the authentication data. Once completed, the authentication information should be similar to Figure 16-5.
   - **Alias**

     Enter the name of the security information alias, such as `websphere bank`.
   - **User ID**

     Enter a user ID with the proper authority to access the database.
   - **Password**

     Enter the password for the user ID.

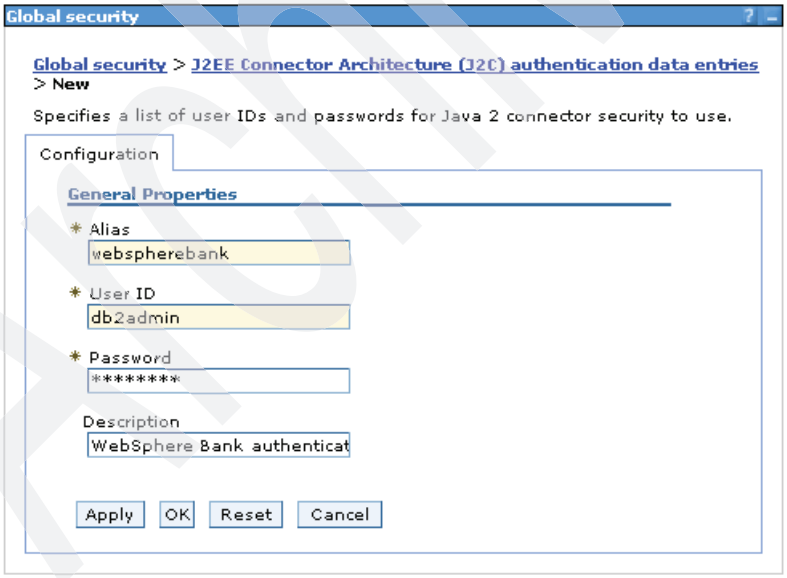


Figure 16-5 Creating WebSphere Bank JAAS authentication alias

3. Click **OK**.

Archived

Creating the WebSphere Bank JDBC provider

The following steps take you through the creation of a JDBC provider targeting a DB2 database. To create a JDBC provider from the administrative console, do the following:

1. Expand the **Resources** entry and select the **JDBC Providers** entry.
2. Select the scope of this resource. In a standalone server environment, it is sufficient to create the data source at the server level. Otherwise, define it at the node or cell level. A rationale for this is to be able to share the definition across multiple servers in a cluster. To change this, select the server you are deploying to in the scopes list and click **Apply**.
3. Click the **New** button.
4. In the Configuration dialog box, select the general properties for the JDBC provider as shown in Figure 16-6.

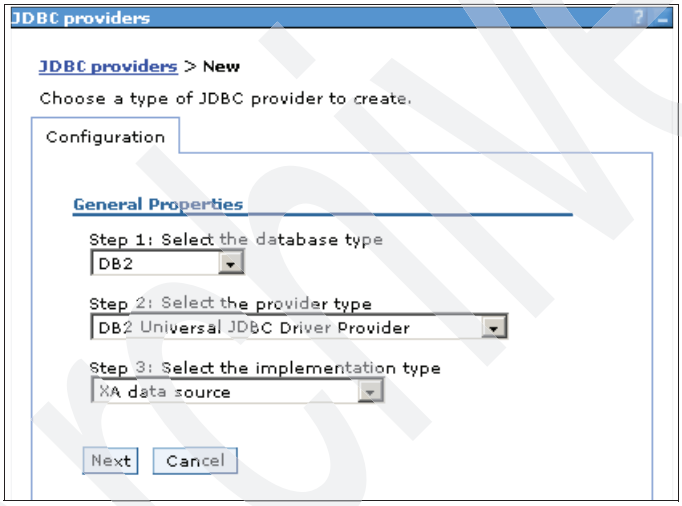


Figure 16-6 Creating a DB2 JDBC provider

   - Database type: **DB2**
   - Provider type: **DB2 Universal JDBC Driver Provider**
   - Implementation type: **XA data source**
5. Specify the following information for the provider, as shown in Figure 16-7.

Note: We used the DB2 XA-capable JDBC Driver for the WebSphere Bank sample. If your application does not require two-phase commit capabilities, use the regular driver. If using an XA-capable driver, it is a best practice to indicate that it is an XA-capable driver by including XA in its name, such as MyJDBCDriverXA.

Archived

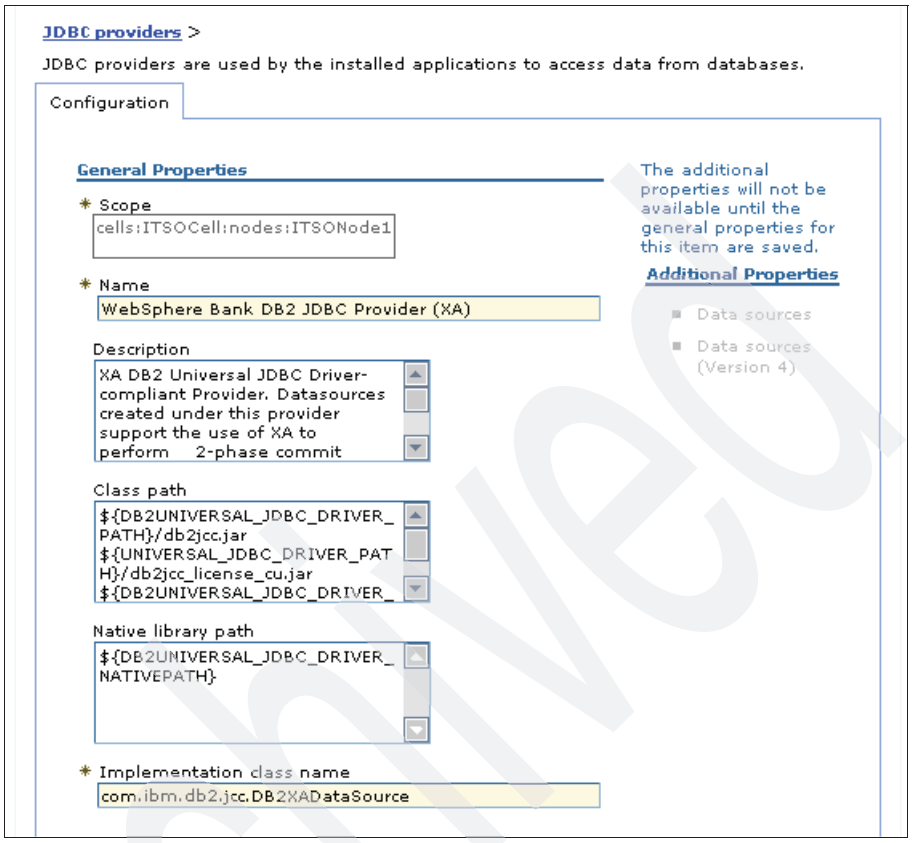


Figure 16-7 WebSphere Bank JDBC provider properties

– **Name**

Enter the JDBC provider name, such as `WebSphere Bank DB2 JDBC Provider (XA)`.

– **Classpath**

Enter the path to the JAR/ZIP file that contains the JDBC provider code, for example, ${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc.jar. We encourage you to use variables rather than hard-code the path to the JDBC drivers.

– **Native library path**

Enter the path to any native libraries required by the JDBC driver.

Archive

– **Implementation class**

Enter the Java class that provides the JDBC service, such as `com.ibm.db2.jcc.DB2ConnectionPoolDataSource`.

The classpath, native library path and the implementation class fields are all automatically completed for known JDBC providers.

6. Click **OK**.

Creating the WebSphere Bank data source

The next step is to create the data source for the WebSphere Bank DB2 database. To create a data source, do the following:

1. Select **Resources** → **JDBC Providers**.
2. Select the **WebSphere Bank DB2 JDBC Provider (XA)** and select **Data Sources** under Additional Properties.
3. Click **New** to add the new data source. See Figure 16-8 on page 923.

Archived

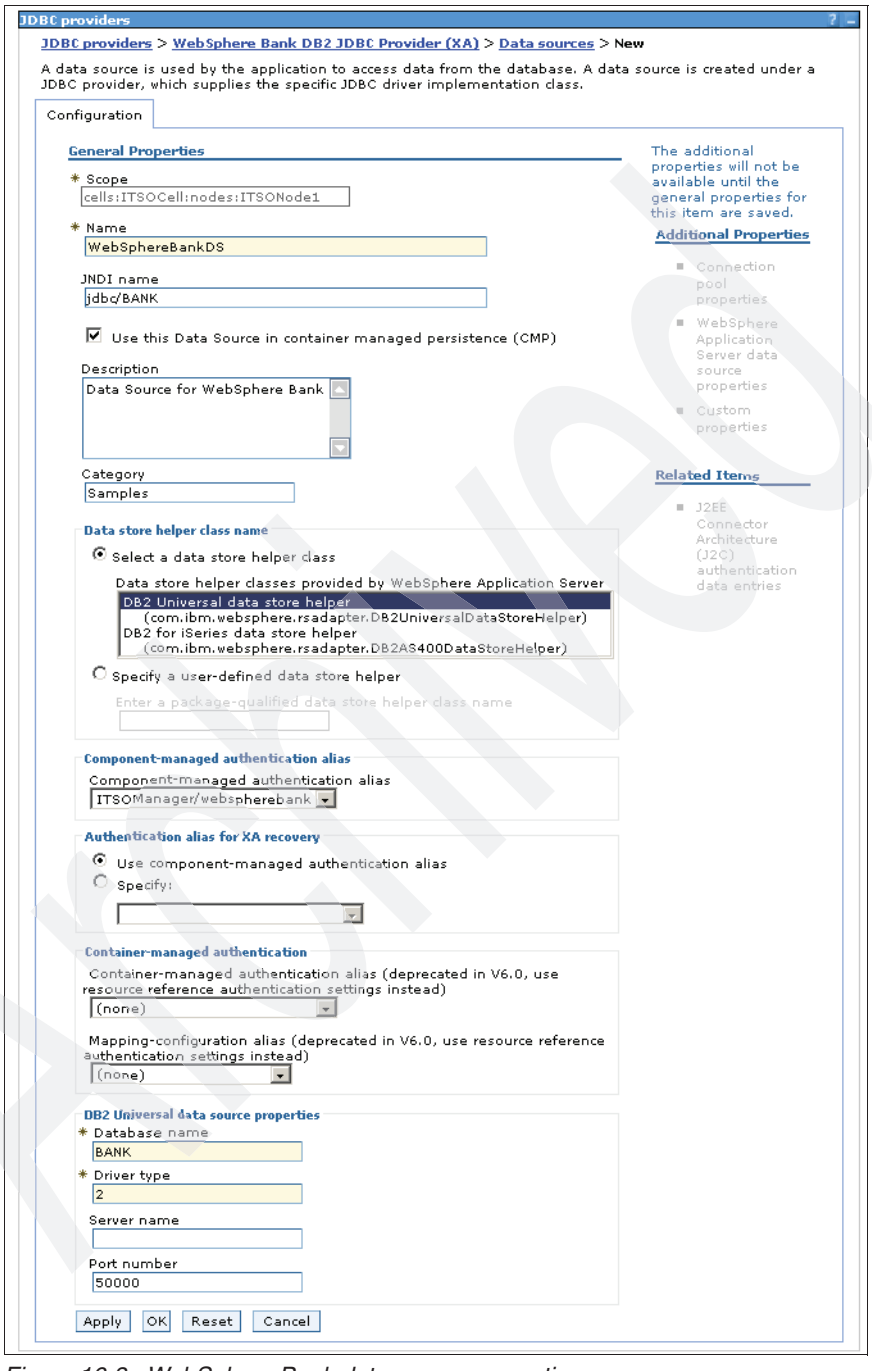


Figure 16-8 WebSphere Bank data source properties

– **Name**

  Enter the data source name, which must be unique in the administrative domain or cell. It is recommended that you use a value indicating the name of the database this data source is targeting, such as "WebSphereBankDS".

– **JNDI name**

  Enter the name by which applications access this data source. If not specified, the JNDI name defaults to the data source name prefixed with jdbc/. For the WebSphere Bank, set this field to jdbc/BANK. This value can be changed at any time after the data source has been created.

– **Use this Data Source in container-managed persistence (CMP)**

  Check this option to indicate that the WebSphere Bank data source is used for persisting the application entity beans.

– **Datasource helper class name**

  Select the name of the class used by the Relational Resource Adapter at runtime to implement the functionality of a specific database driver. Select **DB2 Universal data store helper**.

– **Component-managed authentication alias**

  Enter the J2C alias used when connecting to this data source by selecting the authentication alias created previously, <cell name>/webspherebank.

  Note: The component-managed authentication alias is used when the res-auth tag in the deployment descriptor is set to Application and the application itself does not specify a user ID when obtaining a connection (that is, when datasource.getConnection() is called). The container-managed application alias (used when res-auth tag is set to Container) is deprecated in WebSphere Application Server V6.

– **DB2 Universal data source properties**

  Enter the database name, driver type, server name and port number required to access the database. In our environment the database is called BANK and is located on the same physical machine as our application server so we use the Type 2 driver.

  For DB2, specify at least the database name and the driver type. For Oracle, provide the driver type (thin/oci8), and the database URL, or server name and listening port.

4. Click **OK**.
5. Save the configuration.

6. Test the connection by selecting the data source and clicking the **Test Connection** button.

16.1.7 Configuring the messaging resources

The messaging function of the WebSphere Bank consists of the following resources that need to be defined. The details of defining each are covered in the messaging chapters of this book.

- J2C authentication alias entry

  Create a J2C authentication alias entry with the properties in Table 16-1.

Table 16-1 J2C authentication alias properties

Property	Value
Alias	<*cell*>/samples
User ID	samples
Password	s1amples

- Service integration bus

  The service integration bus is the default messaging provider. By default, there is no bus defined. Define a bus with the the following property in Table 16-2.

Table 16-2 Service integration bus property

Property	Value
Name	SamplesBus

  Let the rest of the properties default.

  To see an example of how to create the service integration bus, see 11.8.2, “Creating a bus” on page 658.

  Next, add the application server to the bus as a member. See 11.8.3, “Adding a bus member using a default data store” on page 660 for information and an example.

  In this case, we are using the default data store in Cloudscape, so no action is needed to create the database.

- Service integration bus queue

  Define a queue to the bus with the following properties in Table 16-3 on page 926.

Archived

Table 16-3 Service integration bus queus properties

Property	Value
Destination type	Queue
Identifier	BankJSConnFactory
Bus member	<*node*>:<*server*>

See 11.8.5, “Creating a queue destination” on page 666 for information and an example of how to do this.

- JMS provider:

  WebSphere Bank uses the default messaging provider. This provider is preconfigured at all scope levels. For information about viewing and working with the default messaging provider, see 10.5.1, “Managing the default messaging JMS provider” on page 514.

- JMS connection factory

  A JMS connection factory is required to create a connection to the service integration bus. The connection factory has the following properties in Table 16-4.

Table 16-4 JMS connection factory properties

Property	Value
Scope	Node
Connection factory name	BankJMSConnFactory
JNDI name	jms/BankJMSConnFactory
Bus name	SamplesBus
Component managed authentication alias	<*cell*>/samples

Let the rest of the properties default.

For information about JMS connection factory properties see “JMS connection factory properties” on page 527. To see an example of how to create this connection factory, see “JMS connection factory configuration” on page 534.

- JMS queue

  A JMS destination is used to address a message to a specific destination on the JMS provider. For this application, the destination is a JMS queue and will reference a queue on the bus. Enter the properties in Table 16-5.

Archived

Table 16-5 JMS queue properties

Property	Value
Scope	Node
Connection factory name	BankJMSQueue
JNDI name	jms/BankJMSQueue
Bus queue name	BankJSQueue

Let the rest of the properties default.

For information about JMS queue properties see “JMS destination properties” on page 535. To see an example of how to create this JMS queue, see “JMS queue configuration” on page 539.

- Activation specification

  The activation specification is used to associated a destination with a message-driven bean. In this case, a message-driven bean associated with this activation specification would be invoked when a message arrives on the BankJSQueue destination on the service integration bus, which is mapped to jms/BankJMSQueue. Enter the following properties in Table 16-6.

Table 16-6 Activation specification properties

Property	Value
Scope	Node
Name	BankActivationSpec
JNDI name	eis/BankActivationSpec
Destination type	queue
Destination queue name	jms/BankJMSQueue
Bus queue name	SamplesBus

Let the rest of the properties default.

For information about activation specification properties see “JMS activation specification properties” on page 545. To see an example of how to create this activation specification, see “JMS activation specification configuration” on page 549.

Archived

16.2 Generating deployment code

At some point, you will need to generate the deployment code for the Enterprise JavaBeans. You can do this in Rational Application Developer, in the Application Server Toolkit, from the command line, or at deployment time using the install panels in the WebSphere administrative console. The deployed code should match the version of the runtime target. If you plan to deploy an EAR file which was developed in WebSphere Studio version 5, we recommend you regenerate the deployed code to match WebSphere Application Server V6.

For information about how to generate the deployed code using the Application Server Toolkit, see “Creating a new database mapping and schema” on page 867.

16.2.1 Using EJBDeploy command line tool

You can generate the EJB deployed code using the EJBDeploy command-line tool. The syntax of the EJBDeploy command is shown in Example 16-4.

Example 16-4 EJBDeploy syntax

```
EJBDeploy (v6.0, dms0444.09)

Syntax: EJBDeploy inputEar workingDirectory outputEar [options]
Options:
  -cp "jar1;jar2"   List of jar filenames required on classpath
  -codegen          Only generate the deployment code, do not run RMIC or Javac
  -bindear:options  Bind references within the EAR
  -dbschema schema  The name of the schema to create
  -dbvendor DBTYPE  Set the database vendor type, to one of:
        DB2UDB_V81      DB2UDB_V82
        CLOUDSCAPE_V5
        DB2UDBOS390_V7  DB2UDBOS390_V8
        DB2UDBISERIES   DB2UDBISERIES_V52 DB2UDBISERIES_V53
        INFORMIX_V73    INFORMIX_V93      INFORMIX_V94
        MSSQLSERVER_V7  MSSQLSERVER_2000
        ORACLE_V9I      ORACLE_V10G       SYBASE_V1200    SYBASE_V1250
  -debug            Compile the code with java debug information
  -keep             Do not delete the contents of the working directory
  -ignoreErrors     Do not halt for compilation or validation errors
  -quiet            Only display errors, suppress informational messages
  -nowarn           Disable warning and informational messages
  -noinform         Disable informational messages
  -rmic "options"   Set additional options to use for RMIC
  -35               Use the WebSphere 3.5 top-down mapping rules
  -40               Use the WebSphere 4.0 top-down mapping rules
  -target           Set the server target, to one of:
```

```
                        WAS510   WAS502   WAS501   WAS500
  -trace            Trace progress of the deploy tool
  -sqlj             Use SQLJ instead of JDBC
  -OCCColumn        Add a column for collision detection for WebSphere 6.0 or
later release
```

For a complete description of the EJBDeploy command and its parameters, see the Information Center. Search for *ejbdeploy*.

Example 16-5 shows a sample EJBDeploy run using the WebSphereBank EAR file.

Example 16-5 EJBDeploy sample run

```
C:\WebSphere\AppServer\bin>ejbdeploy c:\WebSphereBankEnhanced.ear
c:\temp c:\WebSphereBankEnhanced_Deployed.ear -dbvendor DB2UDB_V82

Starting workbench.
Creating the project.
Building: /BankCMRQLEJB_EJB
Deploying jar BankCMRQLEJB_EJB
Validating
[*Information]
imported_classes/com/ibm/websphere/samples/bank/ejb/Customer.class(Method:
getAccountsList(), Class: com.ibm.websphere.samples.bank.ejb.Customer):
CHKJ2500I: java.util.Collection must be serializable at runtime (EJB 2.0:
10.6.9).
[*Warning] ejbModule/META-INF/ejb-jar.xml(BankCMRQLEJB): CHKJ2874W: Migrate
this EJB module's default datasource binding to a default CMP Connection
Factory binding.
Generating deployment code
Refreshing: /BankCMRQLEJB_EJB/ejbModule.
Building: /BankCMRQLEJB_EJB
Invoking RMIC.
Generating DDL
Generating DDL
Building: /WsaEJBDeployUtility
Building: /WsaEJBDeployUtility
Writing output file
Shutting down workbench.
EJBDeploy complete.
0 Errors, 1 Warnings, 1 Informational Messages
```

Archived

Tip: WebSphere Application Server also provides a set of Ant tasks that you can use to automate the packaging and deployment of your applications. One of those tasks allows you to call EJBDeploy. Search for *Ant tasks* in the Information Center for more details.

16.3 Deploying the application

In this section, we show the steps required to deploy the application to WebSphere Application Server. We show you how to deploy a regular EAR file as well as an Enhanced EAR file, and then also how to not honor the configuration information packaged into the Enhanced EAR file.

Follow these steps to deploy the application:

1. Select **Applications** → **Install New Application** from the administrative console navigation bar.
2. Check the **Local file system** box and click the **Browse** button to locate the **WebSphereBank.ear** file. Then click **Next**.

   From the install panels you can install files that are located either on the same machine as the browser you are using to access the WebSphere administrative console, the local file system option, or on the WebSphere Application Server itself, the remote file system option. If you select the Local file system option, the administrative console automatically uploads the file you select to the application server, or to the deployment manager if this is a distributed server environment. If select the Remote file system check box, you can browse all the nodes in the cell to find the file. The file is then, if necessary, uploaded to the application server or deployment manager.
3. In the next window, specify default bindings for the application you are deploying. Unless you check the Override option, bindings already specified in the EAR are not altered. The various bindings you can specify in this page are documented in Table 16-7 on page 931. If you do choose to override bindings, select **Generate Default Bindings** at the top of this window to apply changes to the application you are deploying.

   In this example, the bindings were set in the application EAR file using the Application Server Toolkit and there is no need to override them. The defaults are also correct.

Table 16-7 Application default bindings

Binding name	Detailed information
EJB prefix	You can generate default EJB JNDI names using a common prefix. EJBs for which you did not specify a JNDI name will get a default one, built by concatenating the prefix and the EJB name. If you specify a prefix of myApp/ejb, then JNDI names default to myApp/ejb/EJBName, such as myApp/ejb/Account.
Override	Enter whether you want to override the current bindings. By default, existing bindings are not altered.
EJB 1.1 CMP bindings	You can bind all EJB 1.1 CMP entity beans to a specific data source, including user ID and password.
Connection Factory bindings	You can bind all EJB modules to a specific data source. You will have to go to the next window to override this setting at the EJB level.
Virtual host bindings	You can bind all Web modules to a specific virtual host, such as webspherebank_host.
Specify bindings file	You can also create a specific bindings file using your favorite editor and load it during application installation by clicking **Browse** next to the specific bindings file. For information about using a bindings file, see 16.3.1, “Using a bindings file” on page 936.

4. Click **Next**.
5. The WebSphere Bank EAR file includes a was.policy file which grants the application access to certain resources, when WebSphere security is enabled. The installation panel warns you about this as a potential security exposure and asks you to decide whether to continue installing the application or abort. For the WebSphere Bank sample we are fine, so click **Continue**.

   The rest of the wizard is divided into steps.
6. Step 1: Select installation options

   Step 1 gives you a chance to review the installation options. You can specify various deployment options such as JSP precompiling, whether you want to generate EJB deployment code, or enable reloadable classloaders for Web modules.

   If you are deploying an Enhanced EAR file, this is where you make the decision whether to use the resource configuration information packaged in the Enhanced EAR file or not. If the EAR file you are installing is an Enhanced EAR, the install panel preselects the **Process embedded configuration**

check box. If you do not want to use the resource configuration information packaged in the Enhanced EAR file, you must deselect this check box.

Selecting the Pre-compile JSP option makes WebSphere compile all JSPs in the EAR file during install time. This causes the time-consuming task of JSP compilation to be performed during install time instead of during run time, avoiding the first user that hits the application to pay that penalty.

A second alternative to pre-compile JSPs is to use the JspBatchCompiler script found in the `bin` directory of the profile you are using to compile the JSPs after the application has been installed.

Important: The WebSphere Bank application uses JMS resources, which are not supported by the Enhanced EAR. It is important that you deselect the Process embedded configuration check box.

Click **Next**.

7. Step 2: Map modules to servers

   Select the server on which you want each module deployed. For better performance, we recommend that you deploy all modules from one application in a single server. Especially, do not separate the EJB clients, usually servlets in Web modules, from the EJBs themselves.

   Click the  icon to select all modules in the WebSphere Bank EAR file. In the Clusters and Servers box, select the **WebSphereBankServer**. Then click **Apply**. This assigns all modules to the WebSphereBankServer application server. If you deploy to a cluster, select the cluster instead of the single application server.

   See Figure 16-9 on page 933.

   Web servers: If you have a Web server defined, select both the Web server and WebSphereBankServer in the server list. Press and hold the CTRL key to select multiple servers. Mapping Web modules to Web servers ensures the Web server plug-in will be generated properly.

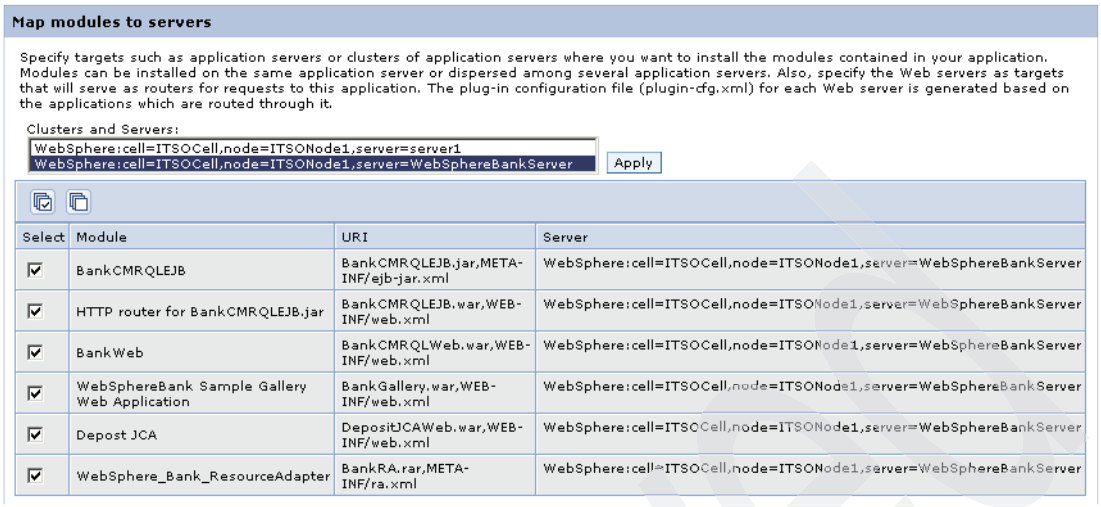


Figure 16-9 Mapping modules to application servers

> **Note:** Steps 3 - 11 allow you to define bindings. We have already taken care of this using the Application Server Toolkit when we packaged the EAR file. You can skip directly to Step 12 if you like. See 17 on page 935.

8. Step 3: Select current back-end ID.

   A single EAR file can contain multiple database mappings. At deployment time, you can choose which one you want to use. In this case, set it to `DB2EXPRESS_V82` because this is the version we are using.

9. Step 4: Provide listener bindings for message-driven beans

   In this step you can select listener ports or ActivationSpecs to which to bind your message-driven beans. The WebSphere Bank application requires the bindings shown in Table 16-8.

Table 16-8 Listener bindings for message-driven beans

Message-driven bean	ActivationSpec
BankListener	eis/BankActivationSpec
BankATMListener	eis/com.ibm.websphere.samples.bank.adapter.BankMessageListener

We configured this in 15.3.3, “Binding the message-driven bean to an ActivationSpec” on page 864 so the defaults shown should be correct.

Click **Next**.

10. Step 5: Provide JNDI names for beans

    Use this window to bind the enterprise beans in your application or module to a JNDI name. In 15.3.1, “Defining EJB JNDI names” on page 861, we defined these values in Table 16-9, so the defaults shown should be correct.

Table 16-9 Webbank enterprise bean JNDI names

EJB Name	JNDI Name
Sender session bean	ejb/Bank/Sender
Transfer session bean	ejb/Bank/Transfer
Account entity bean	ejb/Bank/Account
Customer entity bean	ejb/Bank/Customer

    Click **Next**.

11. Step 6: Bind message destination references to administered objects

    In this step, bind the message destination references to their JNDI names. Again, the values configured in the WebSphere Bank are fine.

    Click **Next**.

12. Step 7: Map JCA resource references to resources

    Each J2C object must be bound to a JNDI name. For the WebSphere Bank application, the ActivationSpec com.ibm.websphere.samples.bank.adapter.BankMessageListener should therefore be bound to a JNDI name.

    In 15.3.3, “Binding the message-driven bean to an ActivationSpec” on page 864 we configured these values, which are fine.

    Click **Next**.

13. Step 8: Provide default data source mapping for modules containing 2.x entity beans

    Specify the default data source for the EJB 2.x module containing 2.x CMP beans. In 15.3.4, “Defining data sources for entity beans” on page 865 we defined the JNDI name for the EJBs in the BankCMRQLEJB module as jdbc/Bank. You see this in the window.

    Click **Next**.

14. Step 9: Map data sources for all 2.x CMP beans

    Specify an optional data source for each 2.x CMP bean. Mapping a specific data source to a CMP bean overrides the default data source for the module containing the enterprise bean (defined in step 7, (12 on page 934). We do not need to do anything here. Click **Next**.

Archived

15. Step 10: Map EJB references to beans

    Each EJB reference defined in your application must be mapped to an enterprise bean. We used the Application Server Toolkit to do this in 15.3.2, “Binding EJB and resource references” on page 862.

    Click **Next**.

16. Step 11: Map resource references to resources

    Each resource reference defined in the application must be mapped to the corresponding resource. The EJB module BankCMRQLEJB has a resource reference to jms/BankJMSConnFactory, which is shown here.

    Click **Next**.

    At this step we now get an Application Resource Warning. What this tells us is that the resource we just referenced, jms/BankJMSConnFactory, is not defined for the specific application server to which we are installing our application. This is fine because the resource is defined at the node level instead.

    Click **Continue**.

17. Step 12: Map virtual hosts for Web modules

    For each Web module, select the virtual host we created for the application (websphere bank_host).

    Click **Next**.

18. Step 13: Map security roles to users and groups

    Because the WebSphere Bank EAR file contains security roles, we need to map them to users and groups in our target environment. However, because we have not enabled J2EE security (global security), WebSphere will not authenticate users trying to access the application. As a result we do not need to map the roles to users and groups.

    Click **Next**.

19. Step 14: Ensure all unprotected 2.x methods have the correct level of protection

    By default, EJB methods are unprotected. On this window, you can elect to refuse all calls to unprotected methods, or specify which methods you want to exclude.

    Again, because we have not enabled J2EE security, WebSphere will not authenticate users trying to access the EJBs.

    Click **Next**.

Archive

20. Step 15: Summary

    The Summary window gives an overview of application deployment settings. If those settings are fine, click **Finish** to deploy the application.

21. Save the configuration.

    If you are working in a distributed server environment, make sure you synchronize the changes with the nodes so they application is propagated to the target application server (s).

Deployment is now complete. If you mapped the Web modules to a Web server, make sure the Web server plug-in is regenerated and propagated to the Web server. For a quick refresh, restart the Web server.

You can now start the WebSphereBankServer application server and try the application by invoking `http://www.webspherebank.com/WebSphereBank`. Make sure that the host name is one of the three names you have added to the webspherebank_host definition. See 16.1.5, "Creating the virtual host for IBM HTTP Server and Apache" on page 915.

16.3.1 Using a bindings file

If generating default bindings during deployment, default names suitable for most applications are used. However, these defaults do not work if:

- You want to explicitly control the global JNDI names of one or more EJB files.
- You need tighter control of data source bindings for container-managed persistence (CMP) beans. That is, you have multiple data sources and need more than one global data source.
- You must map resource references to global resource JNDI names that are different from the `java:comp/env` name

In such cases, you can use a specific bindings file to customize the bindings created.

To use a bindings file when installing an application, load it by clicking **Browse** next to the Specific bindings file option. When using this file, only specify bindings that differ from the defaults, not the full bindings.

Example 16-6 is an example showing a bindings file used to change the JNDI name of an EJB:

Example 16-6 Using a bindings file to change the JNDI of an EJB

```
<?xml version="1.0"?>
<!DOCTYPE dfltbndngs SYSTEM "dfltbndngs.dtd">
<dfltbndngs>
```

```
  <module-bindings>
    <ejb-jar-binding>
      <jar-name>helloEjb.jar</jar-name>
<!-- this name must match the module name in the .ear file -->
      <ejb-bindings>
        <ejb-binding>
          <ejb-name>HelloEjb</ejb-name>
<!-- this must match the <ejb-name> entry in the EJB jar DD -->
          <jndi-name>com/acme/ejb/HelloHome</jndi-name>
        </ejb-binding>
      </ejb-bindings>
    </ejb-jar-binding>
  </module-bindings>
</dfltbndngs>
```

For more information about creating specific bindings file, refer to the Information Center.

16.4 Deploying application clients

To run a Java-based client/server application, the client application executes in a client container of some kind. You might, for example, use a graphical Swing application that calls EJBs on an application server. WebSphere Application Server V6 supports the following five types of application client environments:

- J2EE application client

  This client uses services provided by the J2EE Client Container.

  This client is a Java application program that accesses EJBs, JDBC databases, and JMS queues. The J2EE application client program runs on client machines. This program allows the same Java programming model as other Java programs. However, the J2EE application client depends on the application client runtime to configure its execution environment, and it uses the JNDI name space to access resources, the same as you would in a normal server application like a servlet).

  The J2EE application client brings the J2EE programming model to the client, and provides:

  - XML deployment descriptors
  - J2EE naming (java:comp/env) including EJB references and resource references

  The J2EE application client is launched using the launchClient script which sets up the environment with the necessary classpaths and so on, for you.

Archived

- Thin application client

  This client does not use services provided by the J2EE Client Container.

  This client provides a lightweight Java client programming model and is best suited for use in situations where a Java client application exists, but the application must be enhanced to make use of EJBs. It can also be used where the client application requires a thinner, more lightweight environment than the one offered by the J2EE application client. The Thin application client includes the IBM JDK. When launching the Thin application client you must set up the correct classpaths yourself and make sure that the required libraries for your application and the WebSphere libraries are included.

- Pluggable application client

  This client does not use services provided by the J2EE Client Container.

  This client is similar to the Thin application client, but does not include a JVM. The user is required to provide a JVM. It can also use the Sun JDK instead of the IBM JDK.

- Applet application client

  In the Applet client model, a Java applet embedded in an HTML document executes in a Web browser. With this type of client, the user accesses an enterprise bean in the application server through the Java applet in the HTML document.

- ActiveX to EJB Bridge application client

  The ActiveX application client allows ActiveX programs to access enterprise beans through a set of ActiveX automation objects. The ActiveX application client uses the Java Native Interface (JNI) architecture to programmatically access the Java virtual machine (JVM) API. Therefore, the JVM code exists in the same process space as the ActiveX application (Visual Basic, VBScript, or Active Server Pages files) and remains attached to the process until that process terminates.

The capabilities of the different application clients are shown in Table 16-10.

Table 16-10 Application client features comparison

Available functions	J2EE client	Thin client	Pluggable client	Applet client	ActiveX client
Provides all the benefits of a J2EE platform	Yes	No	No	No	Yes
Portable across all J2EE platforms	Yes	No	No	No	No

Available functions	J2EE client	Thin client	Plugg-able client	Applet client	ActiveX client
Provides the necessary run-time support for communication between a client and a server	Yes	Yes	Yes	Yes	Yes
Supports the use of nicknames in the deployment descriptor files.	Yes	No	No	No	Yes
Supports use of the RMI-IIOP protocol	Yes	Yes	Yes	Yes	Yes
Browser-based application	No	No	No	Yes	No
Enables development of client applications that can access enterprise bean references and CORBA object references	Yes	Yes	Yes	Yes	Yes
Enables the initialization of the client application run-time environment	Yes	No	No	No	Yes
Supports security authentication to enterprise beans	Yes	Yes	Yes	Limited	Yes
Supports security authentication to local resources	Yes	No	No	No	Yes
Requires distribution of application to client machines	Yes	Yes	Yes	No	Yes
Enables access to enterprise beans and other Java classes through Visual Basic, VBScript, and Active Server Pages (ASP) code	No	No	No	No	Yes
Provides a lightweight client suitable for download	No	Yes	Yes	Yes	No

Archive

Available functions	J2EE client	Thin client	Plugg-able client	Applet client	ActiveX client
Enables access JNDI APIs for enterprise bean resolution	Yes	Yes	Yes	Yes	Yes
Runs on client machines that use the Sun Java Runtime Environment	No	No	Yes	No	No
Supports CORBA services (using CORBA services can render the application client code nonportable)	Yes	No	No	No	No

Install the application client environments from the WebSphere installation panels by selecting the **Launch the installation wizard for WebSphere Application Clients** option. The installation package contains the following installable components:

- IBM Java Runtime Environment (JRE), or an optional full Software Development Kit
- WebSphere Application Server run time for J2EE application client applications, or Thin application client applications
- An ActiveX to EJB Bridge run time for ActiveX to EJB Bridge application client applications (only for Windows)
- IBM plug-in for Java platforms for Applet client applications (Windows only)

Note: The J2EE client is automatically installed as part of a full WebSphere install. In other words, if you will run the client application on a machine that already has WebSphere installed, you do not need to install the WebSphere J2EE client on top.

16.4.1 Defining application client bindings

The WebSphere Bank sample application provides four client applications, the GetAccounts, FindAccounts, TransferWS and TransferJMS clients. The various client applications demonstrate the capabilities of the WebSphere Bank sample application.

For an application client to be able to access resources, such as EJBs provided by a J2EE server application, the proper bindings must be set up.

You need to specify the complete naming structure to reach the server where the EJBs are deployed. For example, the machine where we deployed the application for testing has the Network Deployment version installed. The WebSphere Bank application is running in the WebSphereBank application server on node ITSONode1.

Therefore, the ejb/Bank/Customer EJB reference can be bound to:

```
cell/nodes/ITSONode1/servers/WebSphereBankServer/ejb/Bank/Customer
```

If you have created a cluster of application servers, use:

```
cell/clusters/<clusterName>/ejb/Bank/Customer
```

You will also need to change the provider URL, according to the target server. If you are running in a single server environment you can simply use:

```
ejb/Bank/Customer
```

When you have configured the proper bindings for the resources, you must export the EAR file and copy it to the client machine. Although you do not need the complete contents of the EAR file to run the application client, for example the Web modules, it is better to keep a single EAR file. This is mainly for maintenance purposes.

For more information about JNDI and naming, please refer to Chapter 13, “WebSphere naming implementation” on page 769.

16.4.2 Launching the J2EE client

A J2EE client application needs a container to run in. In this example we will use the J2EE application client container. This container can be started using the launchClient program in the *<<was_home>>*/bin directory. The launchClient program has the following syntax:

```
Usage: launchClient [-profileName pName | -JVMOptions options | -help | -?]
<userapp> [-CC<name>=<value>] [app args]
```

The elements of syntax are:

-profileName	This option defines the profile of the Application Server process in a multi-profile installation. The -profileName option is not required for running in a single profile environment or in an Application Clients installation. The default is default_profile.
-JVMOptions	This is a valid Java standand or nonstandard option string. Insert quotation marks around the option string.

`-help, -?`	Print the usage information.
`<userapp.ear>`	Type the path/name of the .ear file containing the client application.

The -CC properties are for use by the Application Client runtime. There are numerous parameters available and because of this we only describe the more commonly used ones. For full explanation of all parameters execute **`launchClient -help`**.

`-CCverbose`	Use this option with `<true | false>` to display additional informational messages. The default is `false`.
`-CCclasspath`	This property is a classpath value. When an application is launched, the system classpath is not used. If you need to access classes that are not in the EAR file or part of the resource classpaths, specify the appropriate classpath here. Multiple paths can be concatenated.
`-CCjar`	This is the name of the client JAR file within the EAR file that contains the application you want to launch. This argument is only necessary when you have multiple client JAR files in the EAR file.
`-CCBootstrapHost`	This option is the name of the host server you want to connect to initially. The format is `your.server.ofchoice.com`.
`-CCBootstrapPort`	This option is the server port number. If not specified, the WebSphere default value (2809) is used.
`-CCproviderURL`	This option provides bootstrap server information that the initial context factory can use to obtain an initial context. WebSphere Application Server initial context factory can use either a CORBA object URL or an IIOP URL. CORBA object URLs are more flexible than IIOP URLs and are the recommended URL format to use. This value can contain more than one bootstrap server address. This feature can be used when attempting to obtain an initial context from a server cluster. In the URL, you can specify bootstrap server addresses for all servers in the cluster. The operation will succeed if at least one of the servers is running, eliminating a single point of failure. The address list does not process in a particular order. For naming operations, this value overrides the `-CCBootstrapHost` and `-CCBootstrapPort` parameters. An example of a CORBA object URL specifying multiple systems is: `-CCproviderURL=corbaloc:iiop:myserver.mycompany.com:9810,:mybackupserver.mycompany.com:2809`

Archive

-CCtrace	Use this option with <true|false> to have WebSphere write debug trace information to a file. The value true is equivalent to a trace string value of com.*=all=enabled. Instead of the value true you can specify a trace string, for example, -CCtrace=com.ibm.ws.client.*=all=enabled. Multiple trace strings can be specified by separating them with a colon (:). You might need this information when reporting a problem to IBM Service. The default is false.
-CCtracefile	This option is the name of the file to which to write trace information. The default is to output to the console.
-CCpropfile	This option is the name of a properties file containing launchClient properties. In the file, specify the properties without the -CC prefix. For example: verbose=true.

The app args are for use by the client application and are ignored by WebSphere.

To start the WebSphere Bank GetAccounts client using the launchClient command execute the command shown in Figure 16-7:

Example 16-7 Launching WebSphere Bank application client

```
C:\WebSphere\AppServer\profiles\AppSrv01\bin>launchClient.bat
c:\WebSphereBankClient.ear -CCBootstrapPort=2809 -CCjar=GetAccounts.jar 100

IBM WebSphere Application Server, Release 6.0
J2EE Application Client Tool
Copyright IBM Corp., 1997-2004
WSCL0012I: Processing command line arguments.
WSCL0013I: Initializing the J2EE Application Client Environment.
WSCL0035I: Initialization of the J2EE Application Client Environment has
completed.
WSCL0014I: Invoking the Application Client class
com.ibm.websphere.samples.bank.client.GetAccounts
Get the account numbers owned by a certain customer

Getting the customer home...

Done....

Finding the customer....
Done....
Account Number: 101
All done!

Finding the customer from invoke ....
Account Number: 101
```

```
All done!

Get accounts owned by customer from invoke...
Account Number: 101
Account balance: 200.0
Get number of customer accounts from invoke...
Account :1
All done!
```

Because the WebSphereBank EAR file contains multiple client applications (JAR files) the `-CCjar` option must be used to specify which client application to launch.

The WebSphere Bank GetAccounts client application is a text-mode application that simply displays the accounts for a certain customer number 100 in our example.

16.5 Updating applications

WebSphere Application Server V6 introduces new features that allow applications to be updated and restarted at a more fine-grained level than in earlier versions. In V6 it is possible to update only parts of an application or module and only the necessary parts are restarted. In V6 you can:

- Replace an entire application (.ear file)
- Replace, add, or remove a single module (.war, EJB .jar, or connector .rar file)
- Replace, add, or remove a single file
- Replace, add and remove multiple files by uploading a compressed file describing the actions to take

If the application is running while being updated, WebSphere Application Server automatically stops the application, or only its affected components, updates the application and restarts the application or components.

When updating an application, only the portion of the application code that changed needs to be presented to the system. The application management logic calculates the minimum actions that the system needs to execute in order to update the application. Under certain circumstances the update can occur without stopping any portion of the running application.

Also introduced in V6 is enhanced support for managing applications in a cluster for continuous availability. The new action, Rollout Update, sequentially updates an application installed on multiple cluster members across a cluster. After you update an application's files or configuration, use the Rollout Update option to

install the application's updated files or configuration on all cluster members of a cluster on which the application is installed.

Rollout Update does the following for each cluster member in sequence:

1. Saves the updated application configuration
2. Stops all cluster members on a given node
3. Updates the application on the node by synchronizing the configuration
4. Restarts the stopped cluster members on that node

This action updates an application on multiple cluster members while providing continuous availability of the application.

16.5.1 Replacing an entire application EAR file

To replace a full EAR with a newer version, do the following:

1. Select **Applications** → **Enterprise Applications**. Select the application to update and click the **Update** button.
2. On the Preparing for the application installation panel, Select the **Full application** option.
3. Select either the **Local file system** or **Remote file system** option. Click the **Browse** button to select the updated EAR file. Click **Next**.
4. Proceed through the remaining panels and make any changes necessary. For information about the panels see “Deploying the application” on page 930. On the Summary panel click **Finish**.
5. When the application has been updated in the Master repository, select the **Save To Master Configuration** link.
6. If you are working in a distributed server environment, make sure the **Synchronize changes with Nodes** option is selected so that the application is distributed to all nodes. Click the **Save** button. The application is distributed to the nodes, updated and restarted as necessary.
7. If the application update changes the set of URLs handled by the application (servlet mappings added, removed or modified), make sure the Web server plug-in is regenerated and propagated to the Web server.

16.5.2 Replacing or adding an application module

To replace only a module, such as an EJB or Web module of an application, do the following:

1. Select **Applications** → **Enterprise Applications**. Select the application to update and click the **Update** button.

Archived

2. On the Preparing for the application installation panel, select the **Single module** option.
3. In the **Relative path to module** field, enter the relative path to the module to replace in the EAR file. For example, if you were to replace the HelloWeb.war Web module in the HelloApp EAR file, you would enter `HelloWeb.war`. If you enter a path that does not exist in the EAR file, the module will be added.
4. Select either the **Local file system** or **Remote file system** option and click the **Browse** button to select the updated module.
5. For Web modules, also enter the context root (e.g. `HelloWeb`) in the **Context root** field.
6. Click **Next**.
7. Proceed through the remaining panels and make any changes necessary. For information about the panels see “Deploying the application” on page 930. On the Summary panel click **Finish**.

> **Note:** If you are adding a module, make sure to select the correct target server for the module in the Map modules to servers step.

8. When the application has been updated in the Master repository, select the **Save To Master Configuration** link.
9. If you are working in a distributed server environment, make sure the **Synchronize changes with Nodes** option is selected so that the application is distributed to all nodes. Click the **Save** button. The application is distributed to the nodes, updated and restarted as necessary.
10. If the application update changes the set of URLs handled by the application (servlet mappings added, removed or modified), make sure the Web server plug-in is regenerated and propagated to the Web server.

16.5.3 Replacing or adding single files in an application or module

To replace a single file such as a GIF image or a properties file in an application or module, do the following:

1. Select **Applications** → **Enterprise Applications**. Select the application to update and click the **Update** button.
2. On the Preparing for the application installation panel, select the **Single file** option.
3. In the **Relative path to file** field, enter the relative path to the file to replace in the EAR file. For example, if you were to replace the `logo.gif` in the `images` diectory of the `HelloWeb.war` Web module, you would enter

`HelloWeb.war/images/logo.gif`. If you enter a path or file that does not exist in the EAR file, it will be added.

4. Select either the **Local file system** or **Remote file system** option and click the **Browse** button to locate the updated file. Click **Next**.
5. On the Updating Application panel, click **OK**.
6. When the application has been updated in the Master repository, select the **Save To Master Configuration** link.
7. If you are working in a distributed server environment, make sure the **Synchronize changes with Nodes** option is selected so that the application is distributed to all nodes. Click the **Save** button. The application is distributed to the nodes, updated and restarted as necessary.

16.5.4 Removing application content

Files can also easily be removed either from an EAR file or from a module in an EAR file.

Removing files from an EAR file

To remove a file from an EAR file, do the following:

1. Select **Applications** → **Enterprise Applications**. Select the application to remove the file from and click the **Remove File** button.
2. In the Remove file dialog box, select the file to be removed in the list as shown in Figure 16-10 and click **OK**.

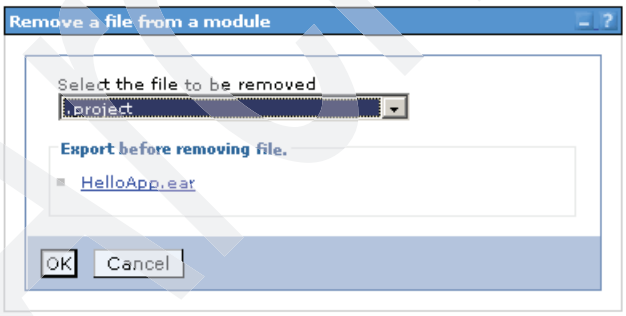


Figure 16-10 Removing a file from an application

3. Save the configuration.

Archive

Removing files from a module

To remove a file from a module, do the following:

1. Select **Applications** → **Enterprise Applications** and click the link for the application to which the module belongs.
2. In the Related items list on the right-hand part of the screen, click the link for Web modules, EJB modules or Connector modules, depending on the type of module the file to remove is.
3. Select the module and click the **Remove File** button.
4. In the Remove file dialog box, select the file to be removed in the list as shown Figure 16-10 and click **OK**.
5. Save the configuration.

16.5.5 Performing multiple updates to an application or module

Multiple updates to an application and its modules can be packaged in a compressed file, .zip or .gzip format, and uploaded to WebSphere Application Server V6. The uploaded file is analyzed and the necessary actions to update the application is taken.

Depending on the contents of the compressed file, this method to update an application can replace files in, add new files to, and delete files from the installed application all in one single administrative action. Each entry in the compressed file is treated as a single file and the path of the file from the root of the compressed file is treated as the relative path of the file in the installed application.

- To replace a file, a file in the compressed file must have the same relative path as the file to be updated in the installed application.
- To add a new file to the installed application, a file in the compressed file must have a different relative path than the files in the installed application.
- To remove a file from the installed application, specify metadata in the compressed file using a file named `META-INF/ibm-partialapp-delete.props` at any archive scope. The `ibm-partialapp-delete.props` file must be an ASCII file that lists files to be deleted in that archive with one entry for each line. The entry can contain a string pattern such as a regular expression that identifies multiple files. The file paths for the files to be deleted must be relative to the archive path that has the `META-INF/ibm-partialapp-delete.props` file.
- To delete a file from the EAR file (not a module), include a `META-INF/ibm-partialapp-delete.props` in the root of the compressed file. In the .props file, list the files to be deleted. File paths are relative to the root of the EAR file.

Archived

For example, to delete a file named `docs/readme.txt` from the root of the HelloApp.ear file, include the line `docs/readme.txt` in the `META-INF/ibm-partialapp-delete.props` file in the compressed file.

- To delete a file from a module in the EAR, include a `module_uri/META-INF/ibm-partialapp-delete.props` in the compressed file. The `module_uri` part is the name of the module, such as HelloWeb.war.

  For example, to delete `images/logo.gif` from the HelloWeb.war module include the line `images/logo.gif` in the `HelloWeb.war/META-INF/ibm-partialapp-delete.props` file in the compressed file.

- Multiple files can be deleted by specifying each file on its own line in the metadata .props file.

Regular expressions can also be used to target multiple files. For example, to delete all JavaServer Pages (.jsp files) from the HelloWeb.war file, include the line `.*jsp` in the `HelloWeb.war/META-INF/ibm-partialapp-delete.props` file. The line uses a regular expression, `.*jsp`, to identify all .jsp files in the HelloWeb.war module.

As an example, assume we have prepared the compressed `HelloApp_update.zip` file shown in Figure 16-11.

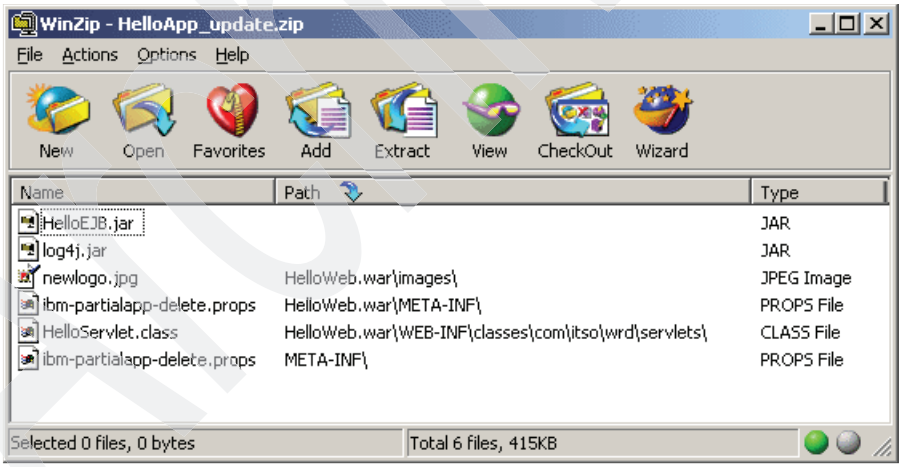


Figure 16-11 HelloApp_update.zip compressed file

The META-INF/ibm-partialapp-delete.props file contains the following line:

```
docs/readme.txt
```

The HelloWeb.war/META-INF/ibm-partialapp-delete.props contains the following lines:

```
images/logo.gif
```

When performing the partial application update using the compressed file, WebSphere does the following:

- Adds the log4j.jar file to the root of the EAR.
- Updates the entire HelloEJB.jar module.
- Deletes the docs/readme.txt file (if it exists) from the EAR file, but not from any modules.
- Adds the images/newlogo.jpg file to the HelloWeb.war module.
- Updates the HelloServlet.class file in the `WEB-INF/classes/com/itso/wrd/servlets` directory of the HelloWeb.war module.
- Deletes the images/logo.gif file from the HelloWeb.war module.

To perform the actions specified in the HelloWeb_updated.zip file do the following:

1. Select **Applications** → **Enterprise Applications**. Select the application to update and click the **Update** button.
2. On the Preparing for the application installation panel, select the **Partial Application** option.
3. Select either the **Local file system** or **Remote file system** option and click the **Browse** button to select the compressed ZIP file with the modifications you have created. Click **Next**.
4. On the Updating Application panel, click **OK**.
5. When the application has been updated in the Master repository select the **Save To Master Configuration** link.
6. If in a distributed server environment, make sure the **Synchronize changes with Nodes** option is selected so that the application is distributed to all nodes. Click the **Save** button. The application is distributed to the nodes, updated and restarted as necessary
7. If the application update changes the set of URLs handled by the application (servlet mappings added, removed or modified), make sure the Web server plug-in is regenerated and propagated to the Web server.

Archived

16.5.6 Rolling out application updates to a cluster

The new Rollout Update feature allows you to easily roll out a new version of an application, or part of an application using the techniques described previously, to a cluster. The Rollout Update feature takes care of stopping the cluster members, distributing the new application, synchronizing the configuration and restarting the cluster members. The operation is done sequentially over all cluster members in order to keep the application continuously available.

When stopping and starting the cluster members, the Rollout Update feature works on node level, so all cluster members on a node are stopped, updated and then restarted, before the process continues to the next node.

Because the Web server plug-in module is not able to detect that an individual application on an application server is unavailable, the Rollout Update feature always restarts the whole application server hosting the application. Because of this, if HTTP session data is critical to your application it should either be persisted to database or replicated to other cluster members using the memory-to-memory replication feature.

The order in which the nodes are processed and the cluster members are restarted is the order in which they are read from the cell configuration repository. There is no way to tell the Rollout Update feature to process the nodes and cluster members in any particular order.

Assume we have an environment with two nodes, ITSONode1 and ITSONode2, and a cluster called HelloCluster, which has one cluster member on each node (HelloServer1 on ITSONode1 and HelloServer2 on ITSONode2). Assume we have an application called HelloApp deployed and running on the cluster. To update this application using the Rollout Update feature we would to the following:

1. Select **Applications** → **Enterprise Applications**. Select the application to update and click the **Update** button.
2. On the Preparing for the application installation panel, select the appropriate action depending on the type of update. In this example we will update the entire application EAR to a new version, so we select the **Full Application** option.
3. Select either the **Local file system** or **Remote file system** option and click the **Browse** button to select the updated EAR file. Click **Next**.
4. Proceed through the remaining panels and make any changes necessary. For information about the panels see “Deploying the application” on page 930. On the Summary panel click **Finish**.
5. When the application has been updated in the master repository, the status window shown in Figure 16-12 on page 952 is displayed:

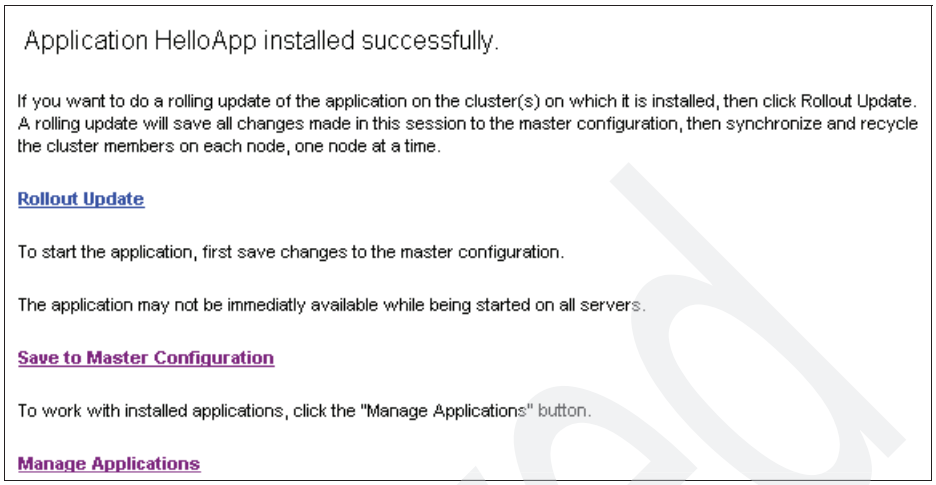


Figure 16-12 Preparing for application rollout

You then have two options to start the rollout action:

- Click the **Rollout Update** link.
- Click the **Manage Applications** link and on the Enterprise Applications panel, select the application and click the **Rollout Update** button.

Note: Do not click the Save to Master Configuration link or otherwise save the configuration yourself. The Rollout Update will do that for you. If you save the configuration yourself, the rollout update action will be canceled and it will be handled as a normal application update.

During the rollout the window in Figure 16-13 on page 953 is displayed in the status window.

Archived

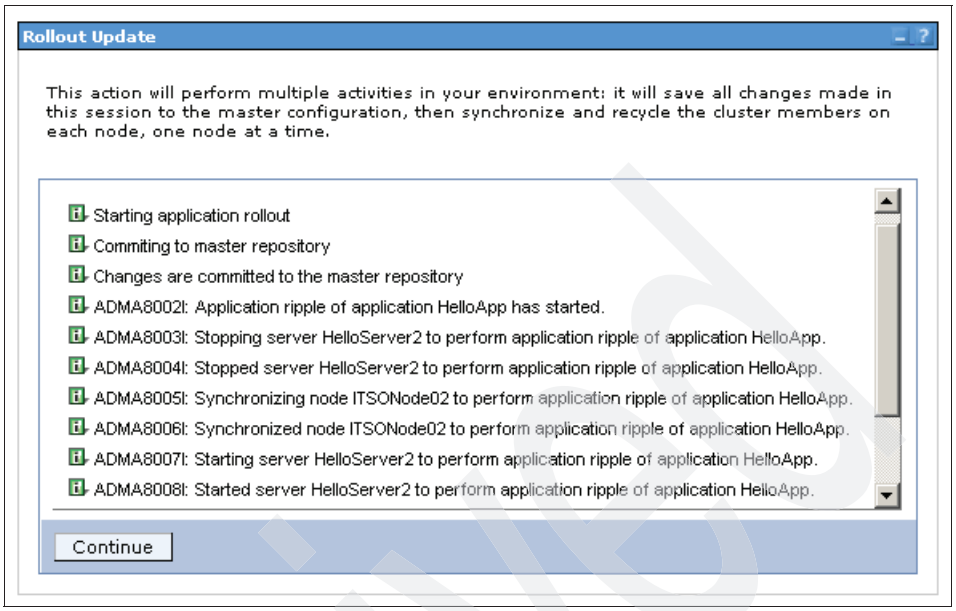


Figure 16-13 Rolling out an application

For each node, the cluster members are stopped, the application is distributed and they are restarted. When the rollout has completed, click **Continue**.

6. If the application update changes the set of URLs handled by the application (servlet mappings added, removed or modified), make sure the Web server plug-in is regenerated and propagated to the Web server.

Note: The automatic file synchronization of the node agent is temporarily disabled during the rollout process and then re-enabled afterwards, if it was previously enabled. The Rollout Update feature works regardless of the automatic file synchronization setting. However, in production systems the automatic synchronization is often disabled anyway to give the administrator greater control over exactly when changes made to the cell configuration are distributed to the nodes.

Although the Rollout Update feature makes it very easy to rollout an application to a cluster while keeping the application continuously available, make sure that your application can handle the roll out.

For example, assume you have version 1.0 of an application running in a cluster consisting of two application servers, server1 and server2, and that HTTP session data is persisted to a database. When you roll out version 2.0 of the

application and server1 is stopped, the Web server plug-in redirects the users on server1 to server2. Then when server1 is started again, bringing up version 2.0 of the application, the plug-in will start distributing requests to server1 again. Now, if the application update incurred a change in the interface of any class stored in the HTTP session, when server1 tries to get these session objects from the database, it might run into a deserialization or class cast exception, preventing the application from working properly.

Another situation to consider is when the database structure changes between application versions, as when tables or column names change name or content. In that case, the whole application might need to be stopped and the database migrated before the new version can be deployed. The Rollout Update feature would not be suitable in that kind of scenario.

So it is very important to understand the changes made to your application before rolling it out.

16.5.7 Hot deployment and dynamic reloading

Hot deployment and dynamic reloading characterize how application updates are handled when updates to the applications are made by directly manipulating the files on the server. In either case, updates do not require a server restart, though they might require an application restart:

- Hot deployment of new components

  Hot deployment of new components is the process of adding new components such as WAR files, EJB JAR files, EJBs, servlets, and JSP files to a running application server without having to stop and then restart the application server.

  However, in most cases such changes require the application itself to be restarted, so that the application server runtime reloads the application and its changes.

- Dynamic reloading of existing components

  Dynamic reloading of existing components is the ability to change an existing component without the need to restart the application server for the change to take effect. Dynamic reloading can involve changes to the :

  - Implementation of an application component, such as changing the implementation of a servlet
  - Settings of the application, such as changing the deployment descriptor for a Web module

To edit the files manually, locate the binaries in use by the server. See “Repository files used for application execution” on page 109. Although the

application files can be manually edited on one or more of the nodes, these changes will be overwritten the next time the node synchronizes its configuration with the deployment manager. Therefore, it is recommended that manual editing of an application's files should only be performed in the master repository, located on the deployment manager machine.

Note: Unless you are familiar with updating applications by directly manipulating the server files, it might be better to use the administrative console Update wizard.

There are three settings that affect dynamic reload:

- Enable class reloading and Reloading interval for the enterprise application.

  In order for application files to be reloaded automatically after an update, Enable class reloading must be enabled and the Reloading interval must be greater than 0.

  Select **Applications** →**Enterprise Applications**. Double-click the application to open the configuration page. These settings can be found in the General Properties section.

- reloadingEnabled in the Web module extensions

  A Web container reloads a Web module only when this setting is enabled.

  This setting is found in the `ibm-web-ext.xmi` file and can be updated using the Application Server Toolkit. Open the Web module deployment descriptor and select the **Extensions** tab.

- Application class loader policy for the application server

  The application class loader policy should be set to **Multiple**. If it is set to Single, the application server will need to be restarted after an application update.

  Select **Servers** →**Application Servers**. Double-click the server to open the configuration page. The setting is found in the General Properties section.

For more information about using hot deployment and dynamic reload, see the *Updating applications* and *Hot deployment and dynamic reloading* topics in the Information Center.

Archive

Archived

17

WebSphere Rapid Deployment

WebSphere Rapid Deployment is a collection of tools and technologies introduced in WebSphere Application Server V6 that make application development and deployment easier than ever before.

WebSphere Rapid Deployment consists of the following:

- Support for annotation-based (Xdoclet) programming
- Fine-grained application updates
- Rapid deployment tools

In this chapter we briefly discuss the concept of annotation-based programming and then show how to use the rapid deployment tools for the following two project types (modes):

- Free-form development projects
- Automatic application installation projects

Using free-form development, you can quickly develop small J2EE applications without using an integrated development environment. Automatic application installation is an easy way to manage your applications by letting WebSphere automatically install, update and uninstall your EAR files.

© Copyright IBM Corp. 2005. All rights reserved.

17.1 Annotation-based programming

Annotation-based programming is a method of speeding up application development by reducing the number of artifacts you need to develop and manage on your own. By adding metadata tags to your Java source code, the WebSphere Rapid Deployment tools can automatically create and manage the artifacts required to build J2EE compliant modules and applications.

For example, consider a stateless session EJB. With annotation-based programming, you simply create a single Java source file containing the bean implementation logic, add a few tags indicating that you want to deploy this class as an EJB, and indicate which methods should be made public on the EJB interface. Using this single artifact, WebSphere Rapid Deployment can create:

- The home and remote interface classes
- A stateless session implementation wrapper class
- The EJB deployment descriptor (ejb-jar.xml)
- The WebSphere-specific binding data
- All of the remaining artifacts required to produce a J2EE application

All you have to deal with is a single Java artifact, the other required artifacts are generated, as shown in Figure 17-1 on page 959.

Archived

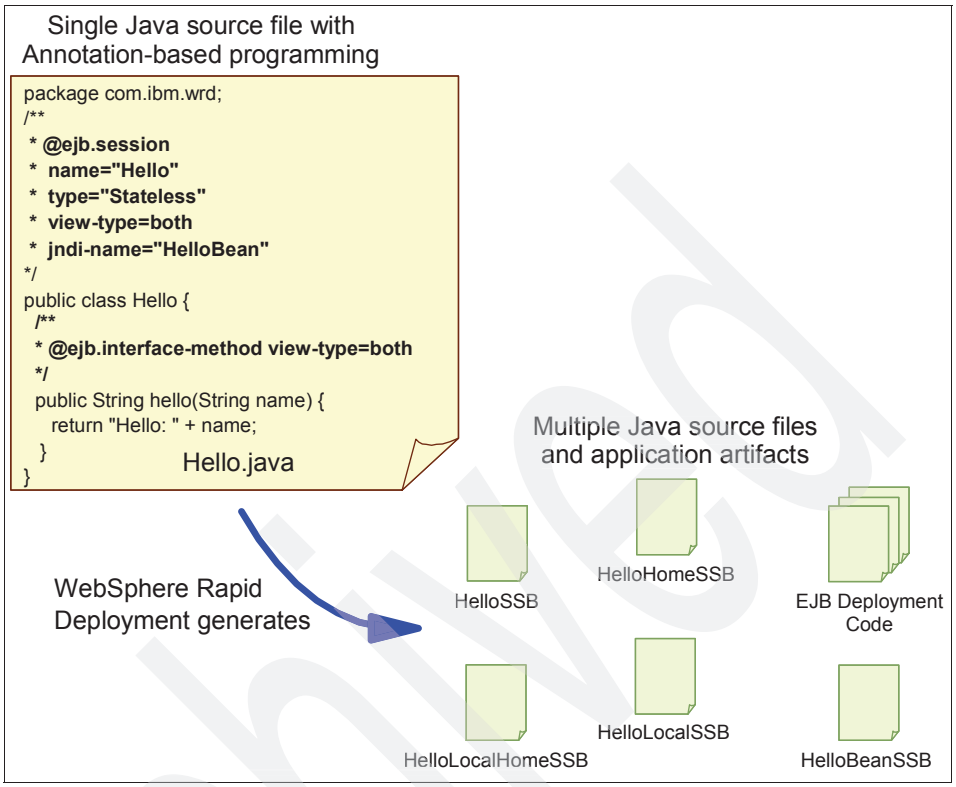


Figure 17-1 Artifacts generated by annotation-based class

WebSphere Rapid Deployment supports numerous tags to help you build your applications. These tags map directly to known J2EE artifacts and deployment descriptor elements. Specifically, WebSphere Rapid Deployment supports tags for the following artifact types and generation targets:

- EJBs
- Servlets
- Java classes
- Web services

WebSphere Rapid Deployment supports annotations using Javadoc-style comments in the package, class, field, or method declarations. The syntax uses XDoclet syntax, where it exists. Currently, WebSphere Rapid Deployment supports the syntax used by XDoclet for J2EE 1.3.

We do not go into the details of annotation-based programming in this book, but we do use it when developing the example application for showing the free-form development projects. For detailed information about annotation-based

Archived

programming, see the WebSphere Information Center. Search for *annotation* and resources on the Internet, for example:

`http://xdoclet.sourceforge.net/xdoclet/index.html`

17.2 Rapid deployment tools

Using the rapid deployment tools part of WebSphere Rapid Deployment you can:

- Package J2EE artifacts quickly into an EAR file.
- Deploy and test J2EE modules and full applications quickly on a server.
- Create a new J2EE application quickly without the overhead of using an integrated development environment (IDE).

For example, you can place full J2EE applications (EAR files), application modules (WAR files, EJB JAR files), or application artifacts (Java source files, Java class files, images, JSPs etc.) into a configurable location on your file system, referred to as the *monitored,* or *project,* directory. The rapid deployment tools then automatically detect added or changed parts of these J2EE artifacts and perform the steps necessary to produce a running application on an application server.

There are two ways to configure the monitored directory, each performing separate and distinct tasks. You can specify the monitored directory as a free-form project or an automatic application installation project.

With the free-form approach, you can place in a single project directory the individual parts of your application, such as Java source files that represent servlets or enterprise beans, static resources, XML files and other supported application artifacts. The rapid deployment tools use your artifacts to place them automatically in the appropriate J2EE project structure, generate any additional required artifacts to construct a J2EE compliant application and deploy that application on a target server.

The automatic application installation project allows you to quickly and easily install, update, and uninstall J2EE applications on a server. If you place EAR files in the project directory, they are deployed automatically to the server. If you delete EAR files from the project directory, the application is uninstalled from the server. If you place a new copy of the same EAR file in the project directory, the application is reinstalled. If you place WAR or EJB JAR files in the automatic application installation project, the rapid deployment tool generates the necessary EAR wrapper, then publishes that EAR file on the server. For RAR files, a wrapper is not created. The standalone RAR files are published to the server.

The advantage of using a free-form project is that you do not need to know how to package your application artifacts into a J2EE application. The free-form project does the packaging part for you. The free-form project is suitable when you just want to test something quickly, perhaps write a servlet that performs a task.

An automatic application installation project simplifies management of applications and relieves you of the burden of going through the installation panels in the WebSphere administrative console, or developing wsadmin scripts to automate your application deployment.

Figure 17-2 describes the two modes.

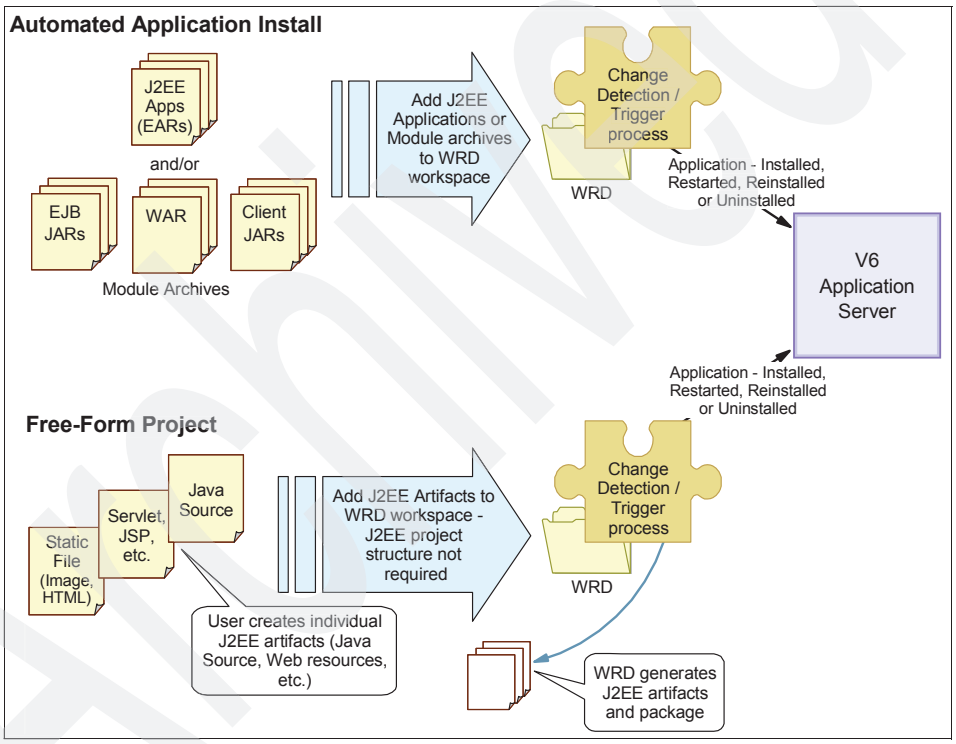


Figure 17-2 WebSphere Rapid Deployment modes

Note: The rapid deployment tools can be configured to deploy applications either onto a local or remote WebSphere Application Server.

17.3 Using rapid deployment commands

The rapid deployment tools are managed by two commands, **wrd-config** and **wrd**. Wrd-config is used to configure a workspace directory for a rapid deployment session, either free-form or automatic application installation mode. The wrd command is then used to tell the rapid deployment tools to start monitoring the directory and act on the changes you introduce to it.

17.3.1 wrd-config command

The **wrd-config** command is used to configure a workspace for either free-form development or automatic application installation mode.

The syntax for the **wrd-config** command is:

```
wrd-config.bat(sh) -project <project_name> -style <freeform| autoappinstall>
[<optional parameters>]
```

Note: Before running the **wrd-config** command, you must set up the WORKSPACE environment variable to point to the workspace directory your rapid deployment session will use. This is done by executing:

- Windows: **set WORKSPACE=<*workspace_root*>**
- Linux: **export WORKSPACE=<*workspace_root*>**

Table 17-1describes the mandatory parameters.

Table 17-1 wrd-config mandatory parameters

Parameter	Description
-project <*"project_name"*>	This is the name of the rapid deployment project that you want to create. The project name needs to be unique in your workspace.
-style <"freeform" | "autoappinstall">	The deployment approach - either the free-form or automatic application installation project is used as the deployment style.

The free-form project allows you to create or drop in your J2EE artifacts (such as servlet, EJB or general Java source or class files, JSPs, static Web content, and all other generic files) into the free-form project. These resources are then automatically compiled (if necessary), placed in the appropriate location in the J2EE project structure and deployed on an application server. Optionally, the rapid deployment tools can also export an EAR file to a target directory.

The automatic application installation project creates a single project that listens for fully composed EAR or module files. If EAR files are placed inside this project, the EAR file is deployed automatically to the server. If the EAR file is deleted, then that application is uninstalled from the server. If you place WAR or EJB jar files in an automatic application installation project, the rapid deployment tool generates the necessary EAR wrapper, then publishes that EAR file on the server. For RAR files, a wrapper is not created. The standalone RAR files are published to the server.

The **wrd-config** command also takes the optional parameters described in Table 17-2. Not all parameters are applicable to the automatic application installation mode, however.

Table 17-2 wrd-config optional parameters

Parameter	Description
-rebuild	Clean and rebuildsthe contents of the rapid deployment project. Use this to start a new session,
-configure	Open an interactive console session to modify any available parameters.
-runtime <“was51” | “was60”>	Targeting your runtime will configure which JRE library to use, based on the runtime location, and configure the project's classpath to contain the WebSphere runtime libraries. Specify as an identifier if either WebSphere Application Server v5.1 ("was51") or WebSphere Application Server v6.0 ("was60") is used as the target run-time environment for rapid deployment processing. If this parameter is not specified, the default setting is WebSphere Application Server v6.0. This parameter is only used with -runtimePath parameter.
-runtimePath <“*was_home*”>	Was_home is the directory where WebSphere Application Server is installed (for example, c:\WebSphere\AppServer).
-j2eeVersion <“1.3” | “1.4”>	Specify either J2EE version 1.3 or 1.4 is used for development and deployment.
-configPath “x:*filename*.xml”	Specify the destination file path for the configuration file, where x is the temporary directory. This file persists the configuration data to an XML file, which can later be used to drive other rapid deployment configuration sessions. If this path is not specified, the default location is in the root of the rapid deployment workspace.

Archive

Parameter	Description
-configData “x:*filename*.xml”	The path of an existing XML configuration file that is used to drive the configuration session. If this path is not specified, the default location of the XML configuration file is in the root of the rapid deployment workspace with the naming convention projectName_headlessconfig.xml, where projectName is the value specified in the -project parameter.
-listStyles	List the available deployment styles (AutoAppInstall and FreeForm) and their descriptions.
-listServers	List the available runtime server targets.
-properties	List the properties for a given deployment project. This parameter is only used with -project.
-buildMode	Specify to disable all console outputs. This is useful for silent builds.
-usage	Display the optional and required parameters for this command.

Note: The runtimePath is not used for targeting the server to publish against, but, rather, specifying the WebSphere libraries you want your application to compile against. If the runtime and runtimePath are not specified, the default runtime environment used is WebSphere Application Server v6.0.

When running **wrd-config**, an interactive Parameter Configuration Settings dialog window is displayed as shown in Example 17-1:

Example 17-1 Parameter Configuration Setting dialog window

```
------------------------------------------------------
Parameter Configuration Settings
------------------------------------------------------
Press ENTER to accept defaults
The * symbol denotes required input

 Enter the server name* ( server1 ) :
 Enter the server JMX host name* ( localhost ) :
 Enter the server JMX port number* ( 8880 ) :
 Enter a path for containing the exported EAR ( -- ) :
 Enter your server username ( -- ) :
 Enter your server password ( -- ) :
```

Table 17-3 describes the settings that should be configured in the dialog window. The order listed is the way they are presented by the dialog window.

Table 17-3 wrd-config Parameter Configuration Settings

Option	Description
serverName	Use the name of the server process you want to publish your application. For example, server1. For WebSphere Application Server Network Deployment, the server name is in the form *<cell_name>/<node_name>/<server_name>*. **Note:** WebSphere Rapid Deployment cannot publish to a cluster.
serverJMXHost	Use the host name of the machine containing the server to which you want to make a connection. For example, localhost. For WebSphere Application Server Network Deployment, type the host name of the Network Deployment Manager.
serverJMXPort	Use the server administrative port number, also known as the SOAP connector port. This port is used for making JMX connections with the server. For example, 8880. In a distributed server environment, use the SOAP port number of thedeployment manager. the default is 8879.
earExportPath	This is an optional field to specify the directory location of the output EAR file that is created by rapid deployment tools and contains the generated classes required for deployment. For example: c:\temp **Note:** When you specify the directory for the output EAR file and then run the wrd-config command, any existing output EAR file of the same name will be overwritten without warning. In addition, the EAR file gets overwritten as additional J2EE artifacts are added to the free-form project to reflect the new changes.
username	(Optional) If security is enabled, specify the user name for current active authentication settings defined in the server configuration.
password	(Optional) If security is enabled, specify the password for current active authentication settings defined in the server configuration.

Example 17-2 shows some examples of how to use the wrd-config command.

Example 17-2 wrd-config examples

To create a new free-form project called MyProject:

```
wrd-config.bat -project "MyProject" -style "freeform"
```

To create a new automatic application installation project called MyProject:

```
wrd-config.bat -project "MyProject" -style "autoappinstall"
```

To persist configuration data to an XML file:

```
WRD-config.bat -project "MyProject" -style "freeform" -runtime "was60"
-runtimePath "c:\WebSphere\AppServer" -configPath
"c:\configData\myHeadlessConfig.xml"
```

To create a new rapid deployment project using an existing XML configuration file:

```
wrd-config.bat -configData "c:\configData\myHeadlessConfig.xml"
```

To clean and rebuild an existing rapid deployment project:

```
wrd-config.bat -project "MyProject" -rebuild
```

To modify available deployment parameters:

```
wrd-config.bat -project "MyProject" -configure
```

To query properties of an existing rapid deployment project:

```
wrd-config.bat -project "MyProject" -properties
```

To query available rapid deployment styles and run-time targets:

```
wrd-config.bat -listStyles -listServers
```

Note: A `projectName_headlessconfig.xml` file is generated in the root of your rapid deployment workspace. Use this file later to configure a project with the same configuration, without having to be prompted again for these parameters. Use the optional `-configData` parameter to complete this task.

17.3.2 wrd command

The **wrd** command is used to configure the rapid deployment tools to start monitoring a project directory, configured by the **wrd-config** command, for changes.

The syntax for the **wrd** command is:

```
wrd.bat(sh) [<optional parameters>]
```

The wrd command has no mandatory parameters, but takes the optional parameters described inTable 17-4.

Table 17-4 wrd optional parameters

Parameter	Description
-monitor	Enable console feedback from the rapid deployment tool.
-project <"project_name">	Use the name of the rapid deployment project that you want to target to run in batch mode. This -project parameter is only used in conjunction with -batch parameter.
-batch	Enable batch mode on a specified target project. Batch mode runs a full build on a rapid deployment workspace and then shuts down the process.
-usage	Display the optional parameters for this command.

Note: Before starting a WebSphere Rapid Deployment session, make sure the application server that will host the application is started. Otherwise, you will see the following error message in the console output:

```
[04:31:32 PM] Publishing HelloApp to server_510658053
[04:31:32 PM] ERROR! Failed to make connection to WebSphere Application
Server.
```

17.4 Free-form projects

Free-form projects provide a simple way to develop small J2EE applications quickly, without having to use a full-blown integrated development environment. However, when developing larger applications, you almost always want the benefits of an integrated development environment, such as Rational Application Developer for WebSphere software to give you features like application modeling, version control, debugging, profiling, and so forth.

Also, if your application cannot conform to the assumptions and defaults made by the rapid deployment tools, you need to use an integrated development environment to give you total control over how your application is packaged.

When using free-form projects, the rapid deployment tools make certain assumptions about the J2EE application being created. The J2EE project structure created will hold one EJB module, one EJB client JAR, one Web module and one utility JAR, all encapsulated in one enterprise application (EAR file). There will be, at most one of each type of module. A module is created only

when necessary. So, unless you have introduced any EJB code in your project, no EJB module is created. Table 17-5 shows the naming conventions used for naming the modules.

Table 17-5 Naming conventions for free-form project contents

Naming conventions for the project folder	Project type
project_name	The free-form project
*project_name*App	A single enterprise application project
*project_name*EJB	A single EJB module project
*project_name*EJBClient	A single EJB client jar project
*project_name*Web	A single Web module project
*project_name*Utility	A single utility Java project

Project_name is the name given the project when setting up the environment and is the directory being monitored.

The rapid deployment tools will configure and update the manifest files and project references to maintain the following default settings:

- The EJB module project will have reference and visibility to the EJB client jar project.
- The Web module project will have reference and visibility to the EJB client jar project.
- The Web module project, EJB module project, and EJB client jar project will have reference and visibility to the utility project.
- The enterprise application project will contain the Web module project, EJB module project, and EJB client jar project.
- The utility Java project is added as a utility JAR inside the enterprise application project.

Each of the projects, excluding the enterprise application project, contains an imported_classes directory. The imported_classes folder is set as an exported entry on the classpath of its containing project. As a result, the directory and any contents within it, is visible to any projects referencing that project.

The initial configuration of the free-form project creates the following subdirectories in the project directory:

- The gen folder is a repository of non-Java artifacts that are generated by the rapid deployment tools.

- The `src` folder is a subdirectory of the `gen` folder and is a repository of Java artifacts generated by the rapid deployment tools. For example, if you drop an annotated session EJB, the remote, home and local interfaces are generated in `src` directory.
- The `bin` directory contains the class files that were compiled for any Java source files dropped into the free-form project.

  When Java source files are dropped into a free-form project, they are compiled into this project and the generated class files are copied into the target J2EE project's imported_classes folder. The source files are not copied into any of the J2EE project structures. The deployed application contains only class files.

If a classification of a resource changes from one artifact type to another that results in changes to its target J2EE module project, the old mapped locations are purged. For example, if Hello.java implemented a servlet, its class files map to the Web module project. If the content of Hello.java changes to implement an enterprise bean, its class files now map to the EJB module project. The dangling class in the Web project is removed.

Any resources that cannot be classified by the rapid deployment tools are mapped to the utility Java project, and their folder structure is preserved as created and dropped into the free-form project. For example, if the following resource exists in the free-form project, `/MyFreeForm/data/myproperties.props`, this file is mapped to `/MyFreeFormUtility/data/myproperties.props`, preserving the `data` folder. In addition, if the class files cannot be classified by the rapid deployment tools as other than a class file, the tool maps these files to the imported classes folder in the utility Java project.

When removing artifacts from the free-form project, the rapid deployment tools remove the artifacts from their mapped J2EE project structure location and, if applicable, also remove the deployment descriptor entries. However, the rapid deployment tools keep any project folders created.

The application generated can be published and deployed either on a local or remote WebSphere Application Server. When publishing to a local server, the server runs the application using the resources (.class files) within the workspace. As a result, the server itself does not maintain the resources for the application on the local server. To have the server maintain the resources within the application, use remote publishing, even though you are using a local server. This way, you can later modify server-specific configurations using the WebSphere administrative console.

To use remote publishing on a local server to keep resources in an application, do the following:

1. Open the `headless.props` file in the `workspace/.plugins/.metadata/com.ibm.ws.rapiddeploy.core` directory, where `workspace` is the directory where your rapid deployment project resides.
2. Set the parameter value of `TREAT_SERVER_AS_REMOTE` to `true`. This enables the WebSphere Administrative console to work with the application.

If the free-form project is configured to a local server (on the same machine running the rapid deployment session), the application synchronizes with the server after every change. If the free-form project is configured to a remote server, changes are batched together in one minute intervals to limit file transfers over the network for every change. A publish is triggered every minute, only when a change occurs.

The rapid deployment tools only manage the J2EE part of your application, not any additional resources (such as JDBC Providers, DataSources, JMS configuration, and so on) your application might also need. You must manually configure these resources using the WebSphere administrative console before your application can run.

17.5 Free-form development example

For the purpose of showing how to develop an application using a free-form project we will create a very very simple J2EE application. When completed, the application will contain the following artifacts:

- A stateless session EJB, called HelloEJB
- A servlet, called HelloServlet
- A JSP, called hello.jsp
- A GIF image, called logo.gif in a directory called images

The EJB has only a remote interface and only a single method, getGreeting, returning a String value. When invoked, HelloServlet looks up HelloEJB and calls its getGreeting method. It then forwards to the JSP, which displays the message along with the GIF image.

The EJB and servlet are developed using annotation-based programming to reduce the number of artifacts that need to be managed.

The WebSphere Rapid Deployment session will be set up to automatically produce an EAR file with the contents of our application.

The scenario we will use contains the following steps:

1. Create a workspace directory

2. Configure a free-form project in the workspace.
3. Launch a WebSphere Rapid Deployment session. This step configures the rapid deployment tools to monitor the project directory for changes.
4. Copy the EJB source code into the project directory.
5. Copy the servlet source code into the project directory.
6. Copy the JSP source code into the project directory.
7. Copy the GIF image into the project directory.
8. Verify that the code has been compiled, packaged it into a J2EE application and is installed on the application server.
9. Test the application.
10. Examine the EAR file exported.

17.5.1 Setting up the environment for free-form development

Before starting out with our free-form example project, we need to set up the environment. Do the following:

1. Open a command prompt. This will be used for running the rapid deployment tools in headless, nongraphical, mode. All output from the tools will be written to this window. We, therefore, also refer to it as the console window.
2. Create a workspace directory, for example `c:\wrd`, by typing:

   ```
   mkdir c:\wrd
   ```

   The workspace directory holds the projects with which you work.
3. Set the WORKSPACE environment variable by typing:

   ```
   set WORKSPACE=c:\wrd
   ```

   Tip: To list all environment variables currently set, type **`set`** in Windows, or **`export -n`** in Linux and press Enter.

4. Configure the rapid deployment tools to create a free-form project called **`Hello`** in the workspace directory by typing:

   ```
   cd <profile_home>\bin
   wrd-config.bat -project "Hello" -style "freeform"
   ```

   <*profile_home*> is the root directory of the application server profile you are using.

   This brings up the interactive Parameter Configuration Settings dialog window. Enter the information as requested in Example 17-3 on page 972.

Example 17-3 Parameter Configuration Settings

```
-------------------------------------------------------
Parameter Configuration Settings
-------------------------------------------------------
Press ENTER to accept defaults
The * symbol denotes required input

 Enter the server name* ( server1 ) : ITSOCell/ITSONode1/server1
 Enter the server JMX host name* ( localhost ) :
 Enter the server JMX port number* ( 8880 ) : 8879
 Enter a path for containing the exported EAR ( -- ) : c:\temp
 Enter your server username ( -- ) :
 Enter your server password ( -- ) :

Configuring the workspace...
Building the workspace...
WebSphere Rapid Deployment configuration completed.
```

When creating this example, we were running in a distributed server environment and, therefore, used the JMX port (8879) for our deployment manager. Our target application server is called `server1` and exists on node `ITSONode1` in the cell called `ITSOCell`.

If you were running on a standalone server, you would use the JMX port number, default 8880**,** for the standalone application server, default `server1`. In our environment, the deployment manager and the application server are located on the same physical machine so we used the default JMX hostname `localhost`. Also, we did not have WebSphere global security enabled, so we did not need to supply a username or password.

We entered `c:\temp` as the exported EAR target directory. This means that the WebSphere Rapid Deployment tools will produce an EAR file for us in this directory in addition to deploying the application to our server.

Note: Make sure the application server that the application will be published on (ITSOCell/ITSONode1/server1 in our example) is started before proceeding to the next step.

When the **`wrd-config`** command has finished, the directory structure shown in Figure 17-3 on page 973 is created in the workspace directory.

Archived

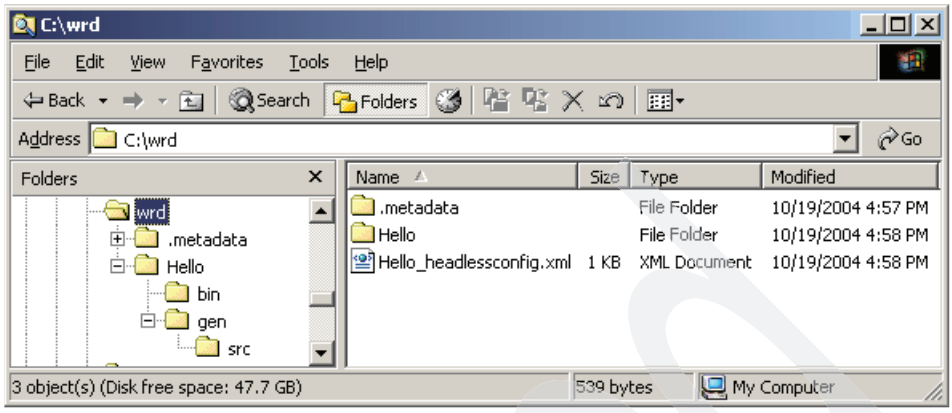


Figure 17-3 Initial free-form project structure

The `Hello` directory is the project directory and will be monitored by the tools. You can have multiple projects in a workspace as long as their names are unique. In the root of the workspace, the `Hello_headlessconfig.xml` file has been created. This file holds the configuration for our project.

5. Launch the WebSphere Rapid Deployment session by typing:

   ```
   wrd.bat -monitor
   ```

   This tells the rapid deployment tools to start monitoring the project directory for changes and display messages on the console.

> **Note:** The **`wrd`** command itself is used to tell the rapid deployment tools to start monitoring the project directory and the optional **`-monitor`** switch enables the console output messages.

The wrd command displays the messages in Example 17-4 in the console window:

Example 17-4 Launching WebSphere Rapid Deployment

```
Launching WebSphere Rapid Deployment.  Please wait...
Starting Workbench...

WebSphere Rapid Deployment ready for e-business...

Type 'q', 'quit', or 'exit' to shut down WebSphere Rapid Deployment processes.
```

Archive

To terminate a WebSphere Rapid Deployment session, type **Q, quit** or **exit** and press **Enter** in the console window. This stops the monitoring of the project directory. You can continue working where you stopped by simply running the **wrd** command again.

17.5.2 Adding application source code

When the WebSphere Rapid Deployment session is configured and monitoring of the project directory is activated, we can introduce the Java files and the other artifacts that make up our application. We will use annotated Java source code and let the rapid deployment compile it for us, but we could also have copied .class files directly to the project directory.

Adding EJB source code

To add EJB source code, do the following:

1. Create a directory called `com\itso\wrd\ejbs` in the `c:\wrd\Hello` project directory. The reason for doing this is that the Java source file must be placed in the correct directory structure, matching the class package name.
2. Using a text editor, add the EJB code shown in Example 17-5 and save the file as `HelloEJBBean.java` in the `c:\wrd\Hello\com\itso\wrd\ejbs` directory:

Example 17-5 Source code for HelloEJBBean.java

```
package com.itso.wrd.ejbs;
import javax.ejb.*;
/**
 * Bean implementation class for Session Bean: HelloEJB
 *
 * @ejb.bean
 * name="HelloEJB"
 * type="Stateless"
 * jndi-name="ejb/com/itso/wrd/ejbs/HelloEJBHome"
 * view-type="remote"
 * transaction-type="Container"
 *
 * @ejb.home
 * remote-class="com.itso.wrd.ejbs.HelloEJBHome"
 *
 * @ejb.interface
 * remote-class="com.itso.wrd.ejbs.HelloEJB"
 *
 */
public class HelloEJBBean implements SessionBean {
   private SessionContext mySessionCtx;
   /**
    * @ejb.interface-method view-type=remote
```

```
     */
    public String getGreeting() {
        return "Hello WebSphere!";
    }
    public SessionContext getSessionContext() {
        return mySessionCtx;
    }
    public void setSessionContext(SessionContext ctx) {
        mySessionCtx = ctx;
    }
    public void ejbCreate() throws CreateException { }
    public void ejbActivate() {}
    public void ejbPassivate() { }
    public void ejbRemove() { }
}
```

After a few seconds, the introduced change is picked up by the rapid deployment tools and the console window displays the contents of Example 17-6 on page 976.

Archived

Example 17-6 Adding the HelloEJBBean.java

```
[11:14:20 AM]   [/Hello/com/itso/wrd/ejbs/HelloEJBBean.java] Added
[11:14:20 AM]   [/Hello/bin/com/itso/wrd/ejbs/HelloEJB.class] copied to project
[HelloEJBClient]
[11:14:20 AM]   [/Hello/bin/com/itso/wrd/ejbs/HelloEJBBean.class] copied to
project [HelloEJB]
[11:14:20 AM]   [/Hello/bin/com/itso/wrd/ejbs/HelloEJBHome.class] copied to
project [HelloEJBClient]
[11:14:28 AM]   Publishing IBMUTC to server_1617843630
[11:14:30 AM]   Installing New Application:  IBMUTC
...
...
[11:15:45 AM]   Publishing HelloApp to server_1617843630
[11:15:46 AM]   Installing New Application:  HelloApp
...
...
[11:16:05 AM]   ADMA5013I: Application HelloApp installed successfully.
[11:16:41 AM]   Installation Completed Sucessfully:  HelloApp
[11:16:41 AM]   Starting Application:  HelloApp
[11:16:42 AM]   Application Started Sucessfully:  HelloApp
[11:16:43 AM]   Publishing HelloApp to server_1617843630
[11:16:43 AM]   Updating Application.
[11:16:44 AM]   Update is not required.
```

When the annotated EJB source code is introduced in the monitored directory, the necessary EJB files such as home and remote interfaces are generated and compiled. They are packaged in an EJB module which is, in turn, packaged into an application called HelloApp. HelloApp is deployed onto the application server.

If the rapid deployment does not find the Universal Test Client (UTC) on the application server, it will be automatically installed. The Universal Test Client can be used for testing for example EJBs and can be accessed at

`http://localhost:9080/UTC`.

If you open the WebSphere administrative console and select **Applications** → **Enterprise Applications** → **HelloApp** you can verify the application configuration as shown in Figure 17-4 on page 977.

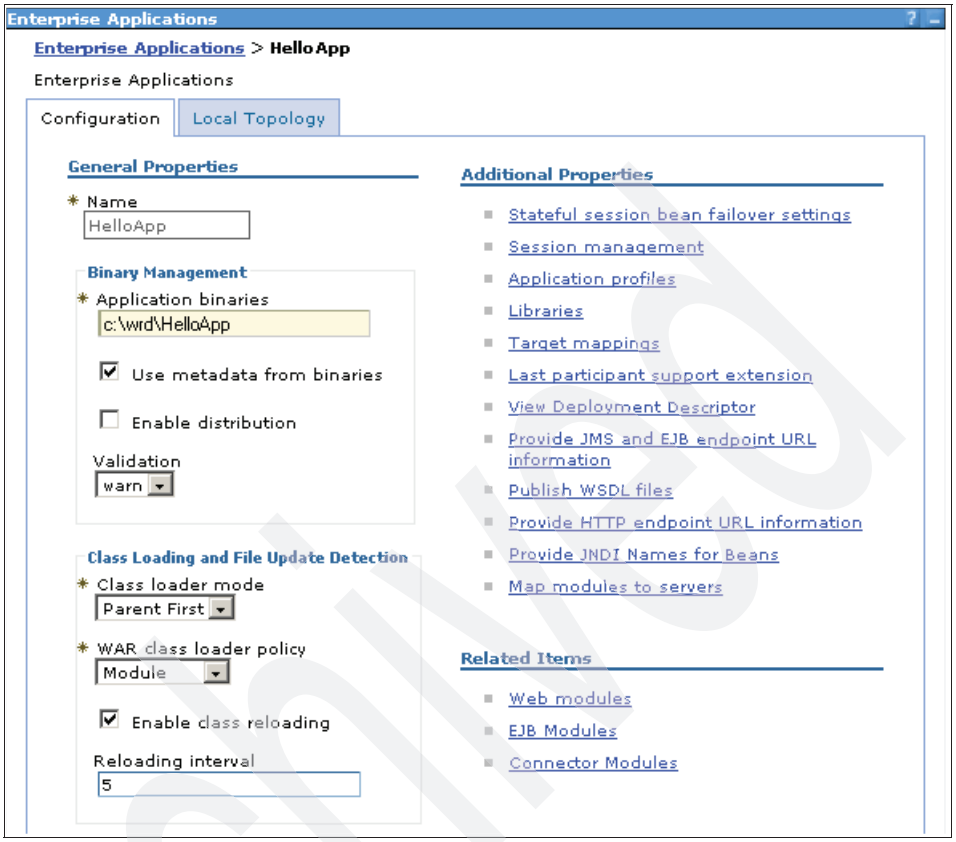


Figure 17-4 HelloApp deployed to WebSphere Application Server

The application resources (binaries) are pointing at the project directory, c:\wrd\HelloApp, indicating that the application server is executing the application using the resources directly from the project directory. This is possible because we are publishing to a local server. If we would have run against a remote server, the rapid deployment tools would have packaged the HelloApp.ear file, distributed it to the remote server and deployed it there. Running against a local server is, of course, faster.

If you click the EJB Modules, you see that the application has one EJB module, HelloEJB.jar, defined. But there are no Web modules yet.

Archived

Note: It is important that the Java source files you add to the project directory are created in the correct directory structure, matching the class files package name. If you simply drop the HelloEJBBean.java class into the c:\wrd\Hello directory instead of the c:\wrd\Hello\com\itso\wrd\ejbs directory, you see a compilation error such as:

```
[05:15:43 PM] 'The declared package does not match the expected package ' in
resource 'HelloEJBBean.java' on line number 1
```

If you are dropping whole EJB modules (ejb.jar files) into a free-form project, you need to generate the deployed code yourself. The EJBDeploy tools do not automatically run on these already packaged ejb.jar files. For information about the EJBDeploy tool, see 16.2.1, “Using EJBDeploy command line tool” on page 928.

If you drop an entity bean, by default the rapid deployment tools will generate a bean-managed persistence (BMP) entry in the deployment descriptor. If you want a container-managed persistence (CMP) bean, specify this using an annotated source file.

A naming convention must be followed for the Java source, compiled class, or annotated source files for each enterprise bean. For example, if an enterprise bean implementation class is named HelloEJBBean.java, follow this convention for any bean interfaces that are defined:

- The remote interface class must be named HelloEJB.java.
- The home interface class must be named HelloEJBHome.java.
- The local interface class must be named HelloEJBLocal.java.
- The local home interface class must be named HelloEJBLocalHome.java.

The primary key class for entity beans can be named anything because the rapid deployment tools logically locate the correct class name by introspecting either the remote home or local home interface.

These rules do not imply that both remote and local view types must be created. These are just naming conventions when creating source for a set of bean interfaces.

A new EJB entry to the deployment descriptor is only added when a minimum set of resources are available in the free-form project. For example, if the bean class is placed in the free-form project, its compiled class maps to the EJB project, but no deployment descriptor entry is created until the rapid deployment tools locate either a set of remote view type classes or local view type classes. For entity beans, a primary key class is also required. If the minimum resources are removed, the bean descriptor entry is removed.

Adding servlet source code

Using the same technique as for the EJB, now add the servlet code by doing the following:

1. Create a directory called `com\itso\wrd\servlets` in the `c:\wrd\Hello` project directory.
2. Add the servlet code shown in Example 17-7 and save the file as `HelloServlet.java` in the `c:\wrd\Hello\com\itso\wrd\servlets` directory:

Example 17-7 Source code for HelloServlet.java

```
package com.itso.wrd.servlets;
import java.io.*;
import javax.naming.*;
import javax.rmi.*;
import javax.servlet.*;
import javax.servlet.http.*;
import com.itso.wrd.ejbs.*;
/**
 * Servlet implementation class for Servlet: HelloServlet
 *
 * @web.servlet name="HelloServlet" display-name="HelloServlet"
 *
 * @web.servlet-mapping url-pattern="/HelloServlet"
 *
 */
public class HelloServlet extends javax.servlet.http.HttpServlet
      implements javax.servlet.Servlet {

   protected void doGet(HttpServletRequest req, HttpServletResponse resp)
      throws ServletException, IOException {
      String greeting = null;
      try {
         InitialContext ctx = new InitialContext();
         Object objectHome =
               ctx.lookup("ejb/com/itso/wrd/ejbs/HelloEJBHome");
         HelloEJBHome helloEJBHome = (HelloEJBHome)
               PortableRemoteObject.narrow(objectHome, HelloEJBHome.class);
         HelloEJB helloEJB = helloEJBHome.create();
         greeting = helloEJB.getGreeting();
      } catch (Exception e) {
         e.printStackTrace(System.out);
      }
      req.setAttribute("greeting", greeting);
      RequestDispatcher dispatch = req.getRequestDispatcher("/hello.jsp");
      dispatch.forward(req, resp);
   }

   protected void doPost(HttpServletRequest req, HttpServletResponse resp)
```

Archived

```
        throws ServletException, IOException {
    doGet(req, resp);
}
```

After a few seconds, rapid deployment tools have picked up the new changes and Example 17-8 is shown in the console window.

Example 17-8 HelloServlet.java added

```
[11:23:21 AM]   [/Hello/com/itso/wrd/servlets/HelloServlet.java] Added
[11:23:21 AM]   [/Hello/bin/com/itso/wrd/servlets/HelloServlet.class] copied to
project [HelloWeb]
[11:23:21 AM]   Publishing HelloApp to server_1617843630
[11:23:22 AM]   Updating Application.
[11:23:23 AM]   Servlet added to web.xml: HelloServlet
[11:23:23 AM]   Servlet mapping added.  URL is: [HelloWeb/HelloServlet]
...
...
```

Using the WebSphere administrative console, you can now see that there is a Web module called HelloWeb.war added to the HelloApp application. The context root for the Web module is the same as its name, HelloWeb. Also, a servlet mapping `HelloWeb/HelloServlet` is added to the web.xml file.

Note: In our very simple example application, our EJB publishes only a remote interface and our servlet looks it up in the JNDI namespace directly without using an EJB reference. We could have used the `@web.ejb-ref` tag if we wanted to use an EJB reference instead of looking it up directly. To access an EJB's local interface, you must use the `@web.ejb-local-ref` tag.

Adding JSP source code

Using the same procedure as before, add the JSP code shown in Example 17-9 and save the file as `hello.jsp` in the `c:\wrd\Hello` directory:

Example 17-9 Source code for hello.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<HEAD>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<META http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<TITLE>Greeting</TITLE>
</HEAD>
<BODY>
<H1><%= request.getAttribute("greeting") %></H1>
```

```
<IMG border="0" src="images/logo.gif" width="211" height="44">
</BODY>
</HTML>
```

When the JSP is added, Example 17-10 on page 981 is shown in the console window.:

Example 17-10 Adding hello.jsp

```
[11:24:58 AM]   [/Hello/hello.jsp] Added
[11:24:58 AM]   [/Hello/hello.jsp] copied to project [HelloWeb]
[11:24:58 AM]   JSP entry added to web.xml: hello
[11:24:58 AM]   Servlet mapping added.  URL is: [HelloWeb/hello]
...
...
```

As you can see in the console window, the web.xml file is updated also with a servlet mapping for the JSP.

Adding a GIF image

Because we want rapid deployment tools to maintain the directory structure for our static resources, we first need to create a directory to hold our image before we can add it to the project.

1. Create a directory called `images` in `c:\wrd\Hello`
2. Copy `logo.gif` to `c:\wrd\Hello\images` directory

The following is shown in the console window:

```
[11:26:05 AM]   [/Hello/images/logo.gif] Added
[11:26:05 AM]   [/Hello/images/logo.gif] copied to project [HelloWeb]
```

17.5.3 Terminating the WebSphere Rapid Deployment session

We have added the source code and resources that make up our example application and can now terminate the WebSphere Rapid Deployment session. Type `Q` and press **Enter** in the console window to do this. This stops monitoring of the project directory.

17.5.4 Verifying results

We are now ready to see what has been generated for us and to test our simple application.

The application we have developed can now be accessed at

`http://localhost:9080/HelloWeb/HelloServlet`.

The output is shown in Figure 17-5 on page 982.

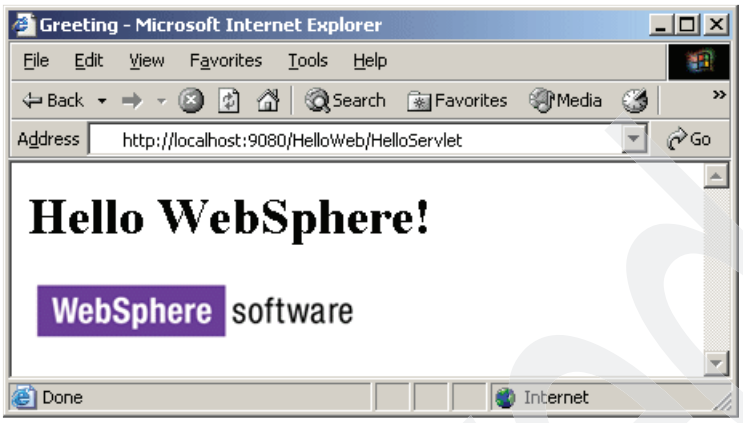


Figure 17-5 HelloServlet output

Note: The rapid deployment tools do not automatically regenerate the HTTP server plug-in configuration file. You must manually regenerate the plug-in configuration file.

In the c:\temp directory, the rapid deployment tools have also produced an EAR file called HelloApp.ear. The contents of this EAR file are shown in Figure 17-6. As you can be see, it uses the default names for the project modules.

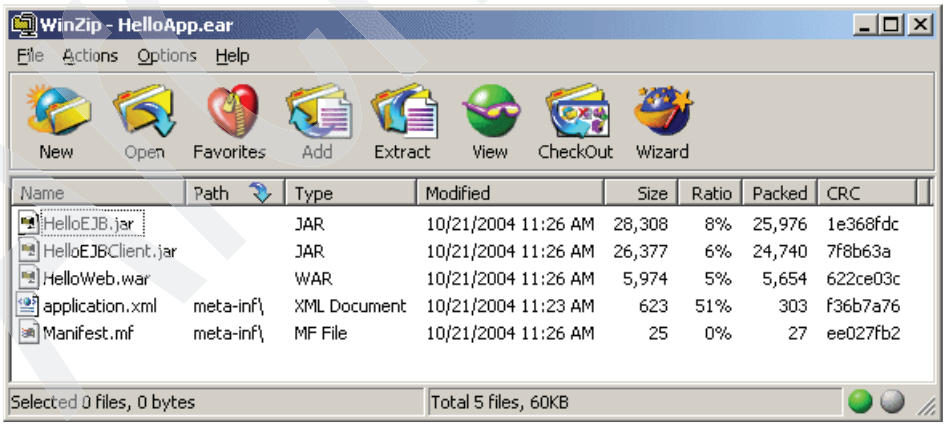


Figure 17-6 HelloApp.ear file produced

17.6 Automatic application installation projects

Automatic application installation mode allows you to set up a directory and have it monitored for either fully composed EAR files or application modules such as WAR files, EJB JAR files, or standalone resource adapter archive (RAR) files.

If you drop EAR files in the monitored directory, the EAR file is automatically deployed to the server. If you delete the EAR file from the monitored directory, that application is uninstalled from the server. If you drop an updated EAR file in the monitored directory, the application is updated on the server. If you drop WAR or EJB JAR files in the monitored directory, the rapid deployment tools generate the necessary EAR wrapper and then publishes that EAR file on the server. For RAR files, a wrapper is not created. The standalone RAR files are published to the server.

The automatic application installation mode only manages your applications, not any resources, such as data sources, they might require. You need to create and manage these resources yourself by using the WebSphere administrative console or wsadmin scripts, before the application can run.

Note: If you drop an EJB module, or an EAR file containing an EJB module, which does not contain EJB deployed code, the rapid deployment tools generate the deployment code with the default EJBDeploy settings, for example, the backend option will be set to DB2UDB_V81. If you want to set a different backend option, run the EJB deployment tools (ejbdeploy.bat) on the EAR file before dropping it into the automatic application installation project. See 16.2.1, “Using EJBDeploy command line tool” on page 928.

To use the automatic application installation mode you must:

1. Create a workspace directory.
2. Configure an `autoappinstall` project in the workspace directory.
3. Configure the rapid deployment tools to monitor the directory for changes.
4. Drop EAR files or application modules into the directory.

We will show how to use the automatic application installation mode using a simple example.

17.7 Automatic application installation example

For the purpose of showing how the automatic application installation mode works, we will use the Hello application we developed in the “Free-form development example” on page 970. In this example, the rapid deployment tools

produced an EAR file, HelloApp.ear, for us. We will use this EAR file in the following scenario:

- Install application
- Update the application
- Uninstall the application

17.7.1 Setting up an automatic application installation session

Using the automatic application installation mode requires you to set up a directory on your system and configure WebSphere to monitor the directory for changes. Do the following:

1. Open a command prompt. This will be used for running the rapid deployment tools in headless, non-graphical, mode. All output from the tools will be written to this window. We therefore also refer to it as the console window).
2. Create a workspace directory, for example `c:\wrd`, by typing:

   ```
   mkdir c:\wrd
   ```

3. Set the WORKSPACE environment variable by typing:

   ```
   set WORKSPACE=c:\wrd
   ```

   Tip: To list all environment variables currently set, you can type **set** in Windows or **export -n** in Linux and press Enter.

4. Configure the WebSphere Rapid Deployment tools to create an automatic application installation project called `install_server1` by typing:

   ```
   cd <profile_home>\bin
   wrd-config.bat -project “install_server1” -style ”autoappinstall”
   ```

   <profile_home> is the root directory of the application server profile.

   This brings up the interactive Parameter Configuration Settings dialog window, Example 17-11. Enter the information as requested:

Example 17-11 Parameter Configuration Settings dialog window

```
------------------------------------------------------------
Parameter Configuration Settings
------------------------------------------------------------
Press ENTER to accept defaults
The * symbol denotes required input

 Enter the server name* ( server1 ) : ITSOCell/ITSONode1/server1
 Enter the server JMX host name* ( localhost ) :
 Enter the server JMX port number* ( 8880 ) : 8879
```

```
 Enter your server username ( -- ) :
 Enter your server password ( -- ) :

Configuring the workspace...
Building the workspace...
WebSphere Rapid Deployment configuration completed.
```

When creating this example, we were running in a distributed server environment and used the JMX port (8879) for our deployment manager. Our target application server is called `server1` and exists on node `ITSONode1` in the cell called `ITSOCell`.

If you were running on a standalone server, you would use the JMX port number (default 8880) for the standalone application server (default `server1`). In our environment, the deployment manager and the application server were located on the same physical machine, so we used the default JMX hostname `localhost`. Also, we did not have WebSphere global security enabled. As a result, we did not need to supply a username or password.

When the **`wrd-config`** command has finished, a directory called `install_server1` is created in the `c:\wrd` directory. The `install_server1` directory is the project directory and is the directory that will be monitored by the tools. In the root of the workspace the `install_server1_headlessconfig.xml` file has been created. This file holds the configuration for our project.

Note: We named our automatic application installation project `install_server1` because the rapid deployment tools then create the `c:\wrd\install_server1` monitored directory for us and this name explains well what the directory is used for. We could have given the project any name, however, because it is only used to name the monitored directory.

The real name of the application server that the application is installed on is configured in the Parameter Configuration Settings dialog window (ITSOCell/ITSONode1/server1).

5. Launch the WebSphere Rapid Deployment session by typing:

   ```
   wrd.bat -monitor
   ```

   This tells the rapid deployment tools to start monitoring the project directory for changes and to display messages on the console.

Note: The **`wrd`** command itself is used to tell the rapid deployment tools to start monitoring the project directory and the (optional) `-monitor` switch enables the console output messages.

The wrd command displaysExample 17-12 in the console window:

Example 17-12 Launching wrd.bat -monitor

```
Launching WebSphere Rapid Deployment.  Please wait...
Starting Workbench...

WebSphere Rapid Deployment ready for e-business...

Type 'q', 'quit', or 'exit' to shut down WebSphere Rapid Deployment processes.
```

To terminate a rapid deployment session, type **Q, quit** or **exit** and press Enter in the console window. This stops the monitoring of the project directory. You can start monitoring the directory by simply running the **wrd** command again.

17.7.2 Managing applications

When the WebSphere Rapid Deployment session is configured and monitoring of the project directory is activated, we can use it for installing, updating and uninstalling our application.

Installing an application

To install an application, simply copy the EAR file to the monitored directory:

- Copy the HelloApp.ear file to the `c:\wrd\install_server1` directory. The application is automatically installed on the application server, as shown in the console output window in Example 17-13.

Example 17-13 Installing the HelloApp.ear

```
[01:09:12 PM]   [/install_server1/HelloApp.ear] Added
[01:09:12 PM]   !INSTALL_EAR_FILE HelloApp.ear!
[01:09:12 PM]   Publishing HelloApp to server_1617843630
[01:09:14 PM]   Installing New Application:  HelloApp
[01:09:18 PM]   ADMA5016I: Installation of HelloApp started.
...
...
[01:09:19 PM]   ADMA5013I: Application HelloApp installed successfully.
[01:09:35 PM]   Installation Completed Sucessfully:  HelloApp
[01:09:37 PM]   Starting Application:  HelloApp
[01:09:38 PM]   Application Started Sucessfully:  HelloApp
```

You can access the application at

`http://localhost:9080/HelloWeb/HelloServlet`.

The output is shown in Figure 17-5 on page 982.

Note: The rapid deployment tools do not automatically regenerate the HTTP server plug-in configuration file when an application has been installed, updated or uninstalled. You must manually regenerate the plug-in configuration file.

Updating an application

To show what happens when an application is updated, we changed the getGreeting method of the HelloEJB to return another greeting message. We then exported the application as a new HelloApp.ear file and copied it into the `c:\wrd\install_server1` directory, replacing the earlier version of the HelloApp.ear file. When the change is picked up by the WebSphere Rapid Deployment tools, Example 17-14 is shown in the console window:

Example 17-14 Updating the HelloEJB

```
[01:26:57 PM]   [/install_server1/HelloApp.ear] Modified
[01:26:57 PM]   !INSTALL_EAR_FILE HelloApp.ear!
[01:26:57 PM]   Publishing HelloApp to server_1617843630
[01:26:58 PM]   Reinstalling Application.
[01:27:18 PM]   ADMA5017I: Uninstallation of HelloApp started.
...
...
[01:27:18 PM]   ADMA5013I: Application HelloApp installed successfully.
[01:27:34 PM]   Application Updated Successfully. HelloApp
[01:27:39 PM]   Starting Application:  HelloApp
[01:27:39 PM]   Application Started Sucessfully:  HelloApp
```

Access the application again at

`http://localhost:9080/HelloWeb/HelloServlet`

See the new greeting message as shown in Figure 17-7 on page 988.

Archive



Figure 17-7 Updated HelloApp running

Uninstalling an application

To uninstall the application, simply delete the EAR file from the project directory. The rapid deployment tools will automatically uninstall it from the application server. The console window shows Example 17-15.

Example 17-15 Uninstalling HelloApp.ear

```
[01:31:50 PM]   [/install_server1/HelloApp.ear] Deleted
[01:31:50 PM]   !DELETE_EAR_WRAPPER HelloApp.ear!
[01:31:50 PM]   Uninstalling HelloApp:server_1617843630
[01:31:59 PM]   ADMA5017I: Uninstallation of HelloApp started.
...
...
[01:31:59 PM]   ADMA5106I: Application HelloApp uninstalled successfully.
[01:32:35 PM]   Application Uninstalled:  HelloApp
```

Archived

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this IBM Redbook.

IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 991. Note that some of the documents referenced here may be available in softcopy only.

- *WebSphere Application Security V6 Security Handbook,* SG24-6316
- *WebSphere Application Server V6 Scalability and Performance Handbook*, SG24-6392
- *WebSphere Application Server V6: High Availability Solutions,* REDP-3971
- *WebSphere Application Server V6 Migration Guide,* SG24-6369
- *WebSphere Application Server V6: Web Services Development and Deployment*, SG24-6461
- *IBM WebSphere Application Server V5.1 System Management and Configuration*, SG24-6195

Other publications

These publications are also relevant as further information sources:

- *Transaction Processing: Concepts and Techniques* (Jim Gray, Andreas Reuter), Elsevier Science & Technology Books, ISBN 1-55860-190-2 1
- *Enterprise Messaging Using JMS and WebSphere* (Kareem Yusuf), Prentice Hall, ISBN: 0-13-146863-4
- *Java Message Service* (Monson-Haefel, Chappell), O’Reilly, ISBN: 0-596-00068-5
- *Professional JMS* (Grant, Kovacs, et al), Wrox Press Inc., ISBN: 1861004931
- *Enterprise JavaBeans, Fourth Edition* (Monson-Haefel, Burke, Labourey), O’Reilly, ISBN: 0-596-00530-X
- *EJB Design Patterns* (Marinescu), Wiley, ISBN: 0471208310

© Copyright IBM Corp. 2005. All rights reserved.

Online resources

These Web sites and URLs are also relevant as further information sources:

- *Service Data Objects*, John Beatty, Stephen Brodsky, Raymond Ellersick, Martin Nally, Rahul Patel

  ftp://www6.software.ibm.com/software/developer/library/j-commonj-sdowmt/Commonj-SDO-Specification-v1.0.doc

- WebSphere Application Server home page

  http://www.ibm.com/software/webservers/appserv/was/

- WebSphere Application Server system requirements

  http://www.ibm.com/software/webservers/appserv/doc/latest/prereq.html

- WebSphere Application Server support (Fix Packs, fixes, and hints and tips)

  http://www.ibm.com/software/webservers/appserv/support.html

- Java Community Process home

  http://www.jcp.org/en/jsr/all

- *Java 2 Platform Enterprise Edition Specification, v1.4*

  http://java.sun.com/j2ee/j2ee-1_4-fr-spec.pdf

- *Java 2 Platform Enterprise Edition, v 1.4 API Specification* at:

  http://java.sun.com/j2ee/1.4/docs/api/index.html

- WebSphere Application Server Information Center

  http://www.ibm.com/software/webservers/appserv/infocenter.html

- *MBeanInspector for WebSphere Application Server*

  http://www.alphaworks.ibm.com/tech/mbeaninspector

- *Sample Scripts for WebSphere Application Server Versions 5 and 6*

  http://www-106.ibm.com/developerworks/websphere/library/samples/SampleScripts.html

- Tcl Developer Xchange

  http://www.tcl.tk/

- IBM WebSphere Developer Technical Journal

  http://www-106.ibm.com/developerworks/websphere/techjournal/

- JDBC Technology

  http://java.sun.com/products/jdbc/index.html

- Enterprise JavaBeans Technology

  http://java.sun.com/products/ejb/

Archived

- J2EE Connector Architecture

  http://java.sun.com/j2ee/connector/
- JavaMail API Specification

  http://java.sun.com/products/javamail/reference/api/index.html
- IBM alphaWorks emerging technologies

  http://www.alphaworks.ibm.com
- IBM developerWorks

  http://www.ibm.com/developerworks/
- Worldwide WebSphere User Group

  http://www.websphere.org
- *An Introduction to Java Stack Traces*

  http://java.sun.com/developer/technicalArticles/Programming/Stacktrace/
- *Apache HTTP Server Log Files*

  http://httpd.apache.org/docs/logs.html
- IBM HTTP Server documentation library

  http://www.ibm.com/software/webservers/httpservers/library/
- *WebSphere MQ Using Java*

  http://www-306.ibm.com/software/integration/mqfamily/library/manualsa/manuals/crosslatest.html
- Java Message Service (JMS)

  http://java.sun.com/products/jms
- *Persistent Client State HTTP Cookies*

  http://home.netscape.com/newsref/std/cookie_spec.html
- XDoclet Attribute Oriented Programming

  http://xdoclet.sourceforge.net/xdoclet/index.html

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Archived

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Archived

Index

Symbols

$ 314
${DB2UNIVERSAL_JDBC_DRIVER_PATH} 921
${DRIVER_PATH} 330
${LOG_ROOT} 423
${SERVER_LOG_ROOT} 423
${USER_INSTALL_ROOT} 423
$AdminApp 275–276
  edit 307
  editInteractive 307
  install 305–306
  installInteractive 305
  list 297–298
  options 305
  uninstall 306
$AdminConfig 274, 276, 281–282, 285
  attributes 285
  create 302, 308
  create Server 304
  createUsingTemplate 314
  defaults 284
  getid 287, 302, 304, 308, 310, 314
  list 284, 307, 310
  listTemplates 314
  modify 302, 308, 310
  parent 283
  queryChanges 308
  remove 305
  reset 290
  save 302, 306
  show 286, 308
  showAttribute 287, 308
$AdminConfig types 282
$AdminControl 274, 277, 293, 296
  queryNames 277–278, 298
  refreshRepositoryEpoch 101–102
  stopServer 294
$AdminTask 275–276, 281, 288
  createApplicationServer 303
  createCluster 291, 310
  createClusterMember 313
  createSIBus 290–291
  deleteClusterMember 313
  deleteServer 304
  exportServer 263
  exportWasprofile 263
  help 289
$Help 275

A

access intent 875, 878, 880
  application profile 881
  policies 878
access intents
  application profiles 210
  tracing 885
activation specification 927
activation.jar 61, 355
ActivationSpec 50
ActivationSpec JavaBean 489, 491, 493–494, 507, 509
Active Server Pages 938
ActiveX to EJB Bridge application client 938, 940
ActivitionSpec 32
activity session service 211
activity.log 421, 449
ActivitySession Service 15
addNode 220–221, 223, 225
admin_host 38, 240
AdminControl 279
administration services 185, 212
administrative console 68
  changing the session timeout for the adminconsole application 165
  home page 166
  logging in 164
  scope 171
  securing 182
  starting 163
administrative console port 126, 133
administrative console secure port 126, 133
administrative service 29, 33
AdminServer 157
AdminService 93–94
AffinityCookie 405
Aged Timeout 337–338

© Copyright IBM Corp. 2005. All rights reserved.

alias destination 602, 634, 668
ALL_AUTHENTICATED 815–816
analyze 449
annotation-based programming 957–958
Apache 915
apache 459
Apache Server 12
apachectl 396
applet application client 938
application
  deploying 930
  editing with wsadmin 307
  exporting 246
  finding the URL 253
  installation 242, 244, 305, 986
  listing installed 297
  multiple updates 948
  preventing from starting 247
  preventing startup 307
  removing files 947
  single file update 946
  single module update 946
  starting 247
  starting order 247
  starting with wsadmin 298
  stopping 247
  stopping with wsadmin 298
  uninstalling 246, 988
  uninstalling with wsadmin 306
  updating 944, 987
  viewing 248
  viewing EJB modules 250
application class loader 827, 832, 839
application classloader policy 207
application client 937
  deployment 937
  launching 941
application client bindings 940
application client container 26, 28
application extensions class loader 841
application module
  updating 945
Application Profiling 15
application profiling service 210
Application Response Time (ARM) agents 36
application server 24, 69
  clustering 24
  creating 192, 303, 910
  logs and trace 912
  modifying with wsadmin 309
  removing 304
  restarting 202
  runtime attributes 203
  starting 197, 295
  stopping 200–201, 296
Application Server Facilities 484
application server profile 114–117, 119–121, 130, 149, 151, 153, 163, 191, 197, 200, 259, 261, 263–264, 271, 316, 389, 971, 984
Application Server Toolkit 10, 17, 76, 78, 456, 850, 856, 858, 864, 903, 909, 928, 930
application.xml 108, 857
application-client.xml 857
applications
  deployment
    dynamic reload 887
    hot deployment 887
  starting and stopping 178
Asynchronous Beans 15
asynchronous beans 320
asynchronous messaging 465–466
attributes 280, 282, 285
authentication 51, 54–55
  component managed 374
  component-managed 333–334, 352
  container 375
  container-managed 333–334
  resource 374
authentication alias 662, 682
authorization 55
auto reload 887
automatic application installation 957, 960, 963, 983–985
automatic file synchronization 953
autoRequestEncoding 888
autoResponseEncoding 888

B

backend ID 245, 866–867, 869, 933
backupConfig 259–260
bean managed activity session 765
bean managed transaction 765–766
bean managed transactions 505
Bean Scripting Framework (BSF) 69, 268
bean-managed transaction 31
binding 860
  application client bindings 940

compound name 774
configured 770
corbaname 774
CorbaObjectNameSpaceBinding 790
data sources 865
EJB JNDI names 861
EJB references 862
EjbNameSpaceBinding 790
IndirectLookupNameSpaceBinding 790
name 771
overriding defaults 930
simple name 773
StringNameSpaceBinding 790
bindings
configured 35
Name bindings 34
bindings connection 552–553
bindings file 936
BMP 877–878
boot class path 184
bootstrap 785, 791–794, 800, 802–804, 808–809, 822
bootstrap class loader 822, 825
bootstrap endpoint 682
bootstrap server 577, 579, 588–589, 616, 628
BOOTSTRAP_ADDRESS 219
BootstrapBasicMessaging 617
bootstrapnoderoot 796
bootstrapped client 588
BootstrapSecureMessaging 617
bootstrapserverroot 796
BootstrapTunneledMessaging 617
BootstrapTunneledSecureMessaging 618
Built-in Mail Provider 61, 355, 357
bus member 47, 595, 600, 603, 644–645, 648, 654
adding to the service integration bus 664

C

cache 210, 721, 738, 754–755, 759, 870, 874
EJB 871–873
cache disk offload 32
cache ID 716–717
cache identifier 715
cache replication 31
cached connection handles 533
cached handles 341, 353
cacheGroups 285
Caching Proxy 10, 78
CCI 343
cci.jar 344
cell 97, 106, 162
definition 23
cell persistent root 770, 776, 779, 781, 784, 790, 794–796, 798–800, 807–808
definition of 776
cellroot 796
CICS 42, 344
class loader 211, 822, 825, 827–828, 831–832, 835, 839
Java 2 class loaders 822
policies 828
WebSphere class loaders 825
RCP directory 826
RE directory 826
RP directory 826
class loader policy 830
class loading 184
class loading mode 207
class path 184
class preloading 833–834
classloader 931
classloader policy 207
ClassNotFoundException 821
Class-Path 838–839
classpath 857
JDBC provider 329
protocol provider 359
resource adapter 348
URL provider 365
client caching 42
client connection 552–554
client/server replication 725
client/server topology 37
client-server topology 727, 733
Cloudscape 12
cluster 17–18, 23–24, 47, 49–50, 64–65, 75, 167, 191, 235, 241, 244, 597, 625, 638, 640, 643–644, 651–652, 654, 755, 870, 872, 909, 932, 941, 944, 951, 953
add a server using wsadmin 313
adding messaging engines 670
create with wsadmin 310
creating 236
definition 24
managing with wsadmin 299
message-driven beans 651
messaging engine 640, 646

Archived

messaging engines 583
restarting servers 202
starting 299
starting and stopping 239
stopping 299
viewing topology 238
workload management 66
cluster.xml 106
CMP 876, 878, 883–884
CMP 2.0 enterprise bean 345
cmpConnectionFactory 345
Collector tool 418, 451
com.ibm.scripting.host 270
com.ibm.websphere.rsadapter 324
com.ibm.ws.rsadapter.cci 324
com.ibm.ws.rsadapter.jdbc 324
com.ibm.ws.rsadapter.spi 324
com.ibm.ws.scripting.connectionType 270
com.ibm.ws.scripting.defaultLang 270
com.ibm.ws.scripting.traceFile 270
com.ibm.ws.scripting.traceString 270
com.sun.jndi.ldap.LdapCtxFactory 773
Common Client Interface (CCI) 343
Common Secure Interoperability 53
communication channel 97
communication settings 212
compiled language debugger 456
completeObject 301
completeObjectName 278–279, 292, 295
component managed authentication 374
component managed authentication alias 574
component-managed authentication 333–334, 352
component-managed authentication alias 556, 663, 924
CompoundClassLoader 839
concurrency control 877
concurrent message consumers 484
configuration ID 283
configuration reload 657, 659
configuration repository 69–70
connection factory 344–345, 472, 572, 579, 581, 641
bindings 931
CMP 332
data source 325
J2C 349, 351
JCA 344
resource adapter 345
WebSphere MQ 552
connection factory bindings 245
connection handles 341
connection management 486
connection management contract 342
connection manager 30
Connection object 475
connection pool 758–759
connection pooling 321, 335, 342
connection proximity 583–584
Connection Timeout 336–337
ConnectionFactory object 475
ConnectionWaitTimeoutException 336–337
connectors
JMX 87
console
See administrative console
container managed activity session 765
container managed authentication 375
container managed transaction 765–766
container managed transactions 504
container-managed authentication 333–334
container-managed authentication alias 334
container-managed persistence 332, 878
See also CMP
container-managed relationship 884
container-managed transaction 31
context root 253–255, 946
cookies 38, 699, 703, 705–708, 715
CORBA 33, 791, 811, 815
naming service groups 817
naming service users 816
URL 791
corbaloc 35, 770, 777, 779, 781, 784–786, 791–792, 795–796, 801–804, 806, 811
corbaname 35, 770, 773–774, 785–787, 789–790, 793, 800–801
core group 67, 185, 193, 219, 223
core group bridge service 213
core group policy 641, 672
core group service 213
correlation ID 449
CosNaming 33–34, 769, 771, 787, 789, 791, 794, 797, 811
CORBA 797
CORBA interface 772
distributed 787
INS 785
JNDI plug-in 794
security 815

CosNamingCreate 815
CosNamingDelete 816
CosNamingRead 815
CosNamingWrite 815
Covalent Enterprise Ready Server 12
cpplication client module 74
createSubcontext 815
createUsingTemplate 314
CSIv2 51, 57–58, 77
CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS 219
custom profile 115, 117, 120, 138, 143, 145–147, 263
custom services settings 185
custom user registry 54
CVS 73

D

data centric 464
data replication service 698, 724–725, 728
Data Replication Service (DRS) 36
data source 59–60, 72, 662, 922, 924
  binding using the AAT 865
  creating 326, 331
  version 4 325
  version 5 323
data source connection factory 849
data sources
  mapping to CMP beans 934
DataAccessFunctionSet 324
database mapping 867
database mapping editor 869
database persistence 721, 742
database reauthentication 340
DataDirect Technologies JDBC Drivers for WebSphere Application Server 10
datagram 468
datasource helper classname 333
DataSource object 59, 321–322
DataStoreHelper 324, 333
DB2 7, 10, 12
DB2UNIVERSAL_JDBC_DRIVER_PATH 918
DDL 244
debug adapter
  JavaScript 456
  WebSphere 456
debugging service 213
default core group 219
default data source mapping 934
default error page 888
default messaging provider 62–63
default node group 129, 219
default profile 117–118, 122–123, 130, 139, 383–384, 409
Default URL Provider 62
default virtual host 208, 245
default_host 240, 257–258
default_host virtual host 37
DefaultCoreGroup 185
DefaultNodeGroup 23
defaults 284
delegation 822, 831, 839, 841
deployment descriptor 97, 107, 109, 344, 858, 860, 865, 870, 875
  application 97
  EJB module
    IBM extensions 885
  viewing 251, 254
deployment manager 10, 23–24, 33, 68–70, 78, 129–130, 133, 143
  starting 155, 163, 187–188, 293
  stopping 187, 189–190, 293
deployment manager node 129
deployment manager profile 115–116, 120–121, 151, 153, 183, 185, 187–188, 191, 264
deployment.xml 108
DES 730
destination 465–466, 473, 602
  configuration 574
destinations 48
destroySubcontext 816
diagnostic trace service 437
diagnostic trace service settings 185
directory browsing 888
discovery address 95
distributed discovery 94
distributed server configuration 34
distributed server environment 82, 84, 92, 106, 119, 378, 382, 936, 945, 950, 972, 985
distributed transaction 484
dmgr 162
  server 779
  starting 163
dmgr_profile_home 104
DNS domain 708
dumpNameSpace 453, 808
durable subscription 471, 493, 529, 532, 547

Archived

dynamic cache 29, 31, 210, 728
dynamic cache replication 36
Dynamic Query 15
dynamic reload 887, 954–955
DynamicCache object type 284–285

E

Edge Side Include caching 32
editInteractive 307
EIS 343–344, 346
eis/datasourcename_CMP 332
EJB
  binding 773
EJB client 66, 798
EJB collaborator 58
EJB container 26, 28, 36, 65–66, 72, 209, 870
EJB Deploy Tool 928
EJB home 34, 770–771, 789, 797–798, 800–801
EJB module 74
  deployment descriptor
    IBM extensions 885
  viewing 250
EJB references
  mapping to beans 935
EJB security collaborator 57
ejbCreate 501
EJBDeploy 929
EJBHome 742
ejb-jar.xml 857–858, 860
EJBLocalObject 767
EjbNameSpaceBinding 799
EJBObject 742, 767
ejbRemove 500, 502
embedded JMS provider 17
embedded JMS Server 637
end of servlet service 735–736, 747–748
Enhanced EAR 17, 74, 847, 891–894, 903, 908, 932
enterprise applications
  extensions sharing session context 889
  See also application
enterprise beans
  isolation level attributes 877
Enterprise Edition management API 85
Enterprise Information Systems (EIS) 341
Enterprise JavaBeans
  using EJBDeploy 928
Enterprise Service Bus (ESB) 17
enterprise services 42
enterprise Web services 41, 43
EntirePool 338
entity beans 870, 872–873, 877–878, 882, 884–885, 917
  caching options 870
environment entry 184, 211
environment variables 909
  See also variables
epoch 103
EventProvider 91
EventType 438
EVERYONE 815–816
exception destination 625–626
Extended JTA Support 15
extensions class loader 822, 825–827, 841
external caching 32

F

factoryClassname 372–373
fail back 642–643, 672, 674, 677
FailingConnectionOnly 338
failover
  critical services 67
  deployment manager 66
  EJB container 66
  hot 724
  HTTP server 66
  JMS messaging 67
  messaging engine 598
  node agent 66
  stateful session beans 17
  Web container 66
federated name space 34, 770–771, 774, 776–777, 779, 781, 784, 788, 798, 808
FFDC 452
file permission mode mask 184
File Serving Enabled 405
file serving servlet 886
file servlet 886
file synchronization service 99, 227
file transfer 70
findByPrimaryKey 880–881, 884
finder method 880
fire and forget 468
First Failure Data Capture
  See FFDC
First Steps 123, 128–130, 134, 136–137, 142, 200

Archived

foreign bus 606–607, 609–612, 627, 634–635, 657, 678, 680, 682–683
  service integration bus 678
  WebSphere MQ 629
foreign bus link 678
foreign destination 603, 692
free-form development 957, 959
free-form project 961–962, 968–970, 978
free-form projects 967
frequency settings 735
Full Resynchronize 103

G

garbage collection 184, 755
generic JMS provider 62–63
generic server 25
GenPluginCfg 408–409
getAttribute 280
getConnection() 322, 325
getid 283
getLocalHost() 96

H

Handle 742
heap size 184, 211, 755, 910
help 181
high availability 50, 66
  messaging 645, 647, 651
  messaging engine 638
high availability domain 67
High Availability Manager 18
high-volume Web sites 758
HomeHandle 742
host alias 239, 257, 915
host aliases 240, 258
hosts file 916
hot deployment 887, 954
hot failover 724
hot standby 67
  transaction log 67
htpasswd 399
HTTP session persistence 36
HTTP session tracker servlet 454
HTTP transport 208
HttpServletRequest
  getHeader 715
  getSession 706, 751
HttpServletResponse
  encodeRedirectURL 709
  encodeURL 709
HttpSession 713, 735–738, 741–742, 745–747, 749–751, 756, 766
  getAttribute 747
  getId 715
  removeAttribute 745
  setAttribute 745, 747

I

IBM HTTP Server 10, 12, 26, 399, 404, 915
  admin password 399
  administration server port 398
  listen ports 917
  logs 440
  NameVirtualHosts 916
  remote administration 397
  Startup errors 917
  version 459
  VirtualHosts 915
ibm.websphere.preload.classes 834
ibm-application-bnd.xmi 108
ibm-application-ext.xmi 108
ibmconfig directory 858
ibm-ejb-access-bean.xmi 858
ibm-ejb-jar-bnd.xmi files 858
ibm-ejb-jar-ext.xmi 858
ibm-partialapp-delete.props 949
IBMSession
  isOverFlow 713
  sync 735–736
IBMTrackerDebug 454
IBMTrackerDebug servlet 454
IIOP 785, 791, 793
  URL 791
IMAP 61, 355–356, 363
immediate stop 201
IMS 42
inbound user ID 680, 684
InboundBasicMessaging 614
InboundBasicMQLink 614
InboundSecureMessaging 614
InboundSecureMQLink 615, 633
index.html 917
indirect foreign bus 637
IndirectLookupNameSpaceBinding 799
InetAddress.getLocalHost() 96
Informix Dynamic Server 12

Archive

initial context 769–771, 791, 794–796, 801, 809
default 799–800
initial context factory 791–793, 797, 800–801, 809
InitialContext 791
in-memory buffer 432, 438
install 306
installed optional packages 834
installedApps directory 109
installInteractive 305
inter-engine authentication alias 659
interface centric 465
Internationalization Service 15
Interoperable Naming Service (INS) 35, 769–770
invoke 281, 294
invoker servlet 887
isolation level 875–877
isolation level attributes 877
IsolationLevelChangeException 877

J

J2C 344, 349
J2C authentication alias 925
J2C authentication data 919
J2C connection factory 349
J2EE 1.4 85
J2EE application client 937
J2EE client 941
J2EE Connector Architecture (JCA) 30
J2EE Management 77
J2EE security 53
J2SE 1.4 85, 456
JAAS 29, 51, 55, 77
JAAS authentication alias 849, 894
JACC 51, 55, 77
Jacl 272–273, 296–297, 302, 306
JAF 77, 355
JAR manifest file 838
Java 2 Platform, Enterprise Edition (J2EE) 4
Java 2 security 29, 51–52, 55, 108
Java 2 Standard Edition (J2SE) 4
Java Activation Framework (JAF) 355
Java and process management settings 184
Java API for XML Messaging (JAXM) 498
Java API for XML Registries (JAXR 13
Java API for XML-based RPC (JAX-RPC) 13
Java Authentication and Authorization Services (JAAS) 52
Java Authorization Contract with Containers (JACC) 18
Java Build Path 854–855
java comp name 813
java comp/env 369
Java Contract for Containers (JACC) 53
Java DataBase Connectivity 321
See also Resource providers JDBC
Java logging interface 456
Java Management Extension (JMX) 78
Java Management Extensions (JMX) 85
Java Native Interface (JNI) 322
Java Secure Socket Extension (JSEE) 53
java.net.URLConnection 364
java.net.URLStreamHandler 364–365
JavaBeans Activation Framework (JAF) 61, 355–356
JavaMail 59–61, 77, 354–357, 360–361
Built-in Mail Provider 61
JavaScript debug adapter 456
JavaServer Faces (JSF) 15
JavaServer Pages 888
javax.ejb.EJBHome 741
javax.ejb.EJBObject 741
javax.ejb.MessageDrivenBean 499
javax.ejb.MessageDrivenContext 503
javax.jms.JMSException 582
javax.jms.MessageListener 500
javax.jms.Queue 850
javax.naming.Context 741
javax.naming.directory 773
javax.naming.ldap 773
javax.naming.ObjectFactory 372
javax.servlet.http.HttpSessionActivationListener 750
javax.servlet.http.HttpSessionAttributeListener 750
javax.servlet.http.HttpSessionListener 750
javax.transaction.UserTransaction 741
javax.xml.messaging.ReqRespListener 500
JAXM 500
JAXP 77
JAXR 41, 77
JAX-RPC 41, 43–44, 48, 77
JCA 12, 341, 343, 349
CCI implementation 60
connection manager 60
resource adapter 61
services 29–30
JCA CCI 323–324
JCA connection manager 323, 343

JCA connector 50
JCA resource adapter 342
JCA resource reference 934
JCA Web services 42
JDBC 77
JDBC driver 59, 918
JDBC provider 59, 662, 849, 920
  configuring with wsadmin 313
  creating 326
JDBC resource provider 59
JDT 456
JMS 8, 12, 17, 77
JMS activation specification 509–510, 515, 518, 545, 547, 549–551, 571
JMS activation specification. See also ActivationSpec JavaBean
JMS administered object 550
JMS administered objects 472, 490, 523, 526
JMS client 49, 576, 580–581, 588, 616
JMS connection 475
JMS connection factory 472, 515, 527, 534, 849, 926
JMS destination 32, 473, 491, 926
  generic JMS provider configuration 574
JMS domains 471
JMS exception 482
JMS message 476–477
JMS message selector 478
JMS provider 59, 62, 471, 514, 926
  default messaging 514, 519, 529, 539, 541, 547, 580
  generic 522–523, 572
  WebSphere MQ 519, 552
JMS queue 850, 926
JMS server 25, 785
JMS session 476
JMX 36, 67, 69, 77, 83, 87, 270, 276–277, 972, 985
  agent 87
  architecture 86
  connectors 87
  enabled management application 87
  ObjectName 92
JMX connector 185
JMX MBeans 58
JNDI 33–34, 64, 72, 75, 245, 771, 789, 797, 799–800, 802–804, 806–809, 811, 813, 815
  APIs 775
  caching 797
  client 791
  Context.list() 810
  EJB Home 789
  initial context 809
  initial context factory 800–801
  javax.naming package 772
  javax.naming.provider.url 791
  JMS 473–474
  objects registered by dmgr server 779
  over CosNaming 771
  provider URL 35, 770
  service provider 800
  using to federate name space 797
JNDI bindings 847
JNI 184, 828, 938
JRas 913
JSESSIONID 38, 705, 707, 716
JSP
  finding the URL 253
JSP precompile 931
JspBatchCompiler 932
JSR 101 41
JSR 109 41, 43, 79
JSR-003 85
JSR-077 35
JSR-109 48
JSR-77 85
JSR-88 907
JTA 77
JTA XAResource API 484
Jython 273

L

Last Participant Support 15
launchClient 941–942
LDAP 51, 54, 77, 773, 797
legacyRoot 779, 781, 784, 799, 808
Light Weight Third Party Authentication (LTPA) 55
lindex 307
listener port 50, 509–510, 554, 568–570, 572, 933
listTemplates 314
Load Balancer 10, 78
load balancing 413
Location Service Daemon (LSD) 96
log 186, 198, 201, 396, 418
  activity 421, 428, 443
  activity.log 429
  HTTP 382
  IBM HTTP Server 440, 442–443

Archived

JVM 420–421, 425–426
merging 449
native 420
native_stderr.log 421, 428
native_stdout.log 421, 428
process 420
service 421, 428, 443
standard 421
startServer 189–190, 198–199
stderr 184
stdout 184
stopServer.log 202
System.err 421
System.out 421
SystemErr.log 420
SystemOut.log 420, 423, 458–459
Web server 401
Web server plug-in 439
Log Analyzer 78, 418, 421, 430, 443, 448, 913
Merging activity logs 449
starting 444
Symptom database 443
Updating the symptom database 450
log and trace settings 186
logs
file formatting 424
file rotation 424
profile creation 128, 134, 142
startServer.log 129, 135
SystemOut.log 129, 135
loopback address 96
loose coupling 464
Lotus Domino Enterprise Server 12
LTPA 51, 55, 77

M

mail from 362
mail provider 356
mail store 362
mail transport 362
mail.jar 61, 355
managed application server 191
managed node 25, 378, 381–382, 386, 388–389
managed objects 85, 91
managed process 82, 772, 776–777, 781
managed server 16, 97
ManagedConnectionFactory 325, 345
manifest 838
MANIFEST.MF 844
master configuration 165
master repository 97
match criteria 642, 675
Max Connections 336–337
maximum failed deliveries 625
maximum in-memory session count 699
Mbean extensions 185
MBean proxy 89
MBean server 88–89
mbeanIdentifier 278
MBeans 86–87, 91, 93–94, 279, 292
server 86
TraceService 301
mbList 277
mediation 50, 606, 659
memory-to-memory replication 699, 723, 725, 728–729
memory-to-memory session persistence 40
message consumer 468, 476, 479, 481, 610, 640
message consumer pattern 468
message consumers 645
message endpoint 488, 494–496
message endpoint proxy 496
message listener 29, 32, 481, 483, 489, 505
message listener service 212
message order 513
message point 603
message producer 468, 476, 478, 605–606, 610, 640
message producer pattern 468
message selector 546
message store 48
message-driven bean 50, 75, 77
message-driven beans 212, 245, 488, 493, 495, 497–502, 504–505, 507, 509–513, 515, 545–549, 551, 568, 572, 605, 640, 649, 651, 848, 861, 927, 933
binding 864
life cycle 501
MessageEndpointFactory 488
messaging bus 46
messaging client 636
messaging engine 47, 49, 581, 595–597, 619, 638, 644, 649
data store 601, 620–621, 623, 660
failover 598
name 600
policy type 642

Archived

preferred server 672
secure communications 619
messaging provider 464, 466
meta-data 285
metadata 111
METHOD_READY state 874–875
Microsoft Internet Information Services 12
Microsoft SQL Server 12
MIME 61, 241, 355
Min Connections 337
missing transaction context 341, 353
monitored directory 960, 983
monitoring policy 212
multi-broker domain 729
multibroker.xml
733–734
multicast 95–96
multicast address 95
multi-row persistent session management 759
multi-row schema 743–744
multi-row session support 757
multithreaded access detection 340
multi-threaded garbage collection 755

N

name bindings 771
name server 34, 770
name service (JNDI) 29
name space 34, 770
viewing contents 453
name space bindings
configuring 811–812
NameService 787, 792, 794, 796
NameServiceCellPersistentRoot 795
NameServiceCellRoot 795
NameServiceNodeRoot 795
NameServiceServerRoot 781, 784, 792–793, 795
namestore.xml 107
naming clients 771
naming service
name bindings 34
nanny process 157
native library path 329
native path 349
native_stderr.log 421, 428
native_stdout.log 421, 428
nboundBasicMQLink 632
nhanced 858
NoClassDefFoundError 825
node
adding 220
See also addNode
clustering 25
definition 23
managing 217
removing 225
See also removeNode
restoring 260
starting 230
stopping 156, 228, 230, 233
synchronization 227
See also syncNode
node agent 33–34, 69, 89, 92, 95–97, 99, 106, 129, 143, 146, 151, 156, 162, 198, 221, 277, 295, 770
definition 70
restarting 233
starting 144, 155, 198, 230, 294
stopping 144, 231, 294
node group 23, 129, 219, 223
node groups 16
node persistent root 770, 776, 781, 784, 790
definition of 776
non-durable subscription 471, 652
nonpersistent message reliability 530
nonpersistent MQ messages 689
non-serializable J2EE objects 741

O

Object Pools 15
object pools 320
Object Request Broker (ORB) 32
Object Request broker (ORB) 29, 32
onMessage 501
operating system security 52
operations 280
optimistic 877
optimistic concurrency 878
OptimisticPredicate 882
Option A EJB caching 870
Option B EJB caching 871
Option C EJB caching 872
Oracle 12
ORB 58
ORB service 185, 211
ORB_LISTENER_ADDRESS 218
outbound user ID 680, 684

Archived

OutboundBasicMQLink 618
OutboundSecureMQLink 618
outputFilename 430
overwrite session management 699

P

parent 282
PARENT_FIRST 831, 836, 839, 841
PARENT_LAST 831, 839, 842
partition ID 716
partitioned queues 639
passivation 209, 874
peer-to-peer mode 724
peer-to-peer replication 725
peer-to-peer topology 37, 725–726, 733
peferred servers only 675
Performance Monitoring Infrastructure (PMI) 35
Performance Monitoring Instrumentation (PMI) 78
Performance Monitoring Interface (PMI) 29
performance monitoring service 206, 212
persistence manager 324, 875, 880, 883, 885
Persistence Resource Adapter 324
persistent area 34
persistent message reliability 530
persistent MQ messages 689
persistent partition 770
persistent session 711, 713, 741
persistent session database 757
persistent store 710, 870
pessimistic 877
PessimisticUpdate 880
pluggable application client 938
pluggable authentication module 54
plug-in configuration file
  automated propagation 412
  automatic regeneration 410
  propagating 411
  regenerating 406, 408
  viewing 407
plugin-cfg.xml 404
PMI request metrics 36
PMI service 35
Point-to-Point domain 482
Point-to-Point messaging model 466–467, 514, 519, 610
poison message 513
pooled connection 59
POP3 61, 355–356, 363
port
  admin_host virtual host 38
  bootstrap 785, 788, 791, 804, 809
  CELL_DISCOVERY_ADDRESS 95
  default host 37
  NODE_DISCOVERY_ADDRESS 95
  NODE_MULTICAST_DISCOVERY_ADDRESS 95
  SSL 240
pre-compile JSP 932
prefer local 237
preferred server 642, 647, 672, 677
preferred servers only 642–643, 647, 672, 674
Principals 55
print 425
print() 421
println() 421, 425
printStackTrace() 421
Private UDDI Registry 79
process definition 211
process execution 184, 211
process group assignment 184
processor partitioning 184
product information
  viewing 206, 457
profile 16
  deleting 263
  exporting and importing 262
Profile creation wizard 121, 123, 128–130, 134, 136, 138, 142, 151, 153–154, 158, 191
profile registry 151
profile_home 104, 118
ProfileCreator 121
profileRegistry.xml 151
profiles
  about 114
  creating 121
  types 116
Programming Model Extensions 15, 29
programming model extensions 210, 320
project directory 960
protocol adapters 87
  JMX 87
protocol provider 61, 354, 359
protocol switch rewriting 710
provider endpoint 578, 628, 641, 682
provider URL 789–790, 793
  javax.naming.provider.url 791
pseudo-synchronous messaging 469

publication point 604
publish/subscribe broker profile 633
publish/subscribe domain 471, 482
Publish/Subscribe messaging model 466–467, 514, 519, 612
publish/subscribe profile 689
pull mode 468, 480
Purge Policy 338
push mode 468, 480
Python 273

Q

quality of service 50
queryNames 277–278, 292
queue destination 515, 535, 537, 539, 554, 562–563, 602–603
  creating 666
queue destinations 561
queue manager 611, 616, 630, 632, 634, 683, 689
queue point 603
QueueConnectionFactory object 474

R

ra.xml 344, 349, 857
Rational Application Developer 10, 17, 43, 73–75
Rational ClearCase 73
Rational Rose 73
Rational Web Developer 10, 75
Rational XDE 73
RCP directory 826
RE directory 826
read ahead 539
read-ahead 883–884
Reap Time 337–338
Reap Timeout 337–338
receiver channel 631–632, 688–689
Redbooks Web site 991
  Contact us xxi
referenceable 64, 370–371, 373
RegenerateKey 730
relational resource adapter 60, 323–324, 333
reloadingEnabled 955
removeNode 225–226
replica 731
replication client 727
replication domain 237, 726, 728–730
replication entry 238
replication mode 733
replication server 727
replicationType 286
replicator entry 728–729
repository 84, 95–97, 104–106
  application data 107
  application execution 109
  saving work to 180
request routing 379
request-reply 468
request-reply pattern 469
reset 290
resource adapter 59, 74, 343–346, 349, 488, 649, 827
  deployment descriptor 489
  installation 345
  lifecycle management 486
  message inflow management 487
  packaging 489
  service integration bus 514, 517, 649
  transaction inflow management 487
  WebSphere Relational Resource Adapter 345
  work management 486
Resource Adapter Archive (RAR) 344
resource environment entry 64
resource environment provider 59, 64, 370–371
resource provider
  J2C 341, 343, 345, 349
  JavaMail 356
  JDBC 326, 331
  URL 364, 368
    Configuring URLs 366
resource providers 58
resource references 935
resources.xml 326, 345, 370
res-sharing-scope 341
restoreConfig 259–260
ripplestart 239
RMI 218, 222, 270
RMI connector 87
RMI/IIOP 66, 222
Rollout Update 25, 944, 951–953
rollout update 244
round robin 414
round robin routing policy 66
RP directory 826

S

SAAJ 77

Archived

sample applications 137
SAP 344
SAS interceptor 58
scalability 50
Scheduler Service (Timer Service) 15
schedulers 320
scope 107, 171–172, 176
security 18, 164, 182
  console 85
  session 751
  session management 751
  Web services 41
  WebSphere Application Server 29, 37, 51, 56–57, 77
  WebSphere UDDI Registry 44
security collaborator 57
security contract 343
security management 486
security roles 246, 935
security server 56
sender channel 630–632, 688–689
serialize session access 699
serve servlets by class name 887
server
  starting 156
  status 156
  stopping 156
server ID 717
server root 770, 790
server root context 792–795, 798, 809
server weight 237
server weighted routing policy 66
server.xml 221
ServerCloneID 415
serverindex.xml 95, 97
serverStatus 135, 156, 163, 194–195
Service Data Object (SDO) 77
Service Data Objects (SDO) 14
service integration 17
service integration bus 17, 45–46, 48–50, 106, 149, 168, 212, 218, 221, 223, 515, 577, 594, 849, 925
  architecture 594
  bus member. See bus member
  clustering 638
  configuration 655
  connecting to 576
  controlling messaging engine selection 579
  creating 658
  creating with wsadmin 290
  data store 620
  destination 602, 667
  exception destination 626
  foreign bus. See foreign bus
  foreign link 680
  JMS activation specification 509, 518, 545–546
  JMS connection factory 515, 527–528
  JMS destination 536
  JMS queue 537, 540
  JMS topic 542
  link 627–629
  load balancing bootstrapped clients 588
  mediation 606
  message-driven beans 649, 651
  messaging engine. See messaging engine
  quality of service 530
  reliability 604, 606
  resource adapter 509, 514, 516, 518
  runtime components 612
  scalability 600
  security 652
  SIB service. See SIB service
  topic destination 543
  topologies 645–646
  transport chain 614
  WebSphere MQ addressing 633–634
  XA recovery 533
service integration bus queue 850
service provider 355
Service Provider Interface (SPI) 343, 772
Servlet 2.2 API 718
servlet caching 208
servlets
  finding the URL 253
  serve by class name 887
servlets by class name 405
session
  performance 752
session administrative object 356
session affinity 698, 703, 705, 715, 755
session beans 873
  caching options 873
  stateful 874
  stateful EJB timeout 874
session cache 710
session cleanup settings 738
session context 701
session ID 702, 705, 710, 715–716
session identifier 706, 709

session invalidation time 738
session management 27, 38, 208
  affinity 715, 717–718
  cleanup schedule 749
  DB2 page sizes 742
  HTTP 698
  invalidating sessions 749, 755
  JSESSIONID 38
  last access time 735
  local 710
  maximum in-memory session count 712
  multi-row schemas 743
  overflow 713
  overflow cache 711–712, 754
  persistent 719–721, 734
  properties 699–700
  row type 743
  security ID 751
  security integration 714
  serializable requirements 741
  serialize session access 714
  session affinity 717
  session cache size 754
  session listeners 749
  session object size 753
  session size 753
  session timeout 755
  session tracking mechanism 703
  single-row schemas 743
  single-row to multi-row migration 744
  SSL ID tracking 704–705
  SSL session identifiers 38
  time-based write frequency 737
  write contents 745–746
  URL rewriting
  See also cookies
session management properties
  application 700
  application server 700, 761–762
  Overwrite Session Management 701
  Web module 700
session manager 711, 724, 728, 731, 751, 755, 757–759
  overflow 711
session object size 753
session persistence 39, 698, 705
session scope 700
session store 717
session write interval 737
SessionBeanTimeoutException 875
SESSIONS table 743
setAttribute 280, 300–301
setCharacterEncoding 889
setMessageDrivenContext 500, 502
setupCmdLine 221
shared libraries 834, 842
shared library class loader 832
shared session context 701
showall 288
showAttribute 287
Showlog 421
showlog 430
SIB JMS Resource Adapter 517, 649
SIB service 577, 612–615, 641, 648, 655–657, 682
sib.client.ssl.properties 619
SIB_ENDPOINT_ADDRESS 578, 614–615, 656
SIB_ENDPOINT_SECURE_ADDRESS 614–615, 656
SIB_MQ_ENDPOINT_ ADDRESS 614
SIB_MQ_ENDPOINT_ SECURE_ADDRESS 615
SIB_MQ_ENDPOINT_ADDRESS 615, 689
SIB_MQ_ENDPOINT_SECURE_ADDRESS 615
sibDDLGenerator 623, 662
Simple WebSphere Authentication Mechanism 751
Simple WebSphere Authentication Mechanism (SWAM) 54, 77
single sign-on 55
Singletons 833
SMTP 61, 355–356
SNMP 67
snoop 258
SOAP 79, 222
SOAP connector por 212
SOAP connector port 101, 126, 133
SOAP with Attachments API for Java (SAAJ) 13, 41
SOAP_CONNECTOR_ADDRESS 217, 220, 222
SOAP_CONNECTOR_PORT 218
special header 414
SPI 324, 343, 772
spilling 601
SQL92 conditional expression syntax 478
SSL 44, 216, 619, 629, 631, 705
SSL ID tracking 699
SSL session ID 703, 705
SSL session identifiers 38
SSLV3TIMEOUT 705
standalone application server 191
standalone server environment 83, 106, 119, 384

Standard Java security 52
startApplication 298
startManager 130, 155, 160, 163, 188, 190
startNode 144, 155, 198
startServer 136, 156, 198, 200, 202
startserver 396
startServer.log 189–190, 198–199
Startup Beans 15
stateful 873
stateful session bean failover 698, 729, 760–762, 764
stateful session EJB persistence 36, 40
StateManageable 91
statement cache size 340
StatisticsProvider 91
stderr 420, 423, 428
stdout 420, 423, 428
sticky bean managed unit of work 766
stopApplication 298
stopManager 190
stopNode 144, 156, 228, 231–232
stopServer 137, 156, 201–203, 293
stopServer.log 202
storeUsingOCC 882
stream 71
Stream Handler Class 365
Subject class 55
subscription durability 547
Sun Java System Web Server 12
sun.misc.Launcher$AppClassLoader 822
SWAM 51, 55, 751
Sybase Adaptive Server Enterprise 12
symptom database 450
Symptom database *See* Log Analyzer
synchronization 70, 98, 100, 227–228, 389
  forced 99
  scheduling 99
synchronize 97, 103
synchronous messaging 465
syncNode 103, 228–229
system class loader 822, 825
System.err 420–421, 423, 425, 427
System.out 420–421, 423–425, 427
SystemErr.log 420
SystemOut.log 155, 301, 420, 423, 458–459

T

target groups 585
target mappings 247
target server 402, 960
template 147, 153, 192, 314
  application server (creating) 194
  Web server 393
thin application client 938
thread
  ID 427
thread pool 213
tightly coupled 464
time-based write 737
time-based writes 737
timeout
  aged 337–338
  connection 336–337
  ConnectionWaitTimeoutException 336
  reap 337–338
  session 713, 749, 755
  unused 337
Tivoli Access Manager 53
Tivoli Access Manager (TAM) 18
Tivoli Access Manager Servers for WebSphere Application Server 11
Tivoli Directory Server for WebSphere Application Server 11
Tivoli Performance Viewer 36, 78
topic destination 515, 535–538, 541, 555, 561–562, 566
topic mappings 633
topic space destination 602–603
  creating 668
topic space mappings 629, 682, 689
topic subscriber 471
trace 418, 430–431, 437, 913
  enabling using wsadmin 300
  output 438
  starting for a running server 437
  strings 433
  viewing output 438
  Web server plug-in 439
trace specification string 433
trace string 437
TraceService 300–301
transaction
  bean managed 505
  commit 870
  container managed 504
  isolation level 876–877
  message-driven beans 502

Archived

viewing 205
transaction management 29, 486
transaction management contract 342
transaction service 30, 205, 210
transient area 34
transient partition 770
transport chain 26, 186, 208, 216, 578, 583, 612, 614–616, 619, 629, 631–632, 659, 688
TRIPLE_DES 730
types 282

U

UDDI 41, 46
UDDI Registry 44
UnauthorizedSessionRequestException 751
Unified Modeling Language (UML) 73
unique ID 654
unit of work 447, 449, 765
Universal Description, Discovery and Integration (UDDI) 13
Universal Test Client 976
unmanaged node 378, 381, 386, 389
unmanaged Web server node 26
UriGroup Name 405
URL provider 59, 62, 364, 368
URL rewriting 38, 698–699, 703, 705, 709–710
user registry 53
user rights 127, 133
utility JAR 838–840

V

V5 default messaging provider 63
variable 179–180, 909–910, 918
variables 330
variables.xml 107
versionInfo 458
virtual host 75, 405, 901, 903, 917, 935
  admin_host 38, 240
  and Web server plug-in 241
  architectural overview 239
  architecture 37
  binding 931
  creating 240
  creating with wsadmin 308
  default_host 37, 240, 257–258
  example 914–915
  finding the URL for a servlet or JSP 253, 256–257
  host alias 257
  host aliases 240, 258
  IBM HTTP Server 916
  in a cluster 241
  managing 239
  map to Web modules 246
  mapping 256
  MIME settings 241
  modifying with wsadmin 309
  scope 240
virtual hosting 915
Virtual hosts 915
  See also IBM HTTP Server VirtualHosts

W

WAR class loader 827, 832, 839
WAR classloader 837
was.policy 108, 931
was_home 117
WAS_USER_SCRIPT 383
waslogbr 444
wasprofile 118, 151–153, 263–264
WASService 157–159, 194
WC_defaulthost 914
Web container 26, 57, 64–66, 71, 207, 889
  See also J2EE Web container
  overview 26
Web container inbound chain 27
Web container inbound transport chain 379
Web container transport chain 186
Web module 74–75, 886, 889
  auto reload 887
  default error page 888
  directory browsing 888
  file serving servlet 886
  serve servlets by class name 887
Web security collaborator 57
Web server 16, 25, 57, 932, 945, 950
  adding 392
  configuration file 400
  logs 401
  logs and traces 439
  starting and stopping 395
  viewing status 394
Web server plug-in 10, 26–27, 32, 57, 64–65, 71, 75, 378–380, 945, 951
  overview 27
  regenerating 406

Archived

request routing 412
Web server plugin 27, 240–241, 293, 715, 718–719, 755
generating 409
log 439
regenerating with wsadmin 300
trace 439
Web server plug-in configuration service 410
Web service client 43
Web service endpoint 48
Web service provider 43
Web services 8, 50
WebSphere Application Server support 40
Web Services Description Language (WSDL) 40
Web services engine 27
Web Services Explorer 43
Web services for J2EE (JSR 109) 79
Web services for Java 2 Platform, Enterprise Edition 13
Web Services Gateway 14, 17, 44, 51, 79
Web Services Security (WS-Security) 13
Web Services-Interoperability (WS-I) Basic Profile 13
web.xml 369, 857, 860
WebSphere
security server 56
Version 457
WebSphere Application Server 9
WebSphere Application Server - Express 8
architecture 20
WebSphere Application Server Enterprise 320
WebSphere Application Server Network Deployment 9
architecture 21
WebSphere Business Integration Event Broker 519
WebSphere Business Integration Message Broker 519
WebSphere Business Integration Server Foundation 8, 320
WebSphere Enhanced EAR 74
WebSphere MQ 8, 17, 49, 62–63, 519, 608, 633
link 630
WebSphere MQ client 634
WebSphere MQ client link 637
WebSphere MQ connection factory 552, 558, 571
WebSphere MQ JMS provider 62–63
WebSphere MQ link 630–635, 685
WebSphere Rapid Deployment 17, 75–77, 242, 957, 959–960, 967, 970, 973–974, 981, 985–986
WebSphere Relational Resource Adapter 62, 345
WebSphere Studio 42
WebSphere UDDI Registry 44
Windows service 126, 133, 150, 157, 159, 194, 223
WLM 698, 870
work area partition service 211
work area service 211
WorkArea Service 15
working directory 911
workload management 64, 67, 760
EJS WLM 870
messaging 644–645, 647–648, 651
messaging engine 638, 644
WebSphere Application Server 78
workspace 168
wrd 962, 966, 985
wrd-config 962–966, 972, 985
ws.ext.dirs 826–827
wsadmin 68, 268
definition 268
getid 283
help 269
interactive 271
list 284
profile 271, 273
properties 270, 274
script files 272
show 286
showattribute 287
starting session 268
WSCallerHelper 324
WSDL 44–46, 79
WS-I Basic Profile 79
WSIF 42, 44
WSIL 79
wsinstance 114
wsOptimisticRead 879–880
wsOptimisticUpdate 879
wsPessimisticRead 879
wsPessimisticUpdate 879
wsPessimisticUpdate-NoCollision 883
wsPessimisticUpdate-WeakestLockAtLoad 881
WS-Security 41

X

XAResource 30
XAResources 31
XDoclet 959

Xdoclet 957

Archived

Archived

WebSphere Application Server V6: System Management and

(1.5" spine)
1.5"<-> 1.998"
789 <->1051 pages

Archive



WebSphere Application Server V6 System Management and Configuration Handbook



Read this book and others in the WebSphere Handbook Series

Learn to design and administer your own system

Customize profiles, scripts and applications

This IBM Redbook provides system administrators, developers, and architects with the knowledge to configure a WebSphere Application Server V6 runtime environment, to package and deploy Web applications, and to perform ongoing management of the WebSphere environment.

One in a series of handbooks, the entire series is designed to give you in-depth information about the entire range of WebSphere Application Server products. In this book, we provide a detailed exploration of the WebSphere Application Server V6 runtime environments and administration process.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-6451-00 ISBN 0738492019

Archive