WebSphere Application Server Enterprise Services

JMS Listener

Note!

Before using this information and the product it supports, be sure to read the general information under “Notices”on page 22 .

Contents

Extended messaging service concept articles

This part contains concept topics about the extended messaging service provided by WebSphere Application Server 4.0 enterprise services. These topics are intended to provide background information that you should understand to be able to complete tasks to enable and use the extended messaging service.

An overview of the extended messaging service

WebSphere enterprise applications can use the extended messaging service to automatically receive messages from input queues (JMS destinations) and to coordinate the processing of those messages. This enables automatic asynchronous delivery of messages to an enterprise application, instead of the application having to explicitly poll for messages on the queue. The extended messaging service is shown in the figure Figure 1.

The extended messaging service comprises a listener manager, which controls and monitors one or more JMS listeners. Each JMS listener monitors an input queue (a JMS destination) for incoming messages. When a message arrives on the input queue, the JMS listener passes the message to a user-developed message bean (EJB) for processing. You are recommended to develop a message bean to delegate the business processing of incoming messages to another EJB, to provide clear separation of message handling and business processing. This also enables the business processing to be invoked by either the arrival of incoming messages or, for example, from a WebSphere EJB client. Responses can be handled by another EJB acting as a sender bean, or handled in the message bean. Optionally, this process can happen within the scope of a transaction.

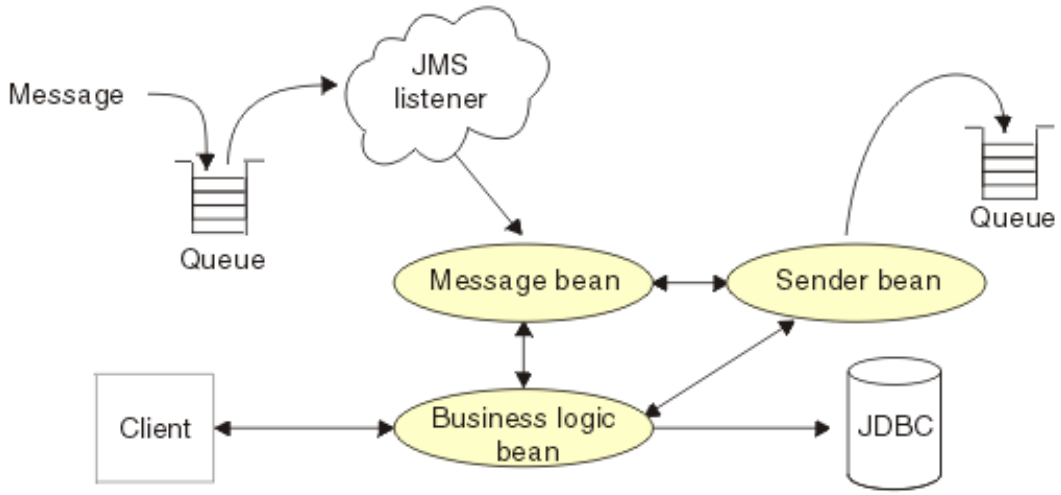

Figure 1 of 4. Message handling by the extended messaging service

The operation of the extended messaging service is controlled by an XML configuration file, which you edit to specify properties of the service; for example, the identities of JMS destinations that are to be monitored, the transactional requirements, and the name of the Enterprise JavaBean to process messages. When an application server is started, it initializes the JMS listener manager based on the enterprise services XML configuration file. The listener manager creates a dynamic session thread pool for use by JMS listeners, creates and starts JMS listeners based upon the XML configuration file, and during server termination controls the cleanup of resources. Each JMS listener completes several steps for the JMS destination that it is to monitor, including:

- Creating a JMS server session pool, and allocating JMS server sessions and session threads for incoming messages.
- Interfacing with JMS ASF to create JMS connection consumers to listen for incoming messages.
- If specified, starting a transaction and requesting that it is committed (or rolled back) when the EJB method has completed.
- Processing incoming messages by invoking the onMessage() method of the specified

EJB.

You can update the configuration file at any time, but if it is updated, the updates only take effect when the server is next started.

The message bean is implemented as a stateless session bean. In a future version of WebSphere, the message bean is to be re-implemented as a J2EE message-driven bean.

The extended messaging programming model

Messaging is evolving to support message-driven beans, as defined within the EJB 2.0 specification. Therefore, you are recommended to develop beans that handle messages in keeping with the latest draft EJB 2.0 specification.

A JMS listener invokes an instance of the stateless session bean defined for the destination in the configuration file. This bean must have an onMessage() method, which should conform to the following rules:

- The method must be declared as **public**.
- The method must *not* be declared as **final** or **static**.
- The return type must be **void**.
- The method must have a single argument of type `javax.jms.Message`.
- The throws clause must *not* define any application exceptions.

For example, a basic onMessage() method for a message bean is shown in the following code extract:

```
public void onMessage(javax.jms.Message msg)
{
 try
   {
     System.out.println("MySessBean.onMessage(), msg text:
"+((TextMessage)msg).getText());
   }
 catch  (Exception err)
 {
    err.printStackTrace();
 }
}
```

Figure 2 of 4. Code example: The onMessage() method of a message bean. This figure shows a code extract for a basic onMessage() method of a message bean. The method extracts the text from an incoming message and sends that text to System.out.

You should also consider the function provided in the ejbCreate() and ejbRemove() methods, for which you are recommended to follow the Message DrivenBeans requirements:

- The method must be declared as **public**.
- The method must *not* be declared as **final** or **static**.
- The return type must be **void**.
- The method must have no arguments.
- The throws clause must *not* define any application exceptions.

Extended messaging, transaction support

The extended messaging service can handle messages read from JMS destinations within the scope of a transaction. If transaction handling is specified for a JMS destination, the JMS Listener starts a global transaction *before* it reads any incoming JMS message from that destination. When the message bean processing has finished, the JMS Listener commits or rolls back (using JTA transaction control) the transaction.

To configure WebSphere to provide transactional support for the extended messaging

service, you complete the following stages:

1. Use the JMSAdmin tool to configure the WAS name space to specify that you are using WebSphere JMS/XA connection factories.

   Note: To enable support for global transactions, you must use the WebSphere JMS/XA connection factories, because these ensure proper transaction behavior.

2. Edit the extended messaging configuration file to specify the transactional requirements separately for each JMS destination.
3. Use the application assembly tool to configure the onMessage() method of the message bean to support the transactional behavior that you want. For future compatibility with the EJB2.0 specification, message beans specify either bean-managed transactions (BMT) or container-managed transactions (CMT) with a transaction behavior of TX_NOT_SUPPORTED or TX_REQUIRED.

Notes:

1. All messages retrieved from a specific destination have the same transactional behavior.
2. To enable the transactional behavior that you want, you must specify the same transaction behavior in the extended messaging configuration file as you configure for the onMessage() method of the message bean.
3. If a destination does not need global transaction support, you can configure the WAS name space using standard non-XA JMSConnectionFactories.
4. If you use non-WebSphere XA connection factories, JMS resources are not recovered if the server fails.

If messages are queued to be sent within a global transaction they are sent when the transaction is committed. If the processing of a message causes the transaction to be rolled back, then the message that caused the bean instance to be invoked is left on the JMS destination.

You can use the **MaxRetries** configuration option for the destination to define the maximum number of times the JMS Listener attempts to read a message from the destination. When MaxRetries has been reached, the JMSListener for that destination is stopped. When you have resolved the problem, you must then stop and restart the given message bean's EJB module from Systems Management.

Extended messaging, security support

Messages arriving at destination being processed by the JMS Listener have no client credentials associated with them. The messages are anonymous. However, some security can be provided if a JMS listener assumes the EJB server process credentials when invoking the message bean.

To enable this security support, when you configure security for your application, you must configure the following elements:

- Resource security for the message bean to allocate its security method groups.
- Security permissions for the bean create() and onMessage() methods so that they can be accessed by the given EJB server principal.

For more information about configuring security for your application, see “5.1.3.1: Securing resources and applications” in the WebSphere Application Server Advanced Edition infocenter. .

Extended messaging, connection pooling

To improve the overall performance of JMS within the system, the extended messaging service enables the connection pooling facility provided by the MQSeries JMS implementation. This support does not affect the performance of the Listener, because it retains its connections while listening on destinations, but does affect the overall JMS system performance. When a connection is no longer required, instead of destroying it, it can be pooled and later reused.

You can use the following two options of the extended messaging service configuration file to configure the corresponding characteristics of the MQSeries connection pool:

Timeout

Defines the number of milliseconds an unused connection is retained within the pool.

Threshold

Defines the maximum number of unused connections retained within the pool.

WebSphere Application Server cloning and MQSeries clustering

This topic provides a summary of information about using WebSphere Application Server horizontal cloning with MQSeries server clustering support. It describes a scenario that shows how the extended messaging service can be configured to take advantage of MQSeries server clustering and provides some information about how to resolve potential runtime failures in the clustering scenario. The information in this topic is based on the scenario shown in Figure 3.

For a WebSphere application server configured to use the extended messaging service, each JMS listener is used to retrieve messages from destinations defined to the server. In the following information the listener configurations are the same for each WebSphere application server. Each application server host contains a WebSphere application server and an MQSeries server. If a host is only used to distribute messages, it only contains an MQSeries server. There can be many servers defined in the configuration, although for simplicity the information in this topic is based on a scenario containing only three servers as shown in Figure 3.

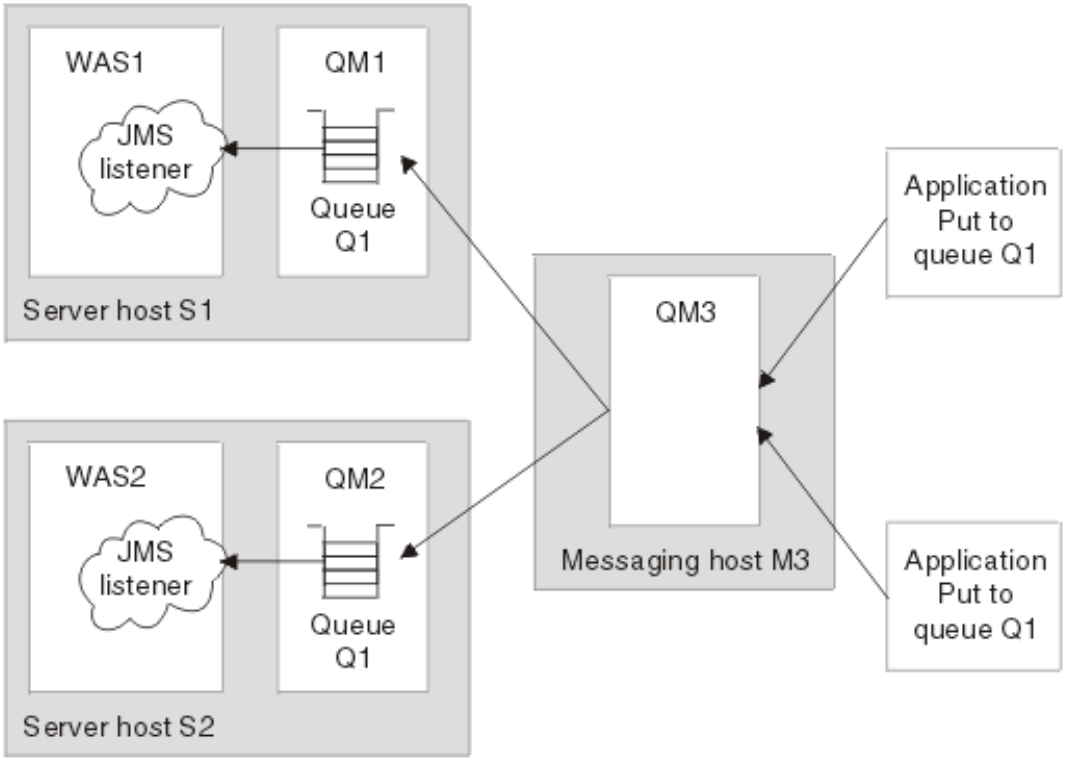

Figure 3 of 4. WebSphere Application Server horizontal cloning with MQSeries clustered queues

The scenario shown in Figure 3 comprises the following three hosts:

- Server host S1 contains the following servers:
  MQSeries server.

  The server is defined to have a queue manager, QM1, and a local queue, Q1. The queue manager belongs to a cluster. The queue is populated by the MQSeries server located on host M3. Applications can add messages directly to the queue, Q1, but would not be subjected to the control of the MQSeries cluster.

  WebSphere Application Server

  This contains a cloned application server, WAS1, which is configured with a JMS listener. The listener is configured to retrieve messages from JMS destination Q1.

- Server host S2 contains the following servers:
  MQSeries server.

  The server is defined to have a queue manager, QM2, and a local queue, Q1. The queue manager belongs to the same cluster as QM1 on host S1. As with QM1, the queue is populated by the MQSeries server located on host M3. Applications can add messages directly to the queue, Q1, but would not be subjected to the control of the MQ cluster.

  WebSphere Application Server

  This contains a cloned application server, WAS2 (identical to WAS1 on host S1), which is configured with a JMS listener. The listener is configured to retrieve messages from JMS destination Q1.

- Messaging host M3 contains the following servers:

MQSeries server.

The server is defined to have a queue manager, QM3, which also belongs to the same cluster as QM1 and QM2. Applications add messages to the queue manager and queue Q1. The cluster to which this queue manager belongs causes messages to be distributed to all other queue managers in the cluster which have queue Q1 defined.

Note: Queue Q1 is not defined as a local queue on this host. If the queue was defined locally, then messages would remain on the server for local processing; messages would not be distributed by the queue manager cluster control to the other queue managers in the cluster that do have the queue defined.

This host does not have a WebSphere application server defined. All message retrieval processing is performed by the other two application servers on hosts S1 and S2.

Recovery scenarios

There are several failure scenarios that could occur with the clustering configuration; for example:

- WAS server failures.

  In this scenario the failure of any single WebSphere application server results in the messages for the specified destination remaining on the queue, until the server is restarted. There is no automatic fail over, and messages are saved on the local queue. A side effect of this failure is for the message queue (Q1) to fill up, as no JMS listener is retrieving messages. The MQSeries cluster queue continues to receive messages until the queue manager stops receiving messages.

- MQSeries Queue Manager failures.

  There are two different failures to consider:

  1. Failure of a queue manager on the same host as a WebSphere application server (for example, failure of QM2 on host S2). In this case the results are very similar to the first configuration. No messages are available to the application server for processing, and similarly no messages are added to the local queues until the MQSeries server is recovered. When the MQSeries server is back online, messages are processed as expected.
  2. Failure of the messaging host M3 and its queue manager, QM3. In this case, the result of an outage is more significant because no messages are delivered to the other queue managers in the cluster. In a fully-deployed and scaled production system, host M3 would not be designed to be a single point of failure, and additional messaging servers would be added to the cluster configuration.

- A transaction processing an incoming messages fails.

  There are two different failures to consider:

  1. No transaction is active. In this case, when no transaction has been started, the message is removed from the queue on the initial read. The message is effectively removed from the system when the application completes, whether or not the application is successful.
  2. The application fails within the scope of a transaction. If the application is processing the message within the scope of a transaction and an error is detected, the system causes a rollback condition where the message is returned to the queue. In this

case, the message is then available to be retrieved again. The JMS listener rereads the message and starts the processing again. If the transaction again fails, the message is returned to the queue, and the sequence of retrieving and processing the message is again repeated. If during the recovery processing the retry value specified in the configuration file for the destination is reached the JMS listener for the destination is stopped. Before subsequent messages can be retrieved the JMS listener must be restarted, by stopping and restarting the Enterprise JavaBean associated with the queue.

Extended messaging service task articles

This part contains task topics about the extended messaging service provided by WebSphere Application Server 4.0 enterprise services. These topics are intended to describe how you should complete tasks to enable and use the extended messaging service.

Installing and configuring MQSeries for use with the extended messaging service

Use this task to install and configure MQSeries for use with the WebSphere Application Server enterprise services extended messaging service:

To install and configure MQSeries for use with the extended messaging service, complete the following steps:

1. Install MQSeries 5.2 server as described in the installation instructions provided with the MQSeries package. Do not install the MQSeries classes for java or jms from the shipped CD.
2. Download and install the MQSeries 5.2 MA88 SupportPac (MQSeries classes for Java and MQSeries classes for Java Message Service) for your operating system from the following url:"http://www-4.ibm.com/software/ts/mqseries/txppacs/ma88.html"
   For information about how to install the MA88 SupportPac, see the installation instructions for your operating system provided on the download Web page.
3. If you want to use MQSeries - Publish/Subscribe support, you need to provide a Publish/Subscribe broker. You can do this by using MQSeries Integrator or the MA0C SupportPac.
   For more information about MQSeries Integrator, see the "MQSeries Integrator Web site at http://www-4.ibm.com/software/ts/mqseries/platforms/#integrator".

   If you want to use the MA0C SupportPac, complete the following substeps:

   a. Download the MQSeries 5.2 MA0C SupportPac (MQSeries - Publish/Subscribe) for your operating system from the following url:"http://www-4.ibm.com/software/ts/mqseries/txppacs/ma0c.h
   b. Install the MA0C SupportPac as described in the installation instructions for your operating system provided on the download Web page.
4. Follow the MQSeries 5.2 instructions for verifying your installation setup.
5. For UNIX platforms, add the following path to your LD_LIBRARY_PATH environment variable:`/mq_install_path/mqm/java/lib`
6. For AIX, as outlined in the MQSeries 5.2 readme.txt, modify the mqs.ini file and add the following attribute to all queue managers to be used with the WebSphere extended messaging service: `IPCCBaseAddress=11`
   In mqs.ini file (located in the /var/mqm directory) search for a line containing the following: `QueueManager:`

   The next line specifies the name of the queue manager. If this queue manager is used with the WebSphere extended messaging service add the above line as the last line of that queue manager's stanza. Repeat for other queue managers as needed.
7. For UNIX platforms, copy the following files from the */mq_install_path*/mqm/java/lib directory to the */WAS_HOME*/lib/ext directory

   fscontext.jar
   providerutil.jar
   com.ibm.mq.jar
   com.ibm.mqjms.jar
8.

/mqm/java/lib directory to the /*WAS_HOME*/bin directory

libmqjbnd02.so
libMQXAi01.so

This task has installed MQSeries and the supportpacs as required for use with the WebSphere extended messaging service.

You should next specify values for properties of the extended messaging service in its configuration file, as described in “Configuring the extended messaging service” on page 11.

Configuring the extended messaging service

Use this task to configure WebSphere Application Server for the extended messaging service, to specify the properties that control message handling by JMS listeners.

To configure properties of the extended messaging service, complete the following steps:

1. Configure JMS resources for use with WebSphere Application Server, as described in “4.6.3: Using Java Message Service (JMS) resources” in the WebSphere Application Server Advanced Edition infocenter. .
   Use the MQSeries JMS administration tool to configure the WAS name space to specify that you are using WebSphere JMS/XA connection factories.

   Note: To enable support for global transactions, you must use the WebSphere JMS/XA connection factories, because these ensure proper transaction behavior.

2. Edit the extended messaging configuration file to specify the properties for each JMS destination. A sample configuration file is provided with WebSphere, and is installed in the WebSphere samples directory.

   a. Create an extended messaging configuration XML file; for example, by copying an existing configuration file or the sample configuration file provided with WebSphere. The configuration file should contain the following format of XML tags:

```
<Config>
  <Pooling>
    <Timeout>timeoutMilliSeconds </Timeout>
    <Threshold>maxUnusedConnections </Threshold>
  </Pooling>
  <Listener>
    <HomeJNDIName> homeName </HomeJNDIName>
    <JMSConnectionFactory> factoryName </JMSConnectionFactory>
    <JMSDestination> destinationName </JMSDestination>
    <JMSDestinationType> destinationType </JMSDestinationType>
    <JMSSubscriptionDurability> durability </JMSSubscriptionDurabil
    <JMSSubscriptionName> subscriptionName </JMSSubscriptionName>
    <JMSMessageSelector> selector </JMSMessageSelector>
    <JMSAcknowledgeMode> acknowledgeMode </JMSAcknowledgeMode>
    <MaxRetries> retries </MaxRetries>
    <MaxSessions> sessions </MaxSessions>
    <Transactional> trueFalse </Transactional>
   </Listener>
  </Config>
```

   b. For each separate JMS destination to be monitored, edit the XML file to specify the properties of the JMS listener for that destination. Specify the properties within the <Listener> and </Listener> tags, like the example above. For more information about the property tags and their values, see “Format of the extended messaging configuration file” on page 18.

3. Specify the name of the configuration file on the **Configuration file URL** property of the **Extended Messaging Custom Service** entry for the application server in WebSphere system management. For example, use the WebSphere Administrative Console to complete the following steps:

   a. Expand the Topology tree to display the application server

   b. Click the application server to display its properties on the right side of the console.

   c. Select the Custom tab from the properties

panel.

d. Select the **Extended Messaging Support Service** then click **Edit**.
   If your application server was migrated from a WebSphere Advanced server installation that does not contain an Extended Messaging Support Custom Service, then you need to add the service by completing the following steps:
   1. Select **Custom Service -> Add**
   2. In the Name field type `Extended Messaging Support Service`
   3. In the Classname field type `com.ibm.cmm.listener.JMSListenerStub`

e. Type the name of the configuration file on the **Configuration file URL** property.

f. Click **OK** on the property sheet to save the property change.

g. Click **Apply** to apply the properties.

4. Specify the JMS Provider classpath for the Extended Messaging Custom Service entry for the application server in WebSphere system management. For example, use the WebSphere Adminitstrative Console to complete the following steps:

   a. Expand the Topology tree to display the application server.

   b. Click the application server to display its properties on the right side of the console.

   c. Select the Custom tab from the properties panel.

   d. Select Extended Messaging Support Service, then click **Edit**.

   e. Specify your JMS Provider classpath on the classpath property.

   f. Click **OK** on the property sheet to save the property change.

   g. Click **Apply** to apply the properties.

This task has specified values for properties of the extended messaging service in its configuration file.

When the extended messaging service initializes, it parses the configuration file. If the extended messaging service encounters an XMLParse error, it sends a message to the console with the file:line:column of the error, and the entry is ignored. If the error is considered as a fatal XML parser error, the parsing fails and initialization of the extended messaging service terminates. You must correct the configuration file before the extended messaging service can be initialized.

Developing a message bean to use extended messaging

Use this task to develop a message bean that a JMS listener invokes when a message arrives on the input queue that the listener is monitoring. You are recommended to develop the message bean to delegate the business processing of incoming messages to another Enterprise JavaBean, to provide clear separation of message handling and business processing. This also enables the business processing to be invoked by either the arrival of incoming messages or, for example, from a WebSphere EJB client. Responses can be handled by another Enterprise JavaBean acting as a sender bean, or handled in the message bean.

You develop a message bean to be invoked by a JMS listener like any other Enterprise JavaBean used by your enterprise applications. To develop a message bean, you complete the following stages:

1. Write the message bean class.
   By convention, the message bean class is named *name*Bean, where *name* is the name you assign to the message bean.

   The message bean class must meet the same requirements as any session bean. It must also define and implement the onMessage() method, which must meet the following requirements:

   - The method must be declared as **public**.
   - The method must *not* be declared as **final** or **static**.
   - The return type must be **void**.
   - The method must have a single argument of type `javax.jms.Message`.
   - The throws clause must *not* define any application exceptions.

   For example, a basic onMessage() method for a message bean is shown in the following code extract:

```
public void onMessage(javax.jms.Message msg)
{
    try
   {
        System.out.println("MySessBean.onMessage(), msg text:
"+((TextMessage)msg).getText());
    }
    catch  (Exception err)
    {
        err.printStackTrace();
    }
}
```

Figure 4 of 4. Code example: The onMessage() method of a message bean. This figure shows a code extract for a basic onMessage() method of a message bean. The method extracts the text from an incoming message and sends that text to System.out.

   For more information about writing the message bean class (as a session bean class), see “Writing the enterprise bean class (session)” in the WebSphere Application Server Advanced Edition infocenter. .

2. Write the message bean's home interface.
   By convention, the message bean's home interface is named *name*Home, where *name* is the name you assign to the message bean.

   For more information about writing the message bean's home interface, see “Writing the home interface (session)” in the WebSphere Application Server

Advanced Edition infocenter. .

3. Write the message bean's remote interface.
   By convention, the message bean's remote interface is named *name*, where *name* is the name you assign to the message bean.

   For more information about writing the message bean's remote interface, see “Writing the remote interface (session)” in the WebSphere Application Server Advanced Edition infocenter. .

When you have developed a message bean to use the extended messaging service, you must package and deploy the bean as part of your enterprise application. For more information about packaging and deploying an enterprise application, see:

1. “Packaging enterprise beans” in the WebSphere Application Server Advanced Edition infocenter. .
2. “Deploying enterprise beans” in the WebSphere Application Server Advanced Edition infocenter. .

Resolving problems with the extended messaging service

Use this overview task to help resolve a problem that you think is related to the extended messaging service.

The extended messaging service uses the standard WebSphere Application Server RAS facilities. If you encounter a problem that you think might be related to the extended messaging service, complete the following stages:

1. Check for extended messaging service messages in the admin console.
   If the extended messaging service fails to initialize, parse its configuration file, or encounters some runtime error an error message labelled with **WMSG** is sent to the admin console. The error message indicates the nature of the problem and provides some detail.
2. Check for more messages in the server's stdout.log.
   For more information about a problem, check the stdout.log file for the application server, which should contain more error messages and extra details about the problem.
3. **Optional:** Get a detailed exception dump for extended messaging.
   If the information obtained in the preceding steps is still inconclusive, you can enable the application server debug trace for the "Messaging" group to provide a detailed exception dump.

Extended messaging service example articles

This part contains example topics about the extended messaging service provided by WebSphere Application Server 4.0 enterprise services. These topics provide an overview of, and links to, the samples provided with WebSphere Application Server enterprise services.

- “Extended messaging service concept articles” on page 1
- “Extended messaging service task articles” on page 8
- “Extended messaging service reference articles” on page 17

Extended messaging samples

The following examples are provided to illustrate use of the extended messaging support:

- Point-to-point samples:
    - "Tutorial: Creating JMS message sample"

      This tutorial is designed to help you develop and deploy a JMS message sample application that tests the WebSphere Application Server extended messaging functions in a point-to-point scenario. This sample illustrates how to develop and deploy an application that comprises the following components:
        - A Java/JMS program that writes a message to a queue.
        - A message bean (a stateless session bean) that is invoked by a JMS listener when a message arrives on a defined queue.

      For more information about this sample, see the samples article "Tutorial: Creating JMS message sample" at `WAS_HOME/Enterprise/samples/messaging/doc/PtoP/wsbldPtoP.ht` (if you have installed the samples option).

    - "Sample: Message Listener (point-to-point)"

      This sample is designed to demonstrate the use and behavior of the extended messaging support for a simple point-to-point scenario. This sample uses the JMS message sample deployed in the sample above.

      For more information about this sample, see the samples article "Sample: Message Listener (Point-to-Point)" at `WAS_HOME/Enterprise/samples/messaging/doc/PtoP/wsrunPtoP.ht` (if you have installed the samples option).

- Publish/subscribe samples
    - "Tutorial: Creating JMS message publish/subscribe sample"

      This tutorial is designed to help you develop and deploy a JMS message sample application that tests the WebSphere Application Server extended messaging functions in a publish/subscribe scenario. This sample illustrates how to develop and deploy an application that comprises the following components:
        - A client program that starts the message sequence by publishing a message to a selected topic.
        - A message bean (a stateless session bean) that is invoked by a JMS listener when the broker passes a message to the listener from a topic to which it has subscribed.

      For more information about this sample, see the samples article "Tutorial: Creating JMS message publish/subscribe sample" at `WAS_HOME/Enterprise/samples/messaging/doc/PubSub/wsbldPubSu` (if you have installed the samples option).

    - "Sample: Message Listener (publish/subscribe)"

      This sample is designed to demonstrate the use and behavior of

the extended messaging support for a simple publish/subscribe scenario. This sample uses the JMS message sample deployed in the publish/subscribe sample above.

For more information about this sample, see the samples article "Sample: Message Listener (publish/subscribe)" at `WAS_HOME/Enterprise/samples/messaging/doc/PubSub/wsrunPubSub.` (if you have installed the samples option).

Extended messaging service reference articles

This part contains reference topics about the extended messaging service provided by WebSphere Application Server 4.0 enterprise services. These topics are intended to provide reference information that provides extra detailed information relevant to the extended messaging service that support the concept and task articles.

- “Extended messaging service concept articles” on page 1
- “Extended messaging service example articles” on page 15
- “Extended messaging service task articles” on page 8

Format of the extended messaging configuration file

The extnded messaging service configuration file has the following general format:

```
<Config>
  <Pooling>
    <Timeout>timeoutMilliSeconds </Timeout>
    <Threshold>maxUnusedConnections </Threshold>
  </Pooling>
  <Listener>
    <HomeJNDIName> homeName </HomeJNDIName>
    <JMSConnectionFactory> factoryName </JMSConnectionFactory>
    <JMSDestination> destinationName </JMSDestination>
    <JMSDestinationType> destinationType </JMSDestinationType>
    <JMSSubscriptionDurability> durability </JMSSubscriptionDurability>
    <JMSSubscriptionName> subscriptionName </JMSSubscriptionName>
    <JMSMessageSelector> selector </JMSMessageSelector>
    <JMSAcknowledgeMode> acknowledgeMode </JMSAcknowledgeMode>
    <MaxRetries> retries </MaxRetries>
    <MaxSessions> sessions </MaxSessions>
    <Transactional> trueFalse </Transactional>
   </Listener>
  </Config>
```

```
<Config>
Pooling-property-set
Listener-property-sets
</Config>
```

The scope of the JMS Listener configuration data, containing one **<Pooling>** property set and one or more **<Listener>** property sets. There must be only one **<Config>** tag pair in the configuration file.

```
<Pooling>
Timeout-property-tag
Threshold-property-tag
</Pooling>
```

This defines the scope of MQSeries connection pooling options. The ConnectionManager allocates connections on a most-recently-used basis, and destroys connections on a least-recently-used basis. You can specify the time that a connection is kept within the pool (**<Timeout>**) and the maximum number of unused connections (**<Threshold>**). There must be only one **<Pooling>** tag pair in the configuration file.

```
<Timeout>
timeoutMilliSeconds
</Timeout>
```

The number of milliseconds an unused connection is retained within the pool. The default is 300000 milliseconds (five minutes).

```
<Threshold>
maxUnusedConnections
</Threshold>
```

The maximum number of unused connections that are retained within the pool. The default is 10.

```
<Listener>
property_tags
</Listener>
```

The scope of each JMS listener destination. Each **<Listener>** tag pair contains the following properties (*property_tags*).

```
<HomeJNDIName>
bean_home_name
</HomeJNDIName>
```

The JNDI lookup name for the home of the message bean (a stateless session bean) that is invoked by this JMS listener, in the form: *bean_name*/*bean_name* .

```
<JMSConnectionFactory>
connection_factory_name
</JMSConnectionFactory>
```

The name of the JMS connection factory that you defined within JNDI (by using the JMSAdmin tool). For more information about defining this factory name, see “Configuring the extended messaging service” on page 11 .

```
<JMSDestination>
jms_destination
</JMSDestination>
```

The name of the JMS destination to be monitored by this listener, and that you defined within JNDI (by using the JMSAdmin tool). The value has the form: *jms*/*destination_name* , where *jms* is the name of your JMS function and *destination_name* is the name of the destination. For more information about defining this factory name, see “Configuring the extended messaging service” on page 11 .

```
<JMSDestinationType>
destination_type
</JMSDestinationType>
```

The type of JMS destination being monitored by the listener. The *destination_type* value must be specified as one of the following to match the type specified by the Java interface implemented by the destination:

javax.jms.Queue

(The default.)

javax.jms.Topic

This property is used for guidance to the deployer as to what type of destination to bind. The JMSListener runtime creation assumes the type of the object returned from JNDI. If the value specified is not valid (neither javax.jms.Queue nor javax.jms.Topic), then an XMLParse error is issued to the console when the JMS Listener initializes.

<JMSSubscriptionDurability>
durability
</JMSSubscriptionDurability>

Whether or not a JMS topic subscription is intended to be durable or nondurable. Specify this property *only if* the JMS Listener is to listen on a topic. The value is specified as one of the following:

durable

nondurable

(The default.)

<JMSSubscriptionName>
subscription-name
</JMSSubscriptionName>

A unique name that is associated with a durable subscriber, for JMS topic subscription.You must specify this property *only if* the JMS Listener is to listen on a topic, for durable subscriptions as indicated by the **<JMSSubscriptionDurability>**durable**</JMSSubscriptionDurability>** property.

<JMSMessageSelector>
jms_message_selector
</JMSMessageSelector>

A JMS message selector that the listener uses to determine which messages a message bean is to receive; for example:

```
<JMSMessageSelector>JMSType = 'car' AND color = 'blue' AND weight >
 2500</JMSMessageSelector>
```

<JMSAcknowledgeMode>
acknowledgeMode
</JMSAcknowledgeMode>

Whether JMS AUTO_ACKNOWLEDGE or DUPS_OK_ACKNOWLEDGE message acknowledgement semantics should be used for messages received by a non-transactional JMS listener. The value is specified as one of the following options:

auto-acknowledge

(The default.)

dups-ok-acknowledge

<MaxRetries>
maximum_retries
</MaxRetries>

The maximum number of retries that this listener attempts if it fails to retrieve a JMS

message from the destination that it is monitoring. The value is an integer number of retries, greater than or equal to 0 (zero); default 0. A value of 0 (zero) indicates no retries.

```
<MaxSessions>
maximum_sessions
</MaxSessions>
```

The maximum number of sessions that this listener can have with the destination that it is monitoring. The value is an integer number of sessions, greater than or equal to 1; default 1.

```
<Transactional>
trueFalse
</Transactional>
```

Whether or not the listener should start a global transaction before reading a message from the destination that it is monitoring. The value is either **True** or **False**; default **True**.

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing IBM Corporation North Castle DriveArmonk, NY 10504-1785 U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:
IBM World Trade Asia Corporation Licensing 2-31Roppongi 3-chome, Minato-ku Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS DOCUMENT "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR CONDITIONS OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the document. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:
IBM Corporation Department LZKS 11400 Burnet Road Austin, TX 78758 U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it

are provided by IBM under terms of the IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples may include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks and service marks

The following terms are trademarks or registered trademarks of the IBM Corporation in the United States, other countries, or both:

Advanced Peer-to-Peer Networking AFS AIX APPN AS/400 CICS CICS OS/2 CICS/400 CICS/6000 CICS/ESA CICS/MVS CICS/VSE CICSPlex DB2 DB2 Universal Database DCE Encina Lightweight Client DFS Encina IBM IBM System Application Architecture IMS IMS/ESA Language Environment	***	MQSeries MVS/ESA NetView Open Class OS/2 OS/390 OS/400 Parallel Sysplex PowerPC RACF RAMAO RMF RISC System/6000 RS/6000 S/390 SAA SecureWay TeamConnection Transarc TXSeries VSE/ESA VTAM VisualAge WebSphere

This software contains RSA encryption code.