Java Messaging Service Security on z/OS

Introduction

This paper discusses the use of JMS in WebSphere® Application Server on z/OS, particularly the security issues associated with this use. The paper is organized as follows:

- “Messaging in the WebSphere world” on page 2 introduces Java™ Messaging Service and Message Driven Beans (MDB).
- “WebSphere MQ” on page 8 discusses the use of JMS in conjunction with WebSphere MQ.
- “Service Integration Bus (SIB)” on page 16 discusses the concepts of the SIB in WebSphere Application Server.
- “SIB and WebSphere MQ” on page 22 discusses how to integrate the SIB with WebSphere MQ.
- “Security samples at an infrastructure level” on page 27 describes some samples of applications using security.
- “The JMS test application” on page 47 explains our JMS test application.
- “Definitions for the JMS test application” on page 81 discusses the definitions we have made for our JMS test application.
“Deploying the test application” on page 85 deals with the deployment aspects of the JMS test application.

“Running the JMSSession EJB client” on page 105 explains how to run the JMS test application.

“Tools used during the tests” on page 105, finally, describes some handy tools and commands used during our testing.

### Messaging in the WebSphere world

In the following sections we explain the concepts of Java messaging in WebSphere Application Server. There are two main elements:

- The Java Messaging Service (JMS), discussed in the next section.

### Java Messaging Service

The Java Messaging Service (JMS) API is an API for accessing enterprise messaging systems from Java programs. It is a part of the J2EE™ standard. For details, refer to:

http://java.sun.com/javaee/releases.jsp

JMS sets the standard, and it is up to the vendors to implement the standard in their messaging products. Most message-oriented middleware (MOM) products available on the market today have implementations, or JMS providers, for the JMS 1.02 and JMS 1.1 standards.

JMS provides a common way for Java programs to create, send, and receive messages. JMS messages are asynchronous, and support both point-to-point message queueing and a publish-subscribe style of messaging (Figure 1 on page 3).
As shown in Figure 1, there are two messaging models:

- **Point-to-point**
  This is a one-to-one approach, where one sender is sending a message to one receiver. This message is transmitted over a JMS queue. The sender adds the message to the queue and the receiver takes this message from the queue.

- **Publish-subscribe, or Pub-sub**
  This a one-to-many approach, where one sender is sending a message to many receivers. This message is transmitted over a JMS topic. This topic is predefined by the administrator of the MOM and is normally related to a message with a certain content (for example, “Sports”). The sender publishes the message to this topic. The message then get forwarded to all receivers that have subscribed to the actual topic of the message.

The Java programs that use the JMS API to send and receive messages are known as *JMS clients*. In JMS 1.0.2b, the following messaging domains were defined:

- For point-to-point: the *QueueSender* and *QueueReceiver*
- For publish-subscribe: the *TopicPublisher* and *TopicSubscriber*
In JMS 1.1, a sender or publisher of messages is called a *MessageProducer*, and a receiver or subscriber of messages a *MessageConsumer*. JMS 1.1 still supports the JMS 1.0.2b API. In the following discussion we use the term “Producer” if a JMS client sends messages and “Consumer” if a JMS client receives messages.

**JMS administration**

To assure portability of JMS applications between different providers the JMS standard defines two types of JMS-administered objects, which are stored in a JNDI tree:

- **Connection Factory**
  JMS clients use these objects to create a connection with the respective provider (for example, a WebSphere MQ queue manager).

- **Destination**
  JMS clients use these objects to send a message or receive a message (for example, a queue in WebSphere MQ).

**JMS messages**

JMS messages are composed of the following parts:

- **Header**
  All messages support the same set of header fields. Header fields contain values used by JMS clients and providers to identify and route messages.

- **Properties**
  A facility to add optional header fields to a message:
  - JMS standard defined fields that are optional
  - Application-specific properties
  - Provider-specific properties

- **Body**
  The application part of the message.

Key message header fields and property values are:

- **JMSDestination**
  Destination where the message is sent.

- **JMSMessageID**
  Contains a value that uniquely identifies a message.

- **JMSCorrelationID**
  Used to link one message with another. Typically used to link a request message with a response message, where the server application code copies
the JMSMessageID from the incoming message to the JMScorrelationID in the response message.

- **JMSReplyTo**
  Destination supplied by a client when a message is sent, to tell the server where to send a response.

- **JMSXUserid**
  The user ID of the user sending the message.

- **JMSType**
  A property supplied by the JMS client to describe the type of the message.

- **<JMS Client Properties>**
  Each JMS client can add its own (application-specific) properties.

For a complete list of header fields and property values see the JMS specification at http://java.sun.com/javaee/releases.jsp.

**JMS message selector**
Based on the header fields and property values discussed above, JMS offers JMS clients the possibility to select messages on JMS destinations by application-specific criteria.

**Note:** Message selectors can only reference header fields and property values of a JMS message, they cannot reference the content of the message body.

The message selectors are coded in a SQL-like style. Example 1 shows how a message selector can be used in a banking environment to retrieve certain JMS messages, which can then be treated in a specific way by the application.

**Example 1  Example of a message selector**

"businessCase = 'withdraw' AND customerRating <> 'gold' AND amount > 100000"

**JMS security**
The JMS specification does not cover message security. Security is considered to be a JMS provider-specific feature, and implementations vary from provider to provider.

The JMS API has no support for message encryption or signing, so it cannot be used to establish application-to-application or end-to-end security.
To work around this, the sending and receiving applications can use routines to do encryption of the message body either before the JMS API is called to send a message or after a message is retrieved. This depends on the JMS provider, such as WebSphere MQ (see “General WebSphere MQ security considerations” on page 14).

**Message-Driven Beans**

Like session and entity beans, Message-Driven Beans (MDBs) are part of the Enterprise JavaBeans™ (EJB™) standard. Details of the EJB specification are available at:

http://java.sun.com/javaee/releases.jsp

A Message-Driven Bean is an EJB that (indirectly) acts as a JMS message consumer. The Message-Driven Bean’s method `onMessage()` is invoked by the J2EE application server (for example, WebSphere) when a message arrives at a destination or a queue the MDB is configured to service. In WebSphere Application Server, the link between the JMS provider (WebSphere MQ or SIB) and the MDB is established by a JMS `Listener Port` definition when using WebSphere MQ as JMS provider, or an `Activation Specification` definition when using SIB as JMS provider. The concept of MDBs is illustrated in Figure 2.

![Figure 2 Message flow when using MDBs](image)
**MDBs and associated identities**

From a security standpoint, MDBs are not running under the identity of the JMS client that produced the message. By default, they run with a server's identity for "unauthenticated" users.

When the EJB for the business logic is called by the MDB, it may be desirable to run it under a user ID different from the MDB user ID, thereby enabling the use of standard J2EE role-based security at the user level. This can be done in either of the following ways:

- Set the Run-As attribute in the MDB Deployment Descriptor. See “Setting the RunAs Deployment Descriptor” on page 66.
- For each message, perform a JAAS login – for example, based on the JMSXUserID. This has the advantage of being more flexible than just setting a fixed value in the MDB Deployment Descriptor. For more details on possible implementations see “Switch user ID based on information passed in message” on page 74.

**Transaction support**

If transaction handling is specified for a destination, the J2EE application server starts a global transaction before it reads an incoming message from the destination and invokes the MDB. When the processing has finished, it commits or rolls back the transaction.

If an MDB itself uses JMS to produce messages, these messages are only sent when the transaction is committed by the J2EE application server. If a transaction rollback occurs, queued messages are discarded, and the incoming message is left on the destination.

In a z/OS environment, Resource Recovery Services (RRS) will be involved to coordinate a commit/rollback on all resources involved in the transaction.

**MDB and JMS message selector**

The JMS message selector discussed in “JMS message selector” on page 5 can also be used for an MDB to determine which messages the MDB receives.

The message selector is set in the deployment descriptor for the MDB and is used by the J2EE application server to select messages on destinations before invoking the respective MDB (see “Use of a JMS message selector” on page 64).

**Security aspects of the MDB message selector**

Because the MDB message selector can be used to select based on the JMS property JMSXUserID (the user ID of the JMS client that produced the message), you can use the MDB message selector to, for example, limit MDBs to be activated only for certain users or business scenarios (Example 2).
Example 2  MDB JMS message selector on JMSXUserID

JMSXUserID = 'andreas ' and JMSType = 'Buy'

Note: When specifying a JMS message selector on the JMSXUserID property, remember that in WebSphere MQ the user ID is right-padded with blanks to a length of 12 characters, so the message selector also must be coded with blanks for this property.

WebSphere MQ

In the following sections we provide relevant information on WebSphere MQ in association to WebSphere Application Server.

Overview

A WebSphere MQ (WMQ) network typically consists of WMQ nodes running on different platforms, and it can involve nodes from many companies. Messages are exchanged in an asynchronous way. The WMQ middleware will queue and persist messages to assure that the messages arrive at the destination even if a node or link is temporary unavailable.

While synchronous communication requires both parties to be available at the same time for the exchange of messages to take place, with asynchronous communication the application producing or sending a message can run even if the receiving partner application is not available.

This asynchronous way of sending and receiving messages reduces the influence a failure on one system can have on the entire network, and makes it easier to connect systems from different vendors and different companies.

WebSphere MQ is supported on more than 40 platforms, and applications using WebSphere MQ can be written in many languages. The need for WebSphere MQ messaging remains even in a services-oriented architecture (SOA) because legacy applications using MQ messaging cannot be easily changed to Web Services. Besides, new SOA- and Java-based applications will still have a need to exchange messages with legacy applications.

In an SOA, Web Services get more and more interest. Currently, Web Services are used mainly as a synchronous Remote Procedure Call (RPC) type of communication, using HTTP as the underlying protocol, but in many situations a true asynchronous protocol would be preferable. Delivery of the message can then be guaranteed even if the receiver is not available when the message is sent. WebSphere MQ supports Web Services over WebSphere MQ as transport.
For more information, see:
www-3.ibm.com/software/integration/support/supportpacs/individual/ma0r.html

In addition, WebSphere MQ is an important component in the Enterprise Service Bus (ESB) concept because WebSphere MQ can connect directly to WebSphere Application Server with a JMS provider shipped with WebSphere MQ, or it can connect with a link (MQ channel) to the new Service Integration Bus (SIB) introduced in WebSphere Application Server Version 6 (for details see “SIB and WebSphere MQ” on page 22). This is illustrated in Figure 3.

**Figure 3**  MQ and WebSphere messaging in an ESB environment

**Using WebSphere MQ to connect WebSphere with other systems**

Many back end z/OS transaction processing systems and consumers of MQ messages are still CICS® and IMS™. In addition, other non-Java systems are being built (for example, based on .Net), so even if the number of Java-based systems using WebSphere and JMS gradually increases, there will still be the need to couple WebSphere Application Server-based Java (JMS) messaging with non-Java messaging.

Figure 4 shows a customer system that uses a non-Java environment to send request messages to WebSphere on z/OS, where they are processed and replied to.
Let’s assume a customer’s system is connected to your system with an MQ link over TCP/IP or SNA to execute a service or transaction in your system. Referring to Figure 4, the steps are as follows:

1. The application constructs and sends a message to its local MQ system with an application-specific message body and an MQ message header (MQMD) including a message ID, a destination MQ system (Queue Manager) and queue name, and the name of the reply queue if the application expects to get a message back. MQ updates more fields in the MQ header and writes the message to a transmission queue for the destination MQ system.

2. The MQ channel system or Message Channel Agent (MCA) picks up the message from the transmission queue and sends it to the destination, where the other MCA receives the message and writes to the destination queue.

3. A WebSphere Listener or Activation Specification is “listening” to the destination queue, reads the message, and starts an MDB that parses the message and calls the correct business function in another EJB.

4. The business logic runs under the user ID inherited from the MDB.

5. The MQ reply is constructed (either in the MDB directly or in a dedicated EJB) using the JMS API. The MQ header is filled with the name of the reply MQ system and queue name. The message ID for the incoming message is placed in the correlation ID field in the header. The bean building the reply message runs under the user inherited from the MDB.
6. The reply message is written to the transmission queue.

7. MQ channel subsystems transport the message back to the origination system and put the message in the reply queue.

8. The application reads the reply from the queue, using the correlation ID.

JMS messages and non-JMS messages in WebSphere MQ

The previous scenario discusses messaging between a non-JMS customer system and a JMS system (WebSphere Application Server on z/OS). Using WebSphere MQ utilities such as amqsbcg (for a local queue manager) or amqscgc (for a queue manager accessed via a client connection) you can see that WebSphere MQ is able to work with JMS messages and non-JMS messages, as shown in Example 3 and Example 4 respectively. The “payload” of both messages was TestText.

But how are the two message types converted in WebSphere MQ when a non-JMS system (such as CICS) and a JMS system (such as WebSphere Application Server) communicate with each other over WebSphere MQ? Luckily, no direct conversion needs to take place since in the WebSphere Application Server WebSphere MQ queue destination you can specify the attribute “Target Client” if the messages on this destination are to be expected in JMS or non-JMS format. So, a WebSphere Application Server MDB can process messages from a non-JMS client. One limitation exists, however: since non-JMS messages do not have all the JMS properties, the use of an MDB message selector is constrained.

Example 3  Example of a JMS message in WebSphere MQ

****Message descriptor****

StrucId : 'MD'  Version : 2
Report : 0  MsgType : 8
Expiry : -1  Feedback : 0
Encoding : 273  CodedCharSetId : 819
Format : 'MQHRF2'
Priority : 4  Persistence : 1
MsgId : X'414D51204C414E44454E42202020203FCABC4420000301'
CorrelId : X'000000000000000000000000000000000000000000000000'
BackoutCount : 0
ReplyToQ : '
ReplyToQMgr : 'LANDENB'
** Identity Context
UserIdentifier : 'andreas'
AccountingToken :
X'16010515000000DC7556D36293DB009AA34694EC0300000000000000000B'
ApplIdentityData :
** Origin Context
PutApplType : '11'
Example 4   Example of a non-JMS message in WebSphere MQ

****Message descriptor****

StrucId : 'MD '  Version : 2
Report : 0  MsgType : 8
Expiry : -1  Feedback : 0
Encoding : 273  CodedCharSetId : 1208
Format : 'MQSTR '
Priority : 4  Persistence : 1
MsgId : X'414D51204C414E44454E42202020203FCABC4420000401'
CorrelId : X'000000000000000000000000000000000000000000000000B'
BackoutCount : 0
ReplyToQ : ''
ReplyToQMgr : 'LANDENB'
** Identity Context
UserIdentifier : 'andreas '
AccountingToken : 'X'1601051500000000C7556D36293DB009AA34694EC030000000000000000000B'
ApplIdentityData : ''
** Origin Context
PutApplType : '11'
PutApplName : 'es\base_v6\java\bin\java.exe'

Example 4   Example of a non-JMS message in WebSphere MQ
WebSphere MQ channel exits

In a wide-spread messaging environment like WebSphere MQ it is often necessary to adapt the behavior of the messaging infrastructure to customer requirements or to intercept the dataflow of the system. To enable this, WebSphere MQ offers so-called exits where user-supplied code is called at pre-defined points.

From the viewpoint of security on WebSphere MQ on z/OS, the most important exits are defined at the Message Channel Agents (MCAs) (Figure 4 on page 10):

- **Security exit**
  This exit is called after the network connection between two channels is established but before messages flow. One example is that the two channels authenticate to each other. Only one security exit can be assigned to a channel.

- **Message exit**
  As the name implies, a message exit is called for every message transmitted – for example, to offer the possibility to compress messages. A list of message exits can be assigned to a channel and the exits are executed in sequence.

- **Send/receive exit**
  A send exit is called directly before a WebSphere MQ channel sends data over the network; a receive exit is called directly after a WebSphere MQ channel has received data over the network. Among other things, they offer the possibility to encrypt/decrypt data before sending it over the network. (Send/receive exits are better suited then message exits for encryption/decryption since they can also encrypt/decrypt the header of the message.) A list of send/receive exits can be assigned to a channel and the exits are executed in sequence.
WebSphere MQ and security

Note: In this paper we are focused on inheriting security context in a WebSphere environment. For the full picture of WebSphere MQ security refer to the pertinent WebSphere MQ product manuals and IBM® Redbooks™.

General WebSphere MQ security considerations
Security in a network with asynchronous messaging and with queue managers on different platforms administered by different companies is a challenge.

Queue managers will probably be in different security domains with their own user registries. Some of the queue managers may have a comprehensive security implementation, while other queue managers may have limited or no security implemented, so the security context passed in the WebSphere MQ message header may have limited value.

In general, WebSphere MQ relies on the application using WebSphere MQ to do authentication of the user. Credentials are passed from the application to WebSphere MQ, and access to the WebSphere MQ resources is checked by the local security system.

On z/OS, RACF® or a SAF-compatible security system can be used to control access to MQ resources like message queues, channels, and commands.

Links or channels between WebSphere MQ queue managers can be secured in various ways. In a TCP/IP network, for example, use of SSL for encryption and X509V3 certificates is a standardized way.

But in a complex WebSphere MQ network, link-level security may not be sufficient. Application-level or end-to-end security—where the application sending the message encrypts or signs the message, and the receiving application decrypts the message and verifies the user—may have to be implemented. The cost and complexity of implementing this application-level security can be relatively high. One product that supports application-level security is WebSphere MQ Extended Security Edition, which is based on top of WebSphere MQ. See http://www-306.ibm.com/software/integration/wmq/securityedition/about/ for more details.

Authentication
When a WebSphere Application Server z/OS application connects to WebSphere MQ, its credentials (user ID and password or user identity associated with the current thread) are checked by WebSphere MQ against RACF.
Authorization
The following major authorizations can be granted to users and groups in WebSphere MQ by giving access to defined RACF CLASSes and PROFILEs:

- Connect: Connect to a WebSphere MQ queue manager
- Browse: Browse a queue
- Get: Get messages from a queue
- Put: Put messages onto a queue

WebSphere MQ security administration
Since WebSphere MQ security is based on RACF, RACF commands are used to administer WebSphere MQ security. This also impacts auditing. Events like authentication success and failure and authorization failure generate SMF records/messages to the console.

Request flow when using WebSphere MQ
For an example, we assume the two queue managers in the scenario shown in Figure 4 on page 10 are in different security domains with their own user registers and security implementations. In the following discussion we consider this scenario, focusing on WebSphere MQ security:

1. First, the application must connect to WebSphere MQ. The credentials the application is running under are used. If security is enabled, the authorization to connect is checked against the local security system.

   Then the application must have write access to the target queue. When the target queue is on a remote system, it is good practice to use a remote queue definition and an alias queue name to shield the application from changes in the infrastructure, and use the alias queue name to enforce security.

   When the message is written, a user ID or UserIdentifier is added in the MQ message header or WebSphere MQ message descriptor. By default, the user ID the application is running under will be placed here. (Note that it has a maximum length of 12 characters.) If the user ID the application is running under is authorized for it, it can set another UserIdentifier.

2. When a message for a remote system arrives at the transmission queue, the channel or Message Channel Agents (MCA) start automatically. It is then possible for the MCA at each end of the channel to authenticate its partner. This can be done using x509v3 certificates.

   The MCA on the sending side must have authority to read from the transmission queue; the MCA on the receiving side must have authority to set context information in the message and write to the target queue.
On the receiving MCA it is possible to control what user ID is used for the incoming message. The UserIdentifier field in the WebSphere MQ header can be used. If the message comes from another security domain this is probably not a good idea since it may not be useful on the local system. Another alternative is to define a fixed MCA user ID on the receiving channel with an attribute that says that only the MCA user ID should be used for incoming messages. This is the user ID used to store the incoming message, but the UserIdentifier in the WebSphere MQ header or message descriptor is still the same. If the target application needs a valid user ID, it is possible to use a channel message exit to set a new user in the UserIdentifier field in the message descriptor. In our sample application we used a channel message exit to move the MCAUserId into the transmission queue header (MQXQH) where the message descriptor and the UserIdentifier are imbedded. If you have one channel per business partner and you only need one user ID for this business partner, then this is a simple and pragmatic solution.

3. The receiving application (in our case a JMS Message-Driven Bean) must have authorization to read from the queue.

The role of the MDB is to parse the message, and it may perform security checks (for example, based on the UserIdentifier field in the WebSphere MQ message descriptor that is available to a JMS application in the JMSXUserID field), and call an EJB to do the business logic.

4. The business logic runs under the user inherited from the MDB.

5. The bean building the reply message runs under the user inherited from the MDB.

6. The MCAs transport the message back to the origination system. The security can be controlled the same way in both channel directions. MCAUserId and message exits can be used to override the UserIdentifier passed in the message descriptor.

7. The application reads the message from the reply queue. This is often a temporary queue created by the application, so the application has full access.

Service Integration Bus (SIB)

In the following sections we describe the Service Integration Bus (SIB) introduced in WebSphere Application Server Version 6.

New messaging support introduced in WebSphere V6

WebSphere Application Server V6 provides a pure Java JMS 1.1 provider that is installed as part of the base server installation and runs completely inside the
application server JVM™. Each application server can host a *messaging engine* (ME). Messaging engines can be interconnected to form a *messaging bus*. Persistent messages are stored in either an embedded Cloudscape™ database or an external database of choice via a JDBC™ driver. Messaging improvements that are provided in WebSphere Application Server V6 include:

- Interoperability with WebSphere MQ
- Complements and extends WebSphere MQ and application server
- All Java implementation
- Better integration of the messaging provider with the application server
- Integrated publish/subscribe
- Integrated reliability, serviceability, and problem determination
- Cluster support
- High availability enablement

WebSphere Application Server V6 supports the following JMS providers:

- Default messaging
  Based on the built-in Service Integration Bus
- WebSphere MQ
  Using an external WebSphere MQ
- Generic
  For adding a JMS provider of choice
- V5 default messaging
  For compatibility with messaging provided in WebSphere Application Server Version 5

**SIB overview**

*Service integration* is a new concept in WebSphere V6 covering both JMS messaging and Web services. Service Integration Bus (SIB) is a core component for implementing an Enterprise Server Bus (ESB), and the new WebSphere ESB product is build on top of SIB.

**Bus**

A *service integration bus* is a messaging system or message bus within a WebSphere cell. A bus can span nodes within a WebSphere cell, and a cell can have several buses defined. Buses can be interconnected with *links*.

A bus is similar to a WebSphere MQ cluster.
**Message engine (ME)**

A bus contains *message engines* (MEs) to send and receive messages from the JMS application. The message engine uses the bus to forward the messages to the destination. Applications connect to a message engine running in the bus (it does not matter if this messaging engine runs in the local WebSphere Application Server instance or not).

A message engine is similar to WebSphere MQ Queue Manager.

**Foreign bus**

The local bus can be connected to other buses with links. The foreign bus can be another service integration bus or WebSphere MQ.

A foreign bus is similar to WebSphere MQ remote queue manager.

**Bus destination**

A *destination* is a location on the bus that the application can send messages to and receive messages from. If the destination is in the same bus, the application can receive and send messages. If the destination is on a foreign bus, the application can only send messages.

The main destination types are:

- **Queue**
  Used for point-to-point messaging

- **Topic space**
  Used for publish/subscribe messaging

- **Foreign**
  Used for destination on a foreign bus

- **Alias**
  Name used for a destination to shield the application from changes in the infrastructure

These definitions are similar on WebSphere MQ, where *foreign* is called *remote*.

**Mediation**

It is possible to connect programs to a queue or topic space destination that can manipulate the messages when the queue is accessed. This facility is called *mediation handler*; and a list of programs, a *mediation handler list*, can be specified for a destination. The mediation handlers run as session beans in the EJB container, and can transform the message, reroute or log the message, or perform any type of processing that is appropriate to do at that point.
Mediations can be useful to build a good application architecture, in which common functions such as logging of messages can be kept out of the business logic.

**Note:** Mediation provided in the service integration bus is not the same as mediation provided in the WebSphere ESB product.

A mediation is (very roughly!) similar to a WebSphere MQ exit.

### SIB and WebSphere administrative console

#### Administration

Administration of the Service Integration Bus is integrated in the WebSphere administrative console, but with WebSphere Application Server Version 6 the wadmin tool has to be used to administer authorization to some of the SIB resources.

From the administration console use the sample command sequences shown to work with the SIB and perform the following common administrative activities:

- To check status of a link:
  
  \[
  \text{Service Integration} \rightarrow \text{Buses} \rightarrow \text{busname} \rightarrow \text{Bus members} \rightarrow \text{Messages engines} \rightarrow \text{name of engine} \rightarrow \text{WebSphere MQ links} \rightarrow \text{link name}
  \]

- To look at a message on a queue:
  
  \[
  \text{Service Integration} \rightarrow \text{Buses} \rightarrow \text{busname} \rightarrow \text{destination} \rightarrow \text{select destination} \rightarrow \text{queue points} \rightarrow \text{select queue point} \rightarrow \text{select runtime flip} \rightarrow \text{messages} \rightarrow \text{select message}
  \]

- To check a JMS destination:
  
  \[
  \text{resources} \rightarrow \text{JMS} \rightarrow \text{default messaging} \rightarrow \text{jms queue} \rightarrow \text{select queue}
  \]

- To check or define a JMS activation specification:
  
  \[
  \text{resources} \rightarrow \text{JMS} \rightarrow \text{default messaging} \rightarrow \text{jms activation specification} \rightarrow \text{select activation specification}
  \]

- To check or define a JMS listener:
  
  \[
  \text{server} \rightarrow \text{application servers} \rightarrow \text{select server} \rightarrow \text{scroll down to find messaging and expand} \rightarrow \text{select Message Listener Service} \rightarrow \text{listener ports} \rightarrow \text{select listener name}
  \]

- To check listener or activation specification in deployment descriptor:
  
  \[
  \text{Applications} \rightarrow \text{Enterprise Application} \rightarrow \text{select application} \rightarrow \text{select Provide listener bindings for message driven bean}
  \]
SIB security

Security for the SIB is integrated in WebSphere and is enabled when *Global Security* is enabled. When a bus is created, a default set of roles is also created. If the *secure* attribute is set on the bus, security roles will be enforced. The default roles allow all authenticated users to access the bus and all the local bus destinations.

**Authentication**

When Global Security is enabled, WebSphere users are authenticated. Authentication is checked between the client and the message engine, as well as between message engines on the bus.

**Authorization**

Authorization to access the bus and resources on the bus is checked against the roles the user/group has been given access to.

The major roles and capabilities available are:

- **BusConnectRole**
  Connect to the bus.

- **Sender**
  Send a message to a destination.

- **Receiver**
  Receive a message from a destination.

- **Browser**
  Browse a destination.

- **Creator**
  Create a (temporary) destination.

**SIB security administration**

In WebSphere Application Server Version 6 access to the bus and resources on the bus is administered through the wsadmin tool. An example is shown in Example 5.

---

**Note:** An alternative tool that can be used to “browse” the SIB is described in “Service Bus Explorer” on page 105.
Example 5  A wsadmin command to administer SIB security

$AdminTask addUserToDestinationRole {-type queue -bus JMSBus -foreignBus MQ4B -destination JMSTESTQ -role Sender -user WSADMIN}

For more information on this topic see the WebSphere V6 Information center or the product manuals.

Security events like authentication success and failure and authorization failure are written into the WebSphere system log.

SIB and SIB interconnection
Multiple Service Integration Buses can be interconnected, as shown in Figure 5.

![Figure 5  Service Integration Bus link between WebSphere cells](image)

In this scenario we have a pure WebSphere environment with two cells connected for message exchange via the Service Integration Bus.

Most of the details concerning this scenario were covered in the discussion of the previous two scenarios.

With cells from different security domains, you probably want to use Inbound user ID and Outbound user ID on the link in both cells.

Messaging between applications in the same WebSphere cell
This scenario is similar to messaging between two WebSphere cells, but in this case you always have the same user registry. There are no links nor Inbound or Outbound user IDs involved. Permission must be given to connect to the bus, and to send to and receive from destinations.
Some of the applications may run under logged-on users’ credentials, others can run with the server credentials.

Using default messaging within a cell without any links you can in most cases just use the default security setup, where all authenticated users have access to the bus and its destinations.

**SIB and WebSphere MQ**

In the following sections we discuss the interoperability between the Service Integration Bus and WebSphere MQ.

**Similarities and differences**

Table 1 compares terminology and features in WebSphere MQ with those in a Service Integration Bus (SIB).

<table>
<thead>
<tr>
<th>Service Integration Bus</th>
<th>WebSphere MQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bus</td>
<td>Cluster</td>
</tr>
<tr>
<td>Message engine</td>
<td>Queue manager</td>
</tr>
<tr>
<td>Foreign bus</td>
<td>Remote queue manager or cluster</td>
</tr>
<tr>
<td>Link</td>
<td>Channel</td>
</tr>
<tr>
<td>Foreign</td>
<td>Remote</td>
</tr>
<tr>
<td>Platform: WebSphere</td>
<td>Platform: ca. 40 platforms</td>
</tr>
<tr>
<td>Supports JMS 1.1 Java</td>
<td>Supports JMS 1.1 and proprietary MQ API Java and many other 3 GL languages</td>
</tr>
<tr>
<td>Supports 5 levels of message persistence</td>
<td>Supports 3 levels of message persistence</td>
</tr>
<tr>
<td>Mediation</td>
<td>Exit (very roughly!)</td>
</tr>
</tbody>
</table>

WebSphere MQ security uses external security systems like RACF, and can be administered by security personnel. Service Integration Bus security is imbedded in WebSphere and requires WebSphere skills.

There are many tools available to operate and monitor a WebSphere MQ network while for SIB you have to use the WebSphere admin console or the SIB Explorer (see “Service Bus Explorer” on page 105).
When to use SIB and when to use WebSphere MQ

The following arguments might help you to decide if you want to use WebSphere MQ or SIB:

- Do you have messaging only between applications in a WebSphere cell? If the answer is “yes” then you could use the SIB.
- Do you need to use mediation handlers? If the answer is “yes” then you may want to use the SIB. (However, even though mediation handlers are a useful low-level construct, the SIB environment does not provide a true ESB, as displayed in Figure 3 on page 9).
- Do you already have a WebSphere MQ infrastructure with administration, security, operation, and monitoring in place? If the answer is “yes” then you probably not want to replace it with the Service Integration Bus.
- Do you have many different systems attached to your WebSphere MQ base messaging infrastructure? If the answer is “yes” then maybe you should also base your WebSphere Application Server environment on WebSphere MQ.
- Do you need administration tools that are integrated into an existing Systems Management environment? If the answer is “yes” then maybe you should use WebSphere MQ.

In general: Service Integration Bus is new with WebSphere Application Server 6.0 and can be used only within WebSphere and Java applications. MQ is proven over an long period, and can be used in practically all environments and on all platforms.

WebSphere MQ with link to SIB

Figure 6 on page 24 shows an infrastructure using an MQ link as a bridge from WebSphere MQ to the Service Integration Bus.
This scenario is similar to the scenario introduced earlier in this chapter. The MQ JMS provider is replaced with an SIB and default messaging JMS provider.

We assume that the security attribute on the defined bus is turned on.

An MQ link is defined in the SIB to a foreign bus that is an MQ queue manager. From the MQ side a sender and receiver channel is defined in the normal way and the SIB in WebSphere looks from the WebSphere MQ side just like a remote MQ queue manager.

The link or channel between the two systems can be secured with SSL in the same way this is done between two MQ systems.

In Figure 6 we use the same numbering for the events used in the previous scenario to illustrate the changes in security:

1. The application puts the message into a queue. Security considerations are the same as in the previous scenario. The user the application is running under must have authority to connect to MQ and write to the queue.

2. The MQ Message Channel Agent (MCA) discovers the message. If the channel or MQ link is not already started, it starts automatically. SSL security with mutual authentication can be used when the session is established.

   The MQ MCA must have authority to read the message from the transmission queue, and the user ID on the receiving side must have permission to connect to the bus and sender permission to the destination.

   On the MQ link an Inbound user ID can be defined similar to the MCAUser1D on MQ.
If an Inbound user ID is defined this user will be used to authenticate message flows from the foreign bus and to authorize individual messages to destinations on the local bus.

As stated previously, in communication with nodes in other security domains you probably want to use the Inbound user ID to get a valid user in your domain. If it is not used the user ID stored in the incoming message, JMSXUserID, is used to check access.

When the Inbound user ID is defined, it replaces the user ID in the incoming message. This is different from using MCAUserID in MQ, where the user ID in the message is unchanged.

3. To read the messages from the destination the Message-Driven Bean must have connect permission to the bus and receive permission for the destination. Doing a JAAS logon with the JMSXUserID or setting up a user ID in the MDB DD and passing control to the business session bean is exactly the same as in the MQ scenario.

The same Message-Driven Bean can be used unchanged in the two scenarios.

4. The business logic runs under the user inherited from the MDB (see “MDBs and associated identities” on page 7).

5. The MDB calls an EJB to build the reply message. It runs under the user inherited from the MDB.

6. The EJB building the reply message must have permission to send messages to the foreign bus, and may need access to the foreign destination if the user ID is not overridden when the message enters the foreign bus. When the message is constructed the current user of this EJB is set in the message context.

7. It is possible to define an Outbound user ID on the MQ link. If an Outbound user ID is used this will be set in the UserIdentifier of the message context.

The user must have sender permission to the foreign bus and the foreign destination. The foreign bus definition provides a default when the foreign destination definition does not exist, which often will be the case in a request/reply type of conversation.

At the receiving MQ channel the MCAUserID can be used to control access to the incoming queue. This will not change the UserIdentifier in the message context. That can be done with an MQ message exit.

8. The application reads the message from the reply queue. This is often a temporary queue created by the application, so the application has full access.
SIB with link to WebSphere MQ

Figure 7 shows an infrastructure using an MQ link as a bridge from WebSphere MQ to the Service Integration Bus.

In this scenario we have a Java servlet or EJB-based application hosted by WebSphere Application Server. The Service Integration Bus and default messaging is used with an MQ link to back-end systems, which typically could be CICS or IMS systems.

WebSphere global security and bus security is turned on. Referring to Figure 7, the actions are as follows:

1. The application running in WebSphere must have connect permission to the local bus, and sender permission to the foreign bus with the back-end server application. The WebSphere application can run under a logged on Web user ID, the server’s user ID, or a “run-as” user ID. Users are authenticated (internally) when they connect to the bus, and authorization is checked for each message. When the message is constructed the current user is set in the message context.

2. If the Outbound user ID is defined on the MQ link, this user ID will be set in the message context and used for authorization. The user ID used must have sender permission to the foreign bus and sender permission to the foreign destination.

   On the receiving MQ side the MCAUser ID can be used, or the user ID in the message must be to defined and given access to write to the destination queue. This is exactly the same as the first scenario with two MQ systems.
3. The application or subsystem on the back end has to have the rights to read the message from WebSphere MQ.

4. The application on the back end might involve several processing steps.

5. The application on the back end might involve several processing steps.

6. The application or the subsystem on the back end has to have the rights to write the response message to WebSphere MQ.

7. If the back-end server was in a different security domain or from a system without enabled security, you probably want to use the Inbound user ID. This ID will be set in the message and used for authorization. Sender permission to destination is required.

8. Receiver permission is needed to read to message from the destination.

**Security samples at an infrastructure level**

In the following sections we provide some examples of exactly how to apply security in the messaging environment.

**Usage of MQ channel exit for security**

As discussed in “WebSphere MQ channel exits” on page 13, WebSphere MQ on z/OS offers the possibility to use the Message Channel exits for security. Two ideas are discussed here, along with how they could be of value for WebSphere on z/OS. However, since these security exits are pure WebSphere MQ, we do not go into much detail.

**Note:** Since the exits discussed here are used by the Message Channel Agents (MCAs) of WebSphere MQ, they are only in effect in a distributed environment. They are not useful when, for example, a CICS region on the same LPAR as WebSphere z/OS sends a WebSphere MQ message to a WebSphere MDB using a WebSphere MQ queue manager on the same LPAR.

**Controlling user identity**

A standard WebSphere MQ message exit can be used on the receiving side to copy the MCAUserId defined in the channel definition to the UserIdentifier (JMSXUserID) field in the message contexts. In this way you can, for example, forward a valid RACF user ID to the Message-Driven Bean. The limitation with this implementation is that only one user ID can be used per MQ channel.
You can build a more advanced version of the exit where it is possible to specify valid user IDs or a generic user ID in the userdata parameter for the exit; userdata is a 32 byte long string specified as a parameter to the exit in the channel definition. It can be easily modified and will be active at the next restart of the channel. If a match is found between the UserIdentifier in the message header and a user ID in the userdata parameter, the UserIdentifier is kept unchanged. If no match is found the MCAUserid is copied.

**Routing of messages**

Another possible use of a message exit is to change the destination of the message so it is routed to another queue. This can be used to route messages to different instances of the MDB, which can be set up with different run-as users. A better solution for such a requirement might be the use of an MDB Message Selector, though.

The same effect could also be achieved without routing the request to different queues; the JMS message selector of the MDB can be used to set up the MDB to consume only messages with a specific value in a field in the message header or message properties. So, several MDBs with different run-as identities and different message selectors can then be set up to listen on the same destination (see “Use of a JMS message selector” on page 64).

**Message exit to control access to queues**

A WebSphere MQ message exit can also set a return code to deny access to a queue. WebSphere MQ then routes the message to a WebSphere MQ “dead letter queue.”

**Service Integration Bus mediation handlers**

In an SIB environment mediation handlers can be written in Java to achieve similar functionality to what MQ exits can do in an MQ environment. They can be used for security purposes, such as to check the data in a JMS message for (seldom changed) business constraints or to change the user ID associated with a message based on the data in the message.

**Note:** A SIB mediation handler is on a per destination basis (for example, JMS queue). WebSphere MQ exits are on a channel basis, so mediation handlers are more powerful and flexible than MQ exits. However, mediation handlers should not be used to hand code an Enterprise Service Bus.

A mediation handler is associated with one or more destinations, and is invoked when the destination is accessed. The mediation has access to the message and its context, and can change both properties and the contents of the message.
A mediation handler implements a generic interface, named `MediationHandler` (com.ibm.websphere.sib.mediation.handler.MediationHandler), which is invoked from the Service Integration Bus. The `handle()` method gets the `MessageContext` passed. The `SIMessage` can be retrieved for the context.

A mediation handler can use the `SIMessage` interface to access supported JMSX properties defined in the JMS API. The `JMSXUserID` property can be modified with the `SetMessageProperty()` method.

The `SIMediationSession` interface defines the methods for querying and interacting with the Service Integration Bus. The API also includes methods that provide information on where the mediation is invoked from, and the criteria that must be met before the message is mediated.

The most important methods of the API are:

- `GetBusName`
  Returns the name of the bus this mediation runs on

- `GetDestinationName`
  Returns the name of destination

- `GetMediationName`

- `GetMessageSelector`

- `GetMessageDiscriminator`

- `receive`
  Receives a `SIMessage`

- `send`
  Sends a copy of the `SIMessage`

- `resetIdentity`
  Changes the identity of the given message to the current run-as identity

**Building a mediation handler**

To build a mediation handler, perform the following steps using Rational® Application Developer Version 6:

1. Create the mediation handler as a Java class that implements `com.ibm.websphere.sib.mediation.handler.MediationHandler`
   
   For sample code see Example 6 on page 35.

2. Wrap the mediation handler into a deployable EJB, as follows:
   
   a. Create a new Enterprise Application containing one EJB project.
   
   b. In the EJB project reference the Java project that holds the Java class created before (via **Properties → Java Build Path**).
c. Open the Deployment Descriptor of the EJB project and select the **Mediation Handlers** tab, as shown in Figure 8.

![Mediation handler in the EJB Deployment Descriptor](image)

*Figure 8  Mediation handler in the EJB Deployment Descriptor*

d. Click **Add**. This will bring up the window shown in Figure 9 on page 31.
e. Make the following entries:
   Name: Mediation
   Description: SecurityMediationHandler

f. Click Browse and select the Java class created previously.

g. Click Show Advanced and check “Define Handler Lists containment.”

h. Click Next. This will bring up the window shown in Figure 10 on page 32.
Figure 10 Defining a mediation handler list

i. Enter MediationHandlerList in the Handler list name field. Click Add. This will bring up the window shown in Figure 11 on page 33.

Note: This name will be referenced later when installing the mediation handler into WebSphere Application Server
j. Click **Finish**.
The completed Deployment Descriptor will look like the panel shown in Figure 12 on page 34.
In the EJB project, a stateless session bean (called Mediation) is automatically generated; it wraps the mediation handler and exposes it as a mediation handler list.

**SIB mediation handler code**

An SIB mediation handler can be used for security to, for example, check the data in a JMS message for (seldom changed) business constraints or change the userid associated with a message based on the data in the message.

Example 6 on page 35 shows an example of the Java code of a mediation handler changing the user ID and data of the message.
package ejbs.helper;

import java.util.Iterator;
import java.util.List;

import javax.xml.rpc.handler.MessageContext;

import com.ibm.websphere.sib.SIApiConstants;
import com.ibm.websphere.sib.SIMessage;
import com.ibm.websphere.sib.mediation.handler.MessageContextException;
import com.ibm.websphere.sib.mediation.messagecontext.SIMessageContext;
import com.ibm.websphere.sib.mediation.session.SIMediationSession;

import commonj.sdo.DataGraph;
import commonj.sdo.DataObject;

public class MediationHandler implements
com.ibm.websphere.sib.mediation.handler.MediationHandler {

public boolean handle(MessageContext msgCtx) throws MessageContextException {

SIMessageContext siMsgCtx = (SIMessageContext) msgCtx;

// Display Message-Userid
System.out.println("Mediation: Userid = " + siMsgCtx.getSIMessage().getUserId() + ">");

// Check contents of Message and switch the
// Userid if requested

try {
    SIMessage msg = siMsgCtx.getSIMessage();
    String msgfmt = msg.getFormat();
    DataGraph data = msg.getDataGraph();
    DataObject msgRoot = data.getRootObject();

    if (msgfmt.equals(SIApiConstants.JMS_FORMAT_TEXT)) {
        if (msgRoot.isSet("data/value")) {
            String msgText = msgRoot.getString("data/value");
            System.out.println("Mediation-Text (IN): " + msgText);
            if (msgText.equals("mediationSwitchUserid")) {

                // Set new Message-Data
                String newData = "switchedByMediation";
                System.out.println("Mediation: Setting MessageData to " + newData);
                msgRoot.setString("data/value", newData);
                msg.setDataGraph(data, SIApiConstants.JMS_FORMAT_TEXT);
            }
        }
    }
}

}
// Set Message-Userid
String newUserid = "Tester";
System.out.println("Mediation: Setting Userid to " + newUserid);
siMsgCtx.getSIMessage().setUserId(newUserid);
} else {
    System.out.println("Mediation: data/value not set");
} else {
    System.out.println("Mediation: Invalid Message Type " + msgfmt);
}

} catch (Exception e) {
    System.out.println("Mediation: " + e.toString());
}

// Continue processing this message
return true;


---

Defining a mediation in WebSphere Application Server

This following steps describe how to define a mediation in WebSphere Application Server using the admin console.

Note: The definitions described here are in addition to those described in “Definitions for the JMS test application” on page 81.

1. Deploy the mediation handler EJB (see “Deploying the SIB mediation handler” on page 100).

2. Create a mediation using the admin console, as follows:
   a. Select Service Integration → Buses → JMSBus → Mediations → New.
   b. Make the following entries:
      Mediation Name: Q2Mediation
      Handler List Name: MediationHandlerList
   c. Check “Allow concurrent mediation.”
      While this is not security-relevant, it is important for parallel execution of mediations and therefore affects throughput.

3. Define that messages on the SIB destination are to be mediated, as follows:
   a. Select Buses → JMSBus → Destinations.
      Select the checkbox in front of Q2, then select Mediate.
b. Select **MediationQ2** from the drop-down list.
c. Click **Next**.
d. Select **Busmember**.
e. Click **Next**.
f. Click **Finish**.

4. Save changes to the WebSphere Master Configuration and sync WebSphere Application Server nodes.

**Sample output from SIB mediation handler**

When the SIB mediation is active you can see messages like the ones shown in Example 7 in the WebSphere Application Server log.

**Example 7 Sample messages for a mediation in the WebSphere Application Server log**

Session writeJms: Running under: WSADMIN

Mediation: Userid = <WSADMIN>
Mediation-Text (IN): mediationSwitchUserid
Mediation: Setting MessageData to switchedByMediation
Mediation: Setting Userid to Tester

MDB: Running under: WSGUEST
MDB: JMSXUserID <Tester> / Text: switchedByMediation
BL: Running under: WSGUEST
MDB: BL returns: OK

**Set up a link between SIB and WebSphere MQ**

For the scenarios described in “WebSphere MQ with link to SIB” on page 23 and “SIB with link to WebSphere MQ” on page 26 a link between SIB and WebSphere MQ is necessary.

**Note:** The definitions described here are *in addition to* those described in “Definitions for the JMS test application” on page 81.

**Overview of the system**

The following artifacts existed for our sample:

- WebSphere Application Server with a defined SIB bus, as follows:
  
  Name: JMSBus
  SIB_MQ_ENDPOINT_ADDRESS: 38542 (see **Servers → Application Servers → <your application server> → Ports**
WebSphere MQ queue manager:

- Name: MQ4B
- TCP/IP port for incoming Connections: 1560 (The TCP/IP port is defined on the Channel Initiator of the WebSphere MQ queue manager.)

Definitions in WebSphere Application Server z/OS

Create the following additional definitions in WebSphere Application Server for the sample.

1. Create a foreign bus that represents the WebSphere MQ queue manager using the WebSphere Application Server admin console, as follows:
   a. Select Buses → JMSBus → Foreign Buses → New.
   b. Enter the Name: MQ4B (the name of the WebSphere MQ queue manager)
   c. Click Next.
   d. Enter the Routing Type: Direct WebSphere MQ Link
   e. Click Next.
   f. Enter the Inbound User ID: WSADMIN (used to authenticate inbound messages from WebSphere MQ and replaces the user ID stored in these messages)
   g. Enter the Outbound User ID: WSADMIN (used to authenticate outbound messages to WebSphere MQ and replaces the user ID stored in these messages)
   h. Click Next.
   i. Click Finish.

2. Define the MQ links between SIB and WebSphere MQ using the WebSphere Application Server admin console, as follows:
   a. Select Buses → JMSBus → Messaging Engines → <your messaging engine> → WebSphereMQ Links → New.
   b. Make the following entries:
      - Name: SIB.WMQ
      - Foreign Busname: MQ4B
      - Queue Manager Name: SIB (the name under which the SIB bus JMSBus is known to WebSphere MQ)
   c. Click Next.
   d. Make the following entries:
      - Sender MQ Channel Name: SIB.T0.WMQ
      - Hostname: wtsc48.itso.ibm.com (the TCP/IP Hostname)
Port: 1560 (defined in the WebSphere MQ Channel Initiator; default value is 1414)
Transport Chain: = “Outbound basic MQ Link”
e. Click **Next**.
f. Enter the Receiver MQ Channel Name: WMQ.T0.SIB
g. Click **Next**.
h. Click **Finish**.

3. Define the WebSphere MQ Foreign Destination (the WebSphere MQ queue) using the WebSphere Application Server admin console, as follows:
   a. Select **Buses** → **JMSBus** → **Destinations** → **New**.
   b. Select **Foreign**.
   c. Click **Next**.
   d. Enter the Identifier: JMSTESTQ (the name of the WebSphere MQ queue)
   e. For Bus, select: MQ4B

4. Create an SIB JMS Queue Destination in WebSphere Application Server, as follows:
   a. Select **Resources** → **Default Messaging** → **JMS Queue** → **New**.
   b. Make the following entries and selections:
      
      ```
      Name: TOWMQ
      JNDI name: jms/TOWMQ
      Bus name: MQ4B (select this before selecting the queue!)
      Queue name: JMSTESTQ
      ```

5. Create a WebSphere MQ JMS Queue Destination in WebSphere Application Server, as follows:
   a. Select **Resources** → **JMS Provider** → **WebSphere MQ** → **WebSphere MQ Queue Destinations**.
   b. Make the following entries:
      
      ```
      Name: TOSIB
      JNDI name: jms/TOSIB
      Base Queue name: TOSIB
      ```

6. Save the changes to the WebSphere master configuration and sync WebSphere Application Server nodes.

7. Make the necessary SIB security definitions in WebSphere Application Server. Use the wsadmin tool of WebSphere Application Server to do this.
   a. Determine the SOAP port of the Deployment Manager using the WebSphere Application Server admin console, by specifying
Check System Admin. → Deployment Manager → Ports → SOAP_CONNECTOR_ADDRESS.

b. In the <WAS_HOME>/bin directory of WebSphere Application Server on z/OS, enter the following command:

```
./wsadmin.sh -conntype SOAP -host <hostname of DepMgr.> -port <port of DepMgr.> -user <WAS Admin-User> -password <WAS Admin-User password>
```

c. At the wsadmin> prompt, type (or enter via Copy and Paste) the commands shown in Example 8.

**Example 8  wsadmin commands for SIB security**

```
wsadmin> $AdminTask addUserToForeignBusRole {-bus JMSBus -foreignBus MQ4B -role Sender -user WSADMIN}

wsadmin> $AdminTask addUserToDestinationRole {-type foreignDestination -bus JMSBus -foreignBus MQ4B -destination JMSTESTQ -role Sender -user WSADMIN}

wsadmin> $AdminTask addUserToDestinationRole {-type queue -bus JMSBus -foreignBus MQ4B -destination JMSTESTQ -role Sender -user WSADMIN}

wsadmin> $AdminConfig save

wsadmin> quit
```

**Setting up WebSphere MQ to exchange messages with SIB**

Use the following WebSphere MQ MQSC commands to define the necessary WebSphere MQ artifacts, using, for example, the MO71 tool. Select Action → MQSC window and enter the command into the top-line of the window.

**Example 9  WebSphere MQ definitions for connection to SIB**

```
DEF QL(SIB) USAGE(XMITQ)

DEFINE CHL(WMQ.TO.SIB) CHLTYPE(SDR) TRPTYPE(TCP)
CONNAME('wtsc48.itso.ibm.com(38542)') XMITQ(SIB)

DEFINE CHL(SIB.TO.WMQ) CHLTYPE(RCVR) TRPTYPE(TCP)

DEFINE QR(TOSIB) RNAME(Q2) RQMNAME(SIB) XMITQ(SIB)

START CHL(WMQ.TO.SIB)
```
Hints and tips to make it work

The following are some hints and tips to help you circumvent some common problems:

- For the first tests disable security in the SIB by selecting Service Integration → Buses → JMSBus and unchecking Secure.

- To get information about authentication problems, add the following custom property to the SIB bus definition:

  audit.bus.authentication=all

- If you see the message CWSIC3015E in the Adjunct Control Region job log, then the WebSphere MQ link SIB.MQ4B and the remote queue manager do not agree on the next message sequence number. For instance, a message with sequence number 3 has been received when sequence number 6 was expected. In this case, you should reset the WebSphere MQ/SIB channels.

Securing the link between SIB and WebSphere MQ via SSL

If WebSphere Application Server (by this, meaning SIB) and WebSphere MQ are on the same LPAR (like in our environment), there is no need for encryption of the TCP/IP traffic since it never leaves the LPAR. If WebSphere Application Server/SIB and WebSphere MQ are on different systems, then WebSphere Application Server/SIB and WebSphere MQ offer the possibility to also encrypt the TCP/IP traffic via SSL.

RunAs Thread Identity versus OS Thread Security and JMS

In Servers → Application Servers → <your application server> → Server security → z/OS Security options you can define two attributes:

- Support the synchronization of the OS thread.

- Enable the connection manager RunAs thread identity.

These two concepts in WebSphere Application Server on z/OS can easily be muddled up, so we clarify them in the next sections.

RunAs Thread Identity

*RunAs Thread Identity* specifies whether the identity of the Java thread executing, for example, an EJB request, is synchronized with the RunAs identity of the WebSphere Application Server z/OS user running this method. The operating system thread identity is kept unchanged and is typically the identity of the WebSphere server. Since some resource managers (DB2® for z/OS, IMS, CICS and WebSphere MQ in “BINDINGS” mode) depend on the Java thread identity for authentication and authorization you can use RunAs Thread Identity when making connection requests to these resource managers.
To enable RunAs Thread Identity, perform the following steps using the WebSphere Application Server admin console:

- Select **Servers → Application Servers → <your application server> → Server security → z/OS Security options**.
- Enable the attribute “Enable the connection manager RunAs thread identity.”
- Set the EJB resource reference for the WebSphere MQ Queue Connection Factory authentication to **Container** (see Figure 13).
- Ensure that no J2C Alias is defined in the WebSphere MQ Queue Connection factory.
- Ensure that WebSphere MQ Connection factory has defined Transport Type: **BINDINGS**.
- Install PK16189 (delivered with WebSphere Application Server Version 6.0.2.6). When RunAs Thread Identity is enabled and you to use the external resource, an **IllegalStateException** is encountered.

![Figure 13 Setting a resource reference to Authentication = Container](image-url)
OS Thread Security (Synchronization of the OS thread)

*OS Thread Security* specifies whether the identity of the z/OS thread (operating system thread identity) executing, for example, an EJB request, is synchronized with the J2EE identity of the WebSphere Application Server z/OS user running this method. Since non-WebSphere-managed resources normally depend on the z/OS thread identity for authentication and authorization, use OS Thread Security only when accessing such resources.

To enable OS Thread Security perform the following steps:

- Select **Servers → Application Servers → <your application server> → Server security → z/OS Security options.**
- Enable the attribute “Support the synchronization of the OS thread.”
- If you want to use this functionality, set the Environment Variable SyncToOSThread=true in the EJB DD (this is covered in detail in the next section).

**Enabling SyncToOS-Thread for an EJB**

To enable SyncToOS-Thread for an EJB, you have to set an EJB environment variable in the EJB DD, as shown in Example 10.

**Example 10  Setting the SyncToOS-Thread environment variable**

```xml
<env-entry>
  <description>SyncToOSThread Setting</description>
  <env-entry-name>com.ibm.websphere.security.SyncToOSThread</env-entry-name>
  <env-entry-value>true</env-entry-value>
  <env-entry-type>java.lang.Boolean</env-entry-type>
</env-entry>
```

The entry to the EJB DD can be added with RAD, as follows:

1. Open the Deployment Descriptor for the EJB Module holding the EJB you want to set this environment variable for.
2. Under the Bean tab select the EJB you want to set this variable for.
3. In the Environment Variables section click **Add**.
4. In the resulting pop-up window (Figure 14 on page 44), enter the following values:
   - **Name**: com.ibm.websphere.security.SyncToOSThread
   - **Description**: SyncToOSThread Setting
   - **Type**: Select **Boolean**
   - **Value**: true
Figure 14 Adding an EJB environment variable

5. Click **Finish**.
   Figure 15 on page 45 shows the EJB DD when the environment variable is set.
A tip to check whether you added the environment variable at the right location: if
the EJB environment variable is set in the EJB DD but SyncToOSThread is not
enabled on the server you will get the following message in the job log of the
WebSphere Application Server z/OS servant region:

error message: BBOJ0081W EJB WASSecJMSSession#JMSSession.jar#JMSSession
requests SyncToOSThread, but the server is not enabled for SyncToOSThread

A test with RunAs Thread Identity and OS Thread Security
We did some tests with RunAs Thread Identity and OS Thread Security using a
connection to WebSphere MQ in BINDINGS mode. Our system was set up as
follows:

- The user ID WebSphere Application Server runs under is ASSR1.
- The J2EE Identity of the EJB accessing WebSphere MQ is WSADMIN.
- The Downstream RunAs Identity of the EJB accessing WebSphere MQ is
  WSADMINX.
Test results
With the definitions described previously we got the test results shown in Table 2.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Application / WSADMIIZ</td>
<td>false</td>
<td>false</td>
<td>WSADMIIZ</td>
</tr>
<tr>
<td>Application / WSADMIIZ</td>
<td>false</td>
<td>true/false</td>
<td>WSADMIIZ WSADMIIZ</td>
</tr>
<tr>
<td>Application / WSADMIIZ</td>
<td>true</td>
<td>true/false</td>
<td>WSADMIIZ WSADMIIZ</td>
</tr>
<tr>
<td>Application / WSADMIIZ</td>
<td>false</td>
<td>false</td>
<td>WSADMIIZ</td>
</tr>
<tr>
<td>Application / none</td>
<td>false</td>
<td>false</td>
<td>ASSR1</td>
</tr>
<tr>
<td>Application / none</td>
<td>false</td>
<td>true/false</td>
<td>ASSR1 ASSR1</td>
</tr>
<tr>
<td>Application / none</td>
<td>true</td>
<td>true/false</td>
<td>ASSR1 ASSR1</td>
</tr>
<tr>
<td>Application / none</td>
<td>true</td>
<td>false</td>
<td>ASSR1</td>
</tr>
<tr>
<td>Container / WSADMIIZ</td>
<td>false</td>
<td>false</td>
<td>WSADMIIZ</td>
</tr>
<tr>
<td>Container / WSADMIIZ</td>
<td>false</td>
<td>true/false</td>
<td>WSADMIIZ WSADMIIZ</td>
</tr>
<tr>
<td>Container / WSADMIIZ</td>
<td>true</td>
<td>true/false</td>
<td>WSADMIIZ WSADMIIZ</td>
</tr>
<tr>
<td>Container / WSADMIIZ</td>
<td>true</td>
<td>false</td>
<td>WSADMIIZ</td>
</tr>
<tr>
<td>Container / none</td>
<td>false</td>
<td>false</td>
<td>ASSR1</td>
</tr>
<tr>
<td>Container / none</td>
<td>false</td>
<td>true/false</td>
<td>ASSR1 ASSR1</td>
</tr>
<tr>
<td>Container / none</td>
<td>true</td>
<td>true/false</td>
<td>WSADMIIX WSADMIIX</td>
</tr>
<tr>
<td>Container / none</td>
<td>true</td>
<td>false</td>
<td>WSADMIIX</td>
</tr>
</tbody>
</table>
The JMS test application

To test security, we created the following artifacts:

- A session EJB (the JMSSession EJB) sending and receiving JMS messages, discussed in “JMSSession EJB” on page 48.
- A session EJB client calling the previously mentioned session EJB with a specified set of test cases, discussed in “JMSSession EJB client” on page 55.
- A Message-Driven Bean (MDB) EJB, discussed in “Message-Driven Bean (MDB)” on page 60.
- A helper class to switch the WebSphere Application Server z/OS JAAS Subject based on User ID and password or User ID and passticket, discussed in “Java helper class for JAAS login” on page 74.
- An SIB mediation handler. See “Service Integration Bus mediation handlers” on page 28.
- A Session EJB (the BusinessLogic session EJB) that simulates the business logic and is called by the JMSSession EJB and the MDB to verify the RunAs identity established by these two EJBs. This component is discussed in “BusinessLogic session EJB” on page 81.

Note: The examples used in this paper are based on sample applications that can be downloaded from the IBM Redbooks Web site. See “Downloadable material” on page 109 for details.

Tips for developing with RAD

In this section we provide some useful tips to consider before you start working in RAD on the samples or on your own similar code.

Define secure server in RAD WebSphere test environment

If you want to test security in your development environment you need a secured WebSphere test environment in RAD. To establish this, double-click your WebSphere server instance in the Servers view of RAD and select “Security is enabled on this server.”
BINDINGS mode and WebSphere MQ not on z/OS

When testing security in BINDINGS mode on non-z/OS platforms the user ID specified as J2C Alias must be the user ID under which the WebSphere Application Server process is running. Otherwise, you get a WebSphere MQ exception when you try to authenticate against WebSphere MQ.

NullPointerException while doing programmatic login

If you get a NullPointerException when doing a programmatic login to WebSphere Application Server using a JAAS subject, make sure that your WebSphere server has security activated.

JMSSession EJB

The purpose of the JMSSession EJB is to do JMS tests (send and receive messages), and then to create messages to be consumed by MDBs.

The JMSSession EJB offers the following functions:

- Sending JMS messages
- Receiving JMS messages
Sending JMS messages
The JMSSession EJB can perform the following functions related to sending JMS messages:

- Send a JMS message using the Connection Factory with default user ID and password.
- Send a JMS message using the Connection Factory with specified user ID and password. This requires that the resource reference of the QueueConnectionFactory used by the JMSSession EJB is set to Authentication = Application. See Figure 17 on page 50.
- Optionally, switch the RunAs identity based on user ID/password.
- Optionally, switch the RunAs identity based on user ID/passticket.
- Optionally, add a set of JMS properties to the JMS message.
- Call the BusinessLogic session EJB which simulates the business logic to check the RunAs identity established by the JMSSession EJB.
Receiving JMS messages
TheJMSSessionEJBcanperformthefollowingfunctionsforreceivingJMS
messages:

- Receive a JMS message using the Connection Factory with default user ID
  and password.

- Receive a JMS message using the Connection Factory with specified user ID
  and password. This requires that the resource reference of the
  QueueConnectionFactory used by the JMSSession EJB is set to
  Authentication = Application. See also Figure 17 on page 50.

- Find out who created the message, based on the JMSXUserID JMS property.
Use a JMS selector (for example, for security) based on JMS message properties.

**JMSSession EJB code**

Example 11 shows the Java code for the JMSSession EJB to write JMS messages and Example 12 on page 54 shows the Java code for the JMSSession EJB to read JMS messages.

**Example 11  Session EJB writeJms**

```java
public String writeJms (String cfUserid, String cfPassword, String jaasUserid, String jaasPassword,
  String jmsCF, String jmsQ, String msgText, String[] jmsPropertyNames, Object[] jmsPropertyValues) {

  InitialContext ctx = null;
  QueueConnectionFactory factory = null;
  QueueConnection connection = null;
  QueueSession session = null;
  Queue ioQueue = null;
  QueueSender queueSender = null;

  try {

    System.out.println("Session writeJms: Running under: "+getSessionContext().getCallerPrincipal());

    if (jaasUserid != null) {
      JmsSecurityHelper secHelper = new JmsSecurityHelper();

      if (jaasPassword != null) {
        if (!secHelper.switchUserId(jaasUserid, jaasPassword)) {
          return "Security-Problem!";
        }
      } else {
        if (!secHelper.switchUserIdUsingPassticket(jaasUserid)) {
          return "Security-Problem (Passticket)!";
        }
      }

      System.out.println("Session writeJms: Running now under: "+getSessionContext().getCallerPrincipal());
    } // end-if

    ctx = new InitialContext();

    factory = (QueueConnectionFactory) ctx.lookup(jmsCF);

    // Code to write JMS message

  } catch (NamingException e) {
    System.out.println(e.toString());
    return "Security-Problem!");
}
```

Java Messaging Service Security on z/OS  51
// Do we want to specify our own Userid / Password for the connection?
// => resAuth for the Resource-Reference in the DD must be set to "Application"
if (cfUserid != null) {
    connection = factory.createQueueConnection(cfUserid, cfPassword);
} else {
    connection = factory.createQueueConnection();
}

connection.start();
session = connection.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);
ioQueue = (Queue) ctx.lookup(jmsQ);

TextMessage outMessage = session.createTextMessage();

// Set JMS Properties for Message-Selector
if (jmsPropertyNames != null && jmsPropertyNames.length != 0) {
    for (int i = 0; i < jmsPropertyNames.length; i++) {
        if (jmsPropertyValues[i] instanceof String ) {
            outMessage.setStringProperty(jmsPropertyNames[i], (String) jmsPropertyValues[i]);
            System.out.println("Session writeJms:" + jmsPropertyNames[i] + " / " + (String) jmsPropertyValues[i]);
            continue;
        }
        if (jmsPropertyValues[i] instanceof Integer ) {
            outMessage.setIntProperty(jmsPropertyNames[i], ((Integer) jmsPropertyValues[i]).intValue());
            System.out.println("Session writeJms:" + jmsPropertyNames[i] + " / " + (Integer) jmsPropertyValues[i]);
            continue;
        }
    // We could continue here with Long , Short, Boolean, ...
    return "Invalid Message Selector-Property";
    }
}

outMessage.setText(msgText);
queueSender = session.createSender(ioQueue);
queueSender.send(outMessage);

//
// Just test the RunAs-Identity we might have set up above
// by calling another EJB
//
try {
    InitialContext ivjInitContext = null;

    BusinessLogicejb = null;
BusinessLogicHome ejbHome = null;

String ejbName = "ejb/ejbs/BusinessLogicHome";

ivjInitContext = new InitialContext();

java.lang.Object homeObject = ivjInitContext.lookup(ejbName);
if (homeObject == null) {
    System.out.println("MDB: BusinessLogic lookup failed");
}

ejbHome = (BusinessLogicHome)
javax.rmi.PortableRemoteObject.narrow((org.omg.CORBA.Object)
    homeObject, BusinessLogicHome.class);

ejb = ejbHome.create();

System.out.println("Session: BusinessLogic returns: " + ejb.doWork(msgText));
}
catch (Exception e) {
    System.out.println("Session " + e.toString());
    return "Calling BusinessLogic failed " + e.toString();
}

return "OK";

}

catch (JMSException je) {
    System.out.println("Session writeJms: JMS Exception: " + je.toString());
    Exception e = je.getLinkedException(); // get MQException if possible
    if (e != null) {
        System.out.println("Session writeJms: JMS Exception (linked exception): " +
            e.toString());
    }
    return "Session writeJms: JMS Exception: " + je.toString();
}

catch (Exception e) {
    System.out.println("Session writeJms: General Exception: " + e.toString());
    return "Session writeJms: General Exception: " + e.toString();
}

finally {
    try { if (queueSender != null) {queueSender.close();} } catch (Exception e) {};
    try { if (session != null) {session.close();} } catch (Exception e) {};
    try { if (connection != null) {connection.close();} } catch (Exception e) {};
}
Example 12  Session EJB readJms

```java
public String readJms (String cfUserid, String cfPassword, String jmsCF, String jmsQ, String messageSelector) {

    System.out.println("Session readJms: Running under: " +
    getSessionContext().getCallerPrincipal());

    InitialContext ctx = null;
    QueueConnectionFactory factory = null;
    QueueConnection connection = null;
    QueueSession session = null;
    Queue ioQueue = null;
    QueueReceiver queueReceiver = null;

    try {
        ctx = new InitialContext();

        factory = (QueueConnectionFactory) ctx.lookup(jmsCF);

        // Do we want to specify our own Userid / Password for the connection?
        // => resAuth for the Resource-Reference in the DD must be set to "Application"
        if (cfUserid != null) {
            connection = factory.createQueueConnection(cfUserid, cfPassword);
        } else {
            connection = factory.createQueueConnection();
        }

        connection.start();
        session = connection.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);
        ioQueue = (Queue) ctx.lookup(jmsQ);

        // Do we want to use a JMS Message Selector?
        if (messageSelector != null) {
            queueReceiver = session.createReceiver(ioQueue, messageSelector);
        } else {
            queueReceiver = session.createReceiver(ioQueue);
        }

        TextMessage inMessage = (TextMessage) queueReceiver.receiveNoWait();

        if (inMessage == null) {
            return "No Message available";
        } else {
            return "Session readJMS: JMSXUserID " + inMessage.getStringProperty("JMSXUserID") + " / Text: " + inMessage.getText();
        }
    }

    catch (JMSException je) {
```
System.out.println("Session readJms: JMS Exception: " + je.toString());
Exception e = je.getLinkedException(); // get MQException if possible
if (e != null) {
    System.out.println("Session readJms: JMS Exception (linked exception): " + e.toString());
    return "Session readJms: JMS Exception: " + je.toString();
}
catch (Exception e) {
    System.out.println("Session readJMS General Exception: " + e.toString());
    return "Session readJMS General Exception: " + e.toString();
}
finally {
    try { if (queueReceiver != null) {queueReceiver.close();} } catch (Exception e) {};
    try { if (session != null) {session.close();} } catch (Exception e) {};
    try { if (connection != null) {connection.close();} } catch (Exception e) {};
}

JMSSession EJB client

The goal of the JMSSession EJB client is to trigger the JMSSession EJB with a predefined set of test cases.

The following test cases are implemented:

- Send a JMS message using a default JMS Connection Factory user ID.
- Send a JMS message using a specified JMS Connection Factory user ID and password.
- Switch the JAAS Subject in the session EJB using user ID and password before sending the JMS message.
- Switch the JAAS Subject in the session EJB using user ID and passticket before sending the JMS message.
- Write a JMS message that asks the MDB to switch RunAs identity using user ID and password.
- Write a JMS message that asks the MDB to switch RunAs identity using user ID and passticket.
- Write a JMS message with JMS message properties to test a JMS message selector.
- Write a JMS message that asks SIB mediation to switch JMS message user ID and JMS message data.
- Receive a JMS message using the default JMS Connection Factory user ID.
- Receive a JMS message using a specified JMS Connection Factory user ID and password.
- Receive a JMS message using a JMS message selector.
- Send a JMS message from SIB to WebSphere MQ and receive it from there.
- Send a JMS message from WebSphere MQ to SIB and receive it from there.

**Note:** In the client code, set the JMS implementation you want to use (variable `useSIB = true or false`) and uncomment the test cases you want to run.

**JMSSession EJB client code**

Example 13 shows the Java code for the JMSSession EJB client.

```java
Example 13 JMSSession EJB client code
import java.util.Properties;
import javax.naming.InitialContext;
import ejbs.JMSSession;
import ejbs.JMSSessionHome;

public class Main {
  public static void main(String[] args) {
    try {
      InitialContext ivjInitContext = null;

      JMSSession ejb = null;
      JMSSessionHome ejbHome = null;

      String initialContext = "com.ibm.websphere.naming.WsnInitialContextFactory";
      String ejbName = "ejb/ejbs/JMSSessionHome";

      Properties properties = new Properties();
      properties.put(javax.naming.Context.INITIAL_CONTEXT_FACTORY, initialContext);
      ivjInitContext = new InitialContext(properties);

      java.lang.Object homeObject = ivjInitContext.lookup(ejbName);
      if (homeObject == null) {
        System.out.println("Client: EJB lookup failed");
      }
    }
  }
}
```
```java
ejbHome = (JMSSessionHome)
javax.rmi.PortableRemoteObject.narrow((org.omg.CORBA.Object) 
    homeObject, JMSSessionHome.class);

ejb = ejbHome.create();

String qcf = null;
String q = null;

// Define if we want to use WMQ or SIB for our tests
boolean useSIB = false;

if (useSIB) {
    qcf = "java:comp/env/jms/QM2";
    q = "java:comp/env/jms/Q2";
} else {
    qcf = "java:comp/env/jms/QM1";
    q = "java:comp/env/jms/Q1";
}

System.out.println("useSIB: " + useSIB);

for (int i = 0; i < 1; i++) {
    System.out.println("Client: Iteration: " + i);
    System.out.println("Client: Calling JMS Writer:");
    
    // Write JMS Message
    
    // Use Default CF-Userid
    // System.out.println("*** WT1 *** " + ejb.writeJms(null, null, null, null, qcf, q, "TestText", null, null));

    // Specify CF-Userid
    // Limitation: When using BINDINGS-Mode on WAS non-z/OS we need to specify here the userid & password WAS is running under...
    // For WMQ z/OS Userid and Password need to be specified UPPERCASE!
    // System.out.println("*** WT2 *** " + ejb.writeJms("WSADMIN", "WSADMIN", null, null, qcf, q, "TestText", null, null));

    // Switch JAAS-Subject in Session-EJB using Userid & Password
    // System.out.println("*** WT3 *** " + ejb.writeJms(null, null, "WSADMIN", "WSADMIN", qcf, q, "TestText", null, null));

    // Switch JAAS-Subject in Session-EJB using Passticket
```
System.out.println("*** WT4 *** " + ejb.writeJms(null, null, "WSADMIX", null, qcf, q, "TestText", null, null));

// Ask MDB to switch JAAS-Subject using Userid & password
// System.out.println("*** WT5 *** " + ejb.writeJms(null, null, null, null, qcf, q, "switchUseridPassword", null, null));

// Ask MDB to switch JAAS-Subject using Passticket
// System.out.println("*** WT6 *** " + ejb.writeJms(null, null, null, null, qcf, q, "switchPassticket", null, null));

// Write JMS Message with properties to test MessageSelector
<!--[if !IE]>]*
String[] names = new String[3];
names[0] = "businessCase";
names[1] = "customerRating";
names[2] = "amount";

Object[] values = new Object[3];
values[0] = new String("withdraw");
values[1] = new String("silver");
values[2] = new Integer(200000);

System.out.println("*** WT7*** " + ejb.writeJms(null, null, null, null, qcf, q, "TestText", names, values));
/*

// Ask Mediation to switch Userid & Messagedata
// System.out.println("*** WT8 *** " + ejb.writeJms(null, null, null, null, qcf, q, "mediationSwitchUserid", null, null));

// Read JMS Message
//

System.out.println("Client: Calling JMS Reader: ");

// Use Default CF-Userid
// System.out.println("*** RT1 *** " + ejb.readJms(null, null, qcf, q, null));

// Specify CF-Userid
// Limitation: When using BINDINGS-Mode on WAS non-z/OS we need to specify here
// the userid & password WAS is running under...
// For WMQ z/OS Userid and Password need to be specified UPPERCASE!
// System.out.println("*** RT2 *** " + ejb.readJms("WSADMIN", "WSADMIN", qcf, q, null));

// Use JMS Message Selector
// System.out.println("*** RT3 *** " + ejb.readJms(null, null, qcf, q,
// "businessCase = 'withdraw' AND customerRating <> 'gold' AND amount > 100000"));

//
// Test Connection from SIB to WMQ
//
System.out.println("Client: Calling SIB -> WMQ: ");
/*
qcf= "java:comp/env/jms/QM2"; // SIB
q = "java:comp/env/jms/TOWMQ"; // "Alias" pointing to WMQ
System.out.println("*** TOWMQ-WT1 *** " + ejb.writeJms(null, null, null, null, qcf, q, "TestText", null, null));
*/

// Wait a little for the message to be transferred
// Remember to stop the JMS Listener / Activation Spec in the system otherwise
// the message may already be consumed before we try to read it!
// Thread.sleep(5000);
/*
qcf= "java:comp/env/jms/QM1"; // WMQ Queuemanager
q = "java:comp/env/jms/Q1"; // "Local Queue" in WMQ
System.out.println("*** TOWMQ-RT1 *** " + ejb.readJms(null, null, qcf, q, null));
*/

//
// Test Connection from WMQ to SIB
//
System.out.println("Client: Calling WMQ -> SIB: ");
/*
qcf= "java:comp/env/jms/QM1"; // WMQ
q = "java:comp/env/jms/TOSIB"; // Remote-Queue pointing to SIB
System.out.println("*** TOSIB-WT1 *** " + ejb.writeJms(null, null, qcf, q, "TestText", null, null));
*/

// Wait a little for the message to be transferred
// Remember to stop the JMS Listener / Activation Spec in the system otherwise
// the message may already be consumed before we try to read it!
// Thread.sleep(5000);
/*
qcf= "java:comp/env/jms/QM2"; // SIB Queuemanager
q = "java:comp/env/jms/Q2"; // "Local Queue" in SIB
System.out.println("*** TOSIB-RT1 *** " + ejb.readJms(null, null, qcf, q, null));
*/
Message-Driven Bean (MDB)

The Message-Driven Bean can perform the following functions:

- Display the user ID the MDB is running under.
- Display the message that was received via JMS.
- If requested via the JMS message, switch the RunAs identity using user ID and password.
- If requested via the JMS message, switch the RunAs identity using user ID and passticket.
- Call the BusinessLogic session EJB, which simulates the business logic to check the RunAs identity established by the MDB.

MDB out of the box

An MDB is installed “out of the box.” Its `onMessage()` method is invoked by WebSphere Application Server on z/OS for every JMS message on a JMS destination. The link between the JMS destination and the MDB is created by a JMS Listener (if the JMS provider is WebSphere MQ) or by a JMS Activation Specification (if the JMS provider is SIB). In the MDB Deployment Descriptor the application developer can propose to the application deployer which JMS Listener or Activation Specification should be used for this MDB.

**MDB Deployment Descriptor with WebSphere MQ JMS listener**

To specify a listener for WebSphere MQ, follow these steps, as illustrated in Figure 18 on page 61:

1. Open the Deployment Descriptor for the MDB you want to define the JMS listener for.
2. Under the Bean tab select **MDB**.
3. In the WebSphere Bindings section select Listener Port, and in the ListenerPort name field enter the name of the JMS listener defined in WebSphere Application Server. The JMS listener in WebSphere Application Server then has defined which JMS destination should be monitored.

![Defining an MDB JMS listener](image)

Figure 18  Defining an MDB JMS listener

If you do not have RAD to set the name of the JMS listener, the Deployment Descriptor of the EJB Module holding the MDB should be set up as shown in Example 14.
Example 14  EJB module DD ibm-ejb-jar-bnd.xmi for JMS listener

```xml
<ejbBindings xmi:type="ejbbnd:MessageDrivenBeanBinding"
xmi:id="MessageDrivenBeanBinding_1153146010330" listenerInputPortName="JMSListener">
  <enterpriseBean xmi:type="ejb:MessageDriven" href="META-INF/ejb-jar.xml#MDB"/>
</ejbBindings>
```

**MDB Deployment Descriptor with SIB Activation Specification**

To specify an Activation Specification for SIB, follow these steps, as illustrated in Figure 19 on page 63:

1. Open the Deployment Descriptor for the MDB you want to define the JMS Activation Specification for.
2. Under the Bean tab select **MDB**.
3. In the WebSphere Bindings section select JCA Adapter and in the ActivationSpec JNDI Name field enter the JNDI-name of the SIB Activation Specification defined in WebSphere Application Server. The SIB Activation Specification in WebSphere Application Server then has defined which JMS queue should be monitored. When defining an Activation Specification in the Deployment Descriptor, you can specify two more attributes:
   - ActivationSpec Authorization Alias
     This is the J2C authentication alias used by the Activation Specification when accessing SIB.
   - Destination JNDI Name
     This is the name of the SIB JMS destination the MDB should use.

Normally, these two attributes should not be set, so that the WebSphere Application Server administrator has control over them.
If you do not have RAD to set the attributes of the Activation Specification, the Deployment Descriptor of the EJB Module holding the MDB should be set up as shown in Example 15.

Example 15  EJB module DD ibm-ejb-jar-bnd.xmi for Activation Specification

```xml
  <ejbJar href="META-INF/ejb-jar.xml#ejb-jar_ID"/>
  <ejbBindings xmi:type="ejbnd:MessageDrivenBeanBinding"
               xmi:id="MessageDrivenBeanBinding_1153146010330" activationSpecJndiName="jms/ACT1"
               activationSpecAuthAlias="WSADMIN" destinationJndiName="jms/Q2">
    <enterpriseBean xmi:type="ejb:MessageDriven" href="META-INF/ejb-jar.xml#MDB"/>
  </ejbBindings>
</ejbnd:EJBJarBinding>
```
Use of a JMS message selector

From within an MDB you can use a JMS message selector for security since you can, for example, specify that this MDB should only be invoked for JMS messages coming from a specified user. To do this you need to define a JMS message selector based on the JMS user ID that created the message (stored in the JMS message property JMSXUserID).

Use the following steps to do this:

1. Open the EJB Deployment Descriptor of the MDB in the Bean tab.
2. In the Activation Configuration section click Add. This will bring up the window shown in Figure 20.

![Figure 20 Setting JMS message selector](image)

3. In the Name field select messageSelector, in the Value field enter ‘<userid>’, where <userid> is the user ID that sends the JMS message.
4. Click Finish. The MDB Deployment Descriptor will now look as shown in Figure 21 on page 65.

**Note:** In WebSphere MQ, the user ID (the JMS message property JMSXUserID) is always 12 characters long, so if the user ID you are using is shorter you have to pad it to the right with blanks.
If you do not have RAD, the Deployment Descriptor of the EJB Module holding the MDB should be set up as shown in Example 16.

**Example 16  EJB Module DD ejb-jar.xml**

```xml
<message-driven id="MDB">
...
<activation-config>
    <activation-config-property>
        <activation-config-property-name>messageSelector</activation-config-property-name>
        <activation-config-property-value>MS_user_ID=Address</activation-config-property-value>
    </activation-config-property>
</activation-config>
```

**MDB Deployment Descriptor with JMS message selector**

If you do not have RAD, the Deployment Descriptor of the EJB Module holding the MDB should be set up as shown in Example 16.
Setting the RunAs Deployment Descriptor

Up to now all the MDBs and all invoked EJBs (for example, for the business logic) run under the same technical user WebSphere Application Server z/OS uses for MDBs. To define the RunAs identity of the MDB (which will also set the user ID EJBs called by the MDB run under) do the following:

1. Select the Assembly tab of the Deployment Descriptor of the MDB.
2. In the Security Role section click Add. This brings up the window shown in Figure 22.

![Add Security Role](image)

Figure 22 Adding a security role

a. Make the following entries:
   - Name: mdbuser
   - Description: User running the MDB

b. Click Finish. The EJB DD Assembly tab will now look as shown in Figure 23.
3. Select the Access tab of the Deployment Descriptor of the MDB.

4. In the Security Identity (Bean Level) section click **Add**. This will bring up the window shown in Figure 24 on page 68.
5. Select “Use identity assigned to specific role” and select `mdbuser`.
6. Click **Next**. The next window is as shown in Figure 25 on page 69.
7. In the Security Identity window, select the MDB you want this identity to be associated with.

8. Click **Finish**. The EJB DD Access tab now looks as shown in Figure 26 on page 70.
9. Click **Finish**.

10. Open the Deployment Descriptor of the Enterprise Application (.ear file) the MDB is contained in and select the Security tab.

11. Click **Gather** to collect the roles defined in the EJB modules contained in this Enterprise Application.

12. Select **mdbuser**. In the Security Role RunAs Bindings section click **Add**. Enter the user ID and password of the user you want to be used as RunAs identity of the MDB.

    The EJB DD Security tab will now look as shown in Figure 27 on page 71.
MDB Deployment Descriptor source for RunAs

If you do not have RAD to set the Deployment Descriptors as described previously, the EJB Module holding the MDB should be set up as shown in Example 17.

Example 17  EJB Module DD ejb-jar.xml

```xml
...<message-driven id="MDB">
    <security-identity id="RunAsSpecifiedIdentity_1155122713713">
        <description></description>
        <run-as>
            <description></description>
            <role-name>mdbuser</role-name>
        </run-as>
    </security-identity>
...```

Figure 27  EAR Deployment Descriptor Security tab
The Deployment Descriptor of the J2EE Application (the .ear file) should look similar to Example 18 and the extensions should look similar to Example 19.

Example 18  J2EE application DD 'application.xml'

```
<security-role id="SecurityRole_1155117397889">
  <description>
    User running the MDB
  </description>
  <role-name>mdbuser</role-name>
</security-role>
```

Example 19  J2EE application DD - extension "ibm-application-bnd.xmi"

```
<applicationbnd:ApplicationBinding xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
  xmlns:applicationbnd="applicationbnd.xmi" xmlns:commonbnd="commonbnd.xmi"
  xmi:id="ApplicationBinding_1155117397879">
  <application href="META-INF/application.xml#Application_ID"/>
  <runAsMap xmi:id="RunAsMap_1155117397879">
    <runAsBindings xmi:id="RunAsBinding_1155117397879">
      <authData xmi:type="commonbnd:BasicAuthData" xmi:id="BasicAuthData_1155117397879"
        userId="mquser" password="{xor}Mi4qLDotaz4="/>
      <securityRole href="META-INF/application.xml#SecurityRole_1155117397889"/>
    </runAs_bindings>
  </runAsMap>
</applicationbnd:ApplicationBinding>
```

**Effect of the MDB RunAs identity**

**Important:** The RunAs identity of the MDB is *not* the identity that the MDB itself runs under, but the identity *all EJBs called by the MDB* run under. The MDB itself runs under a WebSphere Application Server user ID.

Example 20 and Example 21 show the output of two tests, one run without RunAs identity set, and the other with it set.
Example 20  
**MDB and business logic session EJB without RunAs identity of the MDB**

MDB: Running under: WSGUEST  
BL: Running under: WSGUEST  
MDB: BusinessLogic returns: OK

---

Example 21  
**MDB and business logic session EJB with RunAs identity of the MDB set to WSADMIN**

MDB: Running under: WSGUEST  
BL: Running under: WSADMIN  
MDB: BusinessLogic returns: OK

---

**Message-Driven Bean code**

Example 22 shows the Java code of the `onMessage()` method in the MDB.

---

**Example 22  MDB onMessage method**

```java
public void onMessage(javax.jms.Message msg) {

    System.out.println("MDB: Running under: " +
    getMessageDrivenContext().getCallerPrincipal());

    TextMessage inMessage = (TextMessage) msg;

    try {
        System.out.println("MDB: JMSXUserID <" + inMessage.getStringProperty("JMSXUserID") +
        "> / Text: " + inMessage.getText());

        // Switch Userid for Business-Logic using Userid & Password
        if (inMessage.getText().equals("switchUserIdPassword")) {
            JmsSecurityHelper secHelper = new JmsSecurityHelper();
            if (!secHelper.switchUserId("WSADMIZ", "WSADMIZ")) {
                System.out.println("MDB: Security-Problem Userid/Password!");
            }
        }

        // Switch Userid for Business-Logic using generated Passticket
        if (inMessage.getText().equals("switchPassticket")) {
            JmsSecurityHelper secHelper = new JmsSecurityHelper();

            // Use a hard-coded userid here for tests - could
            // be also the userid stored in the message ("JMSXUserID"-Property)
            String userid = "WSADMIX";

            if (!secHelper.switchUserIdUsingPassticket(userid)) {
                System.out.println("MDB: Security-Problem Passticket!");
            }
        }
    }
}
```
InitialContext ivjInitContext = null;

BusinessLogicejb = null;
BusinessLogicHomeejbHome = null;

String ejbName = "ejb/ejbs/BusinessLogicHome";

ivjInitContext = new InitialContext();

java.lang.Object homeObject = ivjInitContext.lookup(ejbName);
if (homeObject == null)
{
    System.out.println("MDB: BusinessLogic lookup failed");
}

ejbHome = (BusinessLogicHome)javax.rmi.PortableRemoteObject.narrow((org.omg.CORBA.Object)homeObject, BusinessLogicHome.class);

ejb = ejbHome.create();

System.out.println("MDB: BusinessLogic returns: " + ejb.doWork(inMessage.getText()));

} catch (Exception e) {
    System.out.println("MDB: " + e.toString());
}

return;
}

---

**Java helper class for JAAS login**

The following sections discuss the Java help class.

**Switch user ID based on information passed in message**

To be able to pass the identity of the user ID (JMSXuserID) in the message to EJBs called by the MDB, we implemented a JAAS login in the Message-Driven Bean. When the business logic session bean is scheduled it runs under the new user ID; therefore, normal J2EE role-based authorization can take place.

With this implementation we are able to handle user ID propagation of asynchronous requests the same way as synchronous requests from RMI/IIOP or a servlet component.
JAAS or Java Authentication and Authorization Service is a standard component in WebSphere and is a strategic API for authorization and authentication. WebSphere ships with a set of JAAS login modules that are used by various components in WebSphere to perform login. Different JAAS login modules are used for different types of authentication (user ID and password, x509 certificate, and so forth) and for different channels such as servlets, RMI/IIOP, and Web services using Web Services Security.

A JAAS login can be done programmatically from an application program to create a new subject, and with proper authorization the application can change to the new subject and run the code under a new user ID.

In addition to the JAAS login modules shipped with WebSphere you can develop your own JAAS custom login modules.

Login using user ID and password
Probably the most used JAAS login module is WSLogin, which requires user ID and password as input to create a new subject. In our MDB example, for the first tests we used a fixed password and a fixed user ID to do the login.

The following is an example scenario:
1. The MDB is triggered by the listener port in WebSphere.
2. The MDB extracts the user ID (JMSXUserID) and some other attributes (for example, the message data) from the message and maps it (for example, based on some business decision) to a “technical user” that the MDB knows the user ID and password of.
3. The MDB calls a JAAS login module with the user ID and password of the technical user.
4. The MDB calls the business logic session EJB with RunAs set to the (RACF) user ID of the technical user used in the previous step.
5. The business logic session EJB runs under the RACF user ID of the technical user who was mapped to the user ID of the user that created the JMS message.

This approach has the drawback that you need to know user ID and password of at least the technical user when doing the login. But this approach is still more flexible than setting the RunAs identity in the MDB Deployment Descriptor because even if you define a fixed number of (technical) users to be used by the MDB (where you know user ID and password), you only need to deploy this MDB once.
Code for login using user ID and password

Example 23 shows the Java code for a JAAS login using user ID and password.

Example 23   Login using user ID and password

```java
public final boolean switchUserId(String userid, String password) {
    LoginContext lc = null;
    try {
        lc = new LoginContext("WSLogin",
                new WSCallbackHandlerImpl(userid, password));
    }
    catch (LoginException le) {
        System.out.println("Cannot create LoginContext. " + le.getMessage());
        return false;
    }
    catch (SecurityException se) {
        System.out.println("Cannot create LoginContext." + se.getMessage());
        return false;
    }
    try {
        lc.login();
    } catch (LoginException le) {
        System.out.println("Fails to create Subject. " + le.getMessage());
        return false;
    }

    // After successfull login switch to the new user or subject...
    Subject s = lc.getSubject();
    try {
        if (s != null) {
            System.out.println("LoginContext " + s);
        } else {
            System.out.println("LoginContext is null!");
        }
        WSSubject.setRunAsSubject(s);
    } catch (WSSecurityException le) {
        System.out.println("Fails to create Subject. " + le.getMessage());
        return false;
    }
    return true;
}
```
Login using RACF passticket

**Note:** The following scenario is based on functionality that is only available in z/OS 1.7 or later. It is a JNI-based Java wrapper to the RACF functions r_ticketserv and r_gensec and is delivered as a Java .jar file (IRRRAcf.jar, in our installation located in /usr/include/java_classes), a .so file (libIRRRAcf.so, in our installation located in /usr/lib) and the documentation (IRRRAcfDoc.jar, in our installation located in /usr/include/java_classes).

In most MDB scenarios, you will not have a password for the JMSXUserID, so the previous scenario (using user ID and password to log in) does not work. We combined use of WSLogin with the RACF PASSTICKET function to generate a passticket and used it as a one time password.

The following is an example scenario:

1. The MDB is triggered by the listener port in WebSphere.
2. The MDB extracts the (RACF) user ID (JMSXUserID) from the message.
3. The MDB generates a passticket for the RACF user ID, using special code that can generate the passticket.
4. The MDB calls a JAAS login module with the (RACF) user ID sent in the message and the passticket generated in the previous step.
5. The MDB calls the business logic session EJB with RunAs set to the (RACF) user ID used in the previous step.
6. The business logic session EJB runs under the RACF user ID of the user who created the JMS message.

**Code for login using RACF passticket**

Example 24 shows the Java code for a JAAS login using a RACF passticket.

**Example 24  Login using RACF passticket**

```java
public boolean switchUserIdUsingPassticket(String userid) {
    IRRPassTicket passTicket;
    // Sample appl info..
    String appl = "CBS390";
    String password = null;
    try {
        passTicket = new IRRPassTicket();
        System.out.println("userid=<" + userid + "> appl=<" + appl + ">");
        // Generate new PassTicket.
        password = passTicket.generate(userid, appl);
    }
```
System.out.println("New PassTicket Password: " + password);
} catch (IRRPassTicketGenerationException bx) {
    System.out.println("Generation: IRR Exception caught: " + bx);
    bx.printStackTrace();
    return false;
}

try {  
    passTicket.evaluate(userid,appl,password);
    System.out.println("Evaluation successful..";

    // Catch eval or generation failures.
} catch (IRRPassTicketEvaluationException ax) {
    System.out.println("Evaluation: IRR Exception caught: " + ax);
    ax.printStackTrace();
    return false;
}

// Just reuse the login with userid & password...
return switchUserId(userid, password);
**WebSphere Application Server definitions for login using RACF passticket**

To be able to use RACF passtickets, the following definitions have to be made in WebSphere Application Server:

1. Add the passticket .jar file to the classpath of the JVM, as follows:
   
   a. Select **Servers** → **Application Servers** → `<your application server>` → **Java and Process Management** → **Process Definition** → **Servant** → **Java Virtual Machine**.
   
   b. Add the fully qualified passticket .jar file to the classpath property on this panel (for example, `/usr/include/java_classes/IRRRacf.jar`).

2. Add the passticket .so file to the LIBPATH of the JVM, as follows:
   
   a. Select **Servers** → **Application Servers** → `<your application server>` → **Java and Process Management** → **Process Definition** → **Servant** → **Custom Properties**.
   
   b. Make one of the following entries:
      
      - If a customs property named LIBPATH already exists, add the directory where the passticket .so file is stored to the existing LIBPATH, separated by a colon (for example, `/SC48/var/itcam/wsam/pesr01a/lib:/usr/lib`).
      
      - If a customs property named LIBPATH does not already exists, then add by clicking **New** and setting the value to the directory where the passticket .so file is stored (for example, `/usr/lib`).

**Note:** Keep in mind the following considerations.

1. WebSphere Application Server has the notion of **shared libraries** (environment → shared libraries) that can be used to define common libraries used by one or more applications. In our case we could use this functionality since the passticket function is based on JNI code. When we associated this shared library with two WebSphere Application Server applications (the MDB and the JMSSession EJB) we got the exception `java.lang.UnsatisfiedLinkError: Native Library /usr/lib/libIRRRacf.so already loaded in another classloader when calling the second application. If you still want to use shared libraries in WebSphere Application Server, one solution is to isolate the passticket functionality into a separate EJB deployed with a separate application.

2. Do not include the IRRRacf.jar in your application .ear file because this .jar is a piece of infrastructure (just like, for example, JDBC drivers). In RAD this file should only be referenced as an **external jar file** on the build path.
**RACF definitions for login using RACF passticket**

In our sample we reused the RACF application “CBS390” that was installed with WebSphere Application Server z/OS when WebSphere Application Server z/OS was running under the user ID “ASSR1.” Our RACF definitions are shown in Example 25.

**Example 25  RACF passticket definitions reusing RACF application CBS390**

```
# Modify existing CBS390
RALTER PTKTDATA CBS390 SSIGNON(KEYMASKED(0123456789abcdef))

# Security profile IRRPTAUTH.CBS390.* to protect the CBS390 from
# unauthorized use.
# This determines who may create PassTickets for CBS390
RDEFINE PTKTDATA IRRPTAUTH.CBS390.* UACC(NONE)

# Permission to create the PassTicket for CBS390
PERMIT IRRPTAUTH.CBS390.* CLASS(PTKTDATA) ID(ASSR1) ACCESS(UPDATE)

# Complete PTKTDATA setup
SETR RACLIST(PTKTDATA) REFRESH
```

If we wanted to use another RACF application (for example, JMSAPPL), the RACF definitions should be similar to those shown in Example 26.

**Example 26  RACF passticket definitions using the new RACF application JMSAPPL**

```
# Activate the PTKDATA class (if not already done)
SETR CLASSACT(PTKTDATA)
SETR RACLIST(PTKTDATA)
SETR RPOTS generic(PTKTDATA)

# Define PTKDATA profiles.
RDEF PTKTDATA JMSAPPL SSIGNON(KEYMASKED(0123456789abcdef))

# Security profile IRRPTAUTH.JMSAPPL.* to protect the JMSAPPL from
# unauthorized use.
# This determines who may create PassTickets for JMSAPPL
RDEF PTKTDATA IRRPTAUTH.JMSAPPL.* UACC(NONE)

# Permission to create the PassTicket for JMSAPPL
PERMIT IRRPTAUTH.JMSAPPL.* CLASS(PTKTDATA) ID([userid WAS is running under]) ACCESS(UPDATE)

# Complete PTKDATA setup
SETR RACLIST(PTKTDATA) refresh
SETR GENERIC(PTKTDATA) refresh
```
Log in using a custom login module

In addition to the JAAS login modules shipped with WebSphere you can develop your own JAAS custom login modules.

The information center and WebSphere manual “Securing applications and their environment” cover JAAS and JAAS login modules in detail.

BusinessLogic session EJB

The mission of the BusinessLogic session EJB is to just print out the J2EE identity it is running under.

BusinessLogic session EJB code

Example 27 shows the Java code of the doWork() method of the BusinessLogic session EJB.

```
Example 27  Code of BusinessLogic session EJB method doWork( )

public String doWork(String data) {
    System.out.println("BL: Running under: " + getSessionContext().getCallerPrincipal());
    return "OK";
}
```

Definitions for the JMS test application

In the following sections we discuss administration considerations regarding the objects we have been talking about so far.
Overview of the objects in WebSphere MQ, DB2, WebSphere Application Server, and SIB

For our tests, we roughly had the following artifacts defined:

- WebSphere MQ: A local queue
- WebSphere Application Server:
  - An SIB bus and Message Engine
  - A set of JMS Queue Connection Factories for connecting to WebSphere MQ and SIB
  - A set of JMS destinations (queues) that pointed to queues in WebSphere MQ and SIB

Setting up WebSphere MQ

Using the WebSphere MQ GUI Administrator (or comparable tool), create a local queue called JMSTESTQ by specifying the MQSC command as shown in Example 28.

Example 28 Creating a WebSphere MQ local queue

```
DEF QL(JMSTESTQ)
```

Setting up WebSphere

This section describes how to use the WebSphere Application Server admin console to define the necessary WebSphere MQ and SIB artifacts.

Define the WebSphere MQ artifacts in WebSphere Application Server

Define the following artifacts associated to WebSphere MQ in the WebSphere Application Server admin console:

1. Define a J2C Alias for the Connection Factory:
   b. Make the following entries:
      - Alias: JMSUSER
      - User ID: <Your user ID>
      Enter your user ID in upper case, otherwise WebSphere MQ might not find it in RACF.
• Password: <Your password>
   Enter your password in upper case, otherwise WebSphere MQ might be able to verify it correctly with RACF.

c. Click OK.

2. Define the WebSphere MQ Queue Connection Factory to JNDI:
   a. Select Resources → JMS Provider → WebSphere MQ → WebSphere MQ Queue Connection Factories → New.
   b. Make the following entries:
      • Name: QM1
      • JNDI name: jms/QM1
      • Queuemanager: <your queuemanager>
      • Container-managed authentication alias: JMSUSER
      • Transport Type: BINDINGS
   c. Click OK.

3. Define the WebSphere MQ Queue Destination to JNDI:
   a. Select Resources → JMS Provider → WebSphere MQ → WebSphere MQ Queue Destinations.
   b. Make the following entries:
      • Name: Q1
      • JNDI name: jms/Q1
      • Base Queue name: JMSTESTQ
   c. Click OK.

4. Define the WebSphere MQ JMS listener:
   a. Select Application servers → <your server> → Messaging → Message Listener Service.
   b. Make the following entries:
      • Listener Port: New
      • Name: JMSListener
      • Initial State: Started
      • Connection factory JNDI name: jms/QM1
      • Connection factory JNDI name: jms/Q1
   – Click OK.
**Define the SIB artifacts in WebSphere Application Server**

Define the following artifacts associated to the Service Integration Bus in the WebSphere Application Server admin console:

1. Define the SIB bus:
   a. Select **Service Integration → Buses → New**.
   b. Make the following entries and selections:
      - Name: JMSBus
      - Mediation authentication Alias: JMSUSER
   c. Click **OK**.

2. Define your server to be part of the bus:
   a. Select **Service Integration → Buses → JMSBus → Bus Members**.
   b. Click **Add**.
   c. Select your server and keep the default checked under Datastore.
   d. Click **Next**.
   e. Click **Finish**.

3. Define a SIB queue on the bus:
   a. Select **Service Integration → Buses → JMSBus → Destinations → New → Queue**.
   b. Specify Identifier: Q2
      - Click **Next**.
      - Assign the queue to a bus member by select your bus member.
      - Click **Next**.
      - Click **Finish**.

4. Add the JMS Queue Connection Factory for SIB to JNDI:
   a. Select **Resources → Default Messaging → JMS Queue Connection Factory → New**.
   b. Make the following entries:
      - Name: QM2
      - JNDI name: jms/QM2
      - Bus name: JMSBus
      - Component managed auth. alias: JMSUSER
      - Click **OK**.
5. Add the SIB JMS Queue Destination to JNDI:
   a. Select **Resources** → **Default Messaging** → **JMS Queue** → **New**.
   b. Make the following entries:
      - Name: Q2
      - JNDI name: jms/Q2
      - Bus name: JMSBus (Select this before selecting the queue!)
      - Queue name: Q2
   c. Click **OK**.

6. Add the JMS Activation Specification (the “JMS listener” for SIB):
   - Select **Resources** → **Default Messaging** → **Activation Specification** → **New**.
   - Make the following entries:
      - Name: ACT1
      - JNDI name: jms/ACT1
      - Destination Type: Queue
      - Destination JNDI name: jms/Q2
      - Bus name: selectJMSBus
      - Authentication Alias: JMSUSER
        (This is needed when the bus is secure so that Activation Specification
        can access the bus!)
   - Click **OK**.

7. Save changes to the WebSphere master configuration and sync WebSphere
   Application Server nodes.

**Deploying the test application**

In the following sections we discuss how to deploy the test application used in
this paper.

**Deploying the JMSSession EJB**

Deploy the JMSSession EJB using the WebSphere Application Server admin
console, as follows:

1. Select **Applications** → **Install New Application**.
2. Select Local file system and click Browse to select the .ear file holding the JMSSession EJB (WASSecJMSSession.ear). This is shown in Figure 28.

![Enterprise Applications]

Figure 28 Deploying JMSSession EJB: Step 1

3. Click Next. The window shown in Figure 29 is returned.

![Enterprise Applications]

Figure 29 Deploying JMSSession EJB: Step 2

4. Click Next. The window shown in Figure 30 on page 87 is returned.
5. Click **Next**.

   In the resulting panel define on which servers and clusters the EJB should run.

6. Using the check box, select the module **JMSSession** and then, in the Clusters and Servers section, select the cluster or server you want this EJB to deploy to.

7. Click **Apply**. The window shown in Figure 31 on page 88 is displayed.
8. Click Next. The window shown in Figure 32 is displayed.

9. Click Next.

On the resulting panel define the mapping of the resource references (for example, a reference to a JMS Queue Connection Factory) to a corresponding artifact defined in WebSphere Application Server (for example,
a WebSphere MQ JMS Queue Connection Factory). Since in the EJB we already used WebSphere Bindings to propose which WebSphere artifact to use for a resource reference, we do not have to do anything here.

Figure 33  Deploying JMSQueue EJB: Step 6

10. Click Next.

The window shown in Figure 34 on page 90 may optionally appear if you deployed your application to a cluster, but have defined your JMS Connection Factories only at the node level.
11. Click **Continue**. The window shown in Figure 35 on page 91 is displayed.
12. Click **Next**. The window shown in Figure 36 is displayed.

**Figure 35** Deploying JMSSession EJB: Step 8

**Figure 36** Deploying JMSSession EJB: Step 9
13. Click **Finish**.

14. Select the link **Save to WebSphere Master Configuration**; on the next panel select **Synchronize changes with nodes** and click **Save** to distribute your changes within the WebSphere Application Server cell.

**Deploying the JMSSession EJB client**
Transfer the J2EE application holding the JMSSession EJB client (WASSecJMSSession.ear) in binary mode to a USS directory (for example, /u/wsadmin).

**Deploying the MDB and BusinessLogic session EJB**
The MDB and BusinessLogic session EJB are in one .ear file, so they are deployed together. Perform the following steps using the WebSphere Application Server admin console:

1. Select **Enterprise Applications → Install**.

2. Select **Local file system** and click **Browse** to select the .ear file holding the MDB and the BusinessLogic session EJB (WASSecMDB.ear), as shown in Figure 37.

![Figure 37 Deploying MDB / BusinessLogic session EJB: Step 1](image)

3. Click **Next**. The window shown in Figure 38 is displayed.
4. Click **Next**. The window shown in Figure 39 is displayed.
5. Click **Next**.
   
   In the resulting panel define on which servers and clusters the EJB should run.

6. Using the check boxes, select the modules **MDB** and **BusinessLogic**; then select in the Clusters and Servers section the cluster or server you want this EJB to deploy to.

7. Click **Apply**. The window shown in Figure 40 is displayed.

![Enterprise Applications](image)

**Figure 40** Deploying MDB/BusinessLogic session EJB: Step 4

8. Click **Next**.

   On the resulting panel you define which JMS listener port/Activation Specification in WebSphere Application Server should be used for the MDB. Since in the MDB we already used WebSphere Bindings to propose which WebSphere artifact (Listener Port of Activation Specification) to use for a resource reference, we do not have to do anything here.
Figure 41   Deploy MDB/BusinessLogic-Session-EJB: Step 5

9. Click **Next**.

The window shown in Figure 42 on page 96 may optionally appear if you defined the Activation Specification on a cluster level but are deploying your application at a node level.
10. Click **Continue**. The window shown in Figure 43 is displayed.

**Figure 42**  Deploying MDB/BusinessLogic session EJB: Step 6

**Figure 43**  Deploying MDB/BusinessLogic session EJB: Step 7
11. Click **Next**.

On the following panel the security roles defined for the MDB (for RunAs identity) are mapped to existing security roles.

Select the role **mdbuser** and select Everyone? and All authenticated?, as shown in Figure 44.

---

**Figure 44** Deploying MDB/BusinessLogic session EJB: Step 8
12. Click **Next**.

On the resulting panel define which RunAs identity is to be used for the MDB. Select **mdbuser** and enter a valid RACF user ID and password into the fields username and password. Click **Apply**. See Figure 45.

**Figure 45**  Deploying MDB / BusinessLogic session EJB: Step 9
13. Click **Next**. The window shown in Figure 46 is displayed.

![Figure 46](image)

**Figure 46**  Deploying MDB/BusinessLogic session EJB: Step 10
14. Click **Next**. The window shown in Figure 47 is displayed.

**Figure 47** Deploying MDB/BusinessLogic session EJB: Step 11

15. Click **Finish**.

16. Select the link **Save to WebSphere Master Configuration**. On the next panel select **Synchronize changes with nodes** and click **Save** to distribute your changes within the WebSphere Application Server cell.

**Deploying the SIB mediation handler**

Perform the following steps using the WebSphere Application Server admin console to define the SIB mediation handler:

1. Select **Applications → Install New Application**.
2. Select **Local file system** and click **Browse** to select the .ear file holding the SIB mediation handler (WASSecMediation.ear), as shown in Figure 48 on page 101.
3. Click **Next**. The window shown in Figure 49 is displayed.

4. Click **Next**. The window shown in Figure 50 on page 102 is displayed.
5. Click **Next**.

   In the resulting panel you define on which servers and clusters the EJB should run.

6. Using the check box select the module **Mediation**, then select in the Clusters and Servers section the cluster or server you want this EJB to deploy to.

7. Click **Apply**. The window shown in Figure 51 on page 103 is displayed.

---

**Figure 50  Deploying SIB mediation handler: Step 3**
8. Click **Next**. The window shown in Figure 52 is displayed.

9. Click **Next**. The window shown in Figure 53 on page 104 is displayed.
10. Click **Next**. The window shown in Figure 54 is displayed.

11. Click **Finish**.
12. Select the link Save to WebSphere Master Configuration; on the next panel select Synchronize changes with nodes and click Save to distribute your changes within the WebSphere Application Server cell.

Running the JMSSession EJB client

To run the test application, run the JMSSession EJB client as follows:

1. In UNIX® System Services, go to the <WAS_HOME>/bin directory.
2. Enter the following command:
   
   `launchClient.sh /<path>/WASSecJMSSession.ear -CCBootstrapPort=38531`

   In this example:
   
   `/<path>/WASSecJMSSession.ear` (in our case) is the .ear file holding the JMSSession EJB client code.
   
   38531 is the RMI-Port of the server the JMSSession EJB was deployed to. (See Servers → Application Servers → <your application server> → Ports → ORB_LISTENER_ADDRESS.)

Tools used during the tests

We used a number of tools during the development and deployment of the test application. They are briefly discussed in the following sections.

Rational Application Developer (RAD)

RAD is the recommended tool from IBM to build J2EE applications. See http://www-306.ibm.com/software/awdtools/developer/application/index.html for more information.

Service Bus Explorer

The SIB Explorer provides a graphical overview of the SIB. You can download this tool from IBM alphaworks at:

http://www.alphaworks.ibm.com/tech/sibexplorer

This tool is based on the SWT graphics library. After some experimenting we found that the SWT libraries version swt-M20060629-1905-win32-win32-x86 works together with SIB Explorer.

Figure 55 on page 106 shows an example of the SIB Explorer.
WebSphere MQ GUI Administrator

You can use the WebSphere MQ GUI Administrator to administer WebSphere MQ on z/OS. You can download this tool (SupportPac™ MO71) from: http://www-1.ibm.com/support/docview.wss?rs=171&uid=swg27007197#1

Since you act as an MQ client to the WebSphere MQ queue manager, you need to create a new Location in MO71 where you click the Client check box. This is shown in Figure 56 on page 107.
Figure 56  MO71 SupportPac: Connect to a remote queue manager
Then click **Configured** and enter data such as TCP/IP address and port of the WebSphere MQ queue manager as shown in Figure 57. The channel name you define here must exist in WebSphere MQ.

![MQ4B/Client Channel Definition](image)

*Figure 57  MO71 SupportPac channel definition (see port specification)*

Once you have the connection to WebSphere MQ on z/OS working you can, for example, browse the queues (see Figure 58 on page 109) or enter administrative MQSC commands to create queues and so on.
The WebSphere MQ utilities amqsbcg and amqsbcgc, which are both delivered with non-z/OS WebSphere MQ in the <WMQ>/Tools/c/Samples/Bin directory, can be used to browse WebSphere MQ queues and to display the messages stored on the queues. See Example 3 on page 11 and Example 4 on page 12 for a sample output.

Downloadable material

The examples used throughout this paper are based on sample applications that can be downloaded from the ITSO additional material Web site at:

ftp://www.redbooks.ibm.com/redbooks/REDP4203

The following files can be downloaded:

WASsecJMSSession.ear .ear file with the JMS session EJB sample code
WASSecMDB.ear .ear file with the MDB sample code
The team that wrote this Redpaper

This Redpaper was produced by a team of specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.

Alex Louwe Kooijmans is a project leader at the International Technical Support Organization, Poughkeepsie Center. He leads residencies and instructs workshops in the area of Java and WebSphere on z/OS®. Alex has worked for IBM since 1986 and this is his second assignment to the ITSO. Before joining the ITSO, he worked in various other roles, such as IT Specialist supporting customers in Europe getting started with WebSphere on various platforms, and Client IT Architect in the financial services sector in The Netherlands. His areas of expertise include Java and WebSphere on z/OS, and integrating WebSphere with DB2, MQ, CICS, and IMS.

Andreas Landenberger is a Consulting IT Specialist working with IBM Software Group Services (ISSW), IBM Germany. He holds a degree in Engineering from the Technical University Munich and has more than 20 years of experience in designing and developing applications on various platforms like Windows®, AIX®, /390 (CICS, IMS, and so forth). He has worked with object-oriented technology since 1992, first with C++ and now with Java, and he contributed to the writing the IBM Redbooks Enterprise JavaBeans for z/OS and OS/390, SG24-6284 and z/OS WebSphere and J2EE Security Handbook, SG24-6846.

Thanks to the following people for their contributions to this project:

Richard M Conway
International Technical Support Organization, Poughkeepsie Center

Tom Hackett, Michael Onghena
IBM Poughkeepsie

Andreas Konecny
IBM ISSW Germany
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 Drive Armonk, NY 10504-1785 U.S.A.

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

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

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. 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.

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.

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

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

© Copyright International Business Machines Corporation 2006. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
This document created or updated on November 21, 2006.

Send us your comments in one of the following ways:

- Use the online Contact us review redbook form found at:
  ibm.com/redbooks
- Send your comments in an email to:
  redbook@us.ibm.com
- Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. HYTD  Mail Station P099, 2455 South Road
  Poughkeepsie, NY 12601-5400 U.S.A.

Trademark

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

- Redbooks (logo) ™
- ibm.com®
- z/OS®
- AIX®
- Cloudscape™
- CICS®
- DB2®
- IBM®
- IMS™
- Rational®
- Redbooks™
- RACF®
- SupportPac™
- WebSphere®

The following terms are trademarks of other companies:

- Enterprise JavaBeans, EJB, Java, JavaBeans, JDBC, JVM, J2EE, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

- Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

- UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.