WebSphere MQ Everyplace V2.0.2

Contents

Working with WebSphere Message Broker

Scenarios

MQe applications on different MQe queue managers communicate with each through the exchange of messages. In the simplest case, the source and target queue managers are either directly connected to each other through MQe channels, or indirectly connected with the messages between them passing through intermediate MQe queue managers. In all of these cases, the standard MQe message object class com.ibm.mqe.MQeMsgObject is used to create these messages or some appropriate subclass (as in the case of administrative messages). The situation becomes more complicated when the intermediate queue managers are not MQe queue managers but are WebSphere® MQ managers. In this case, the messages can be regarded as tunnelling across a MQ network; with the passage across the MQ network having no affect on either the sending or receiving application. Tunnelling requires the following configuration:

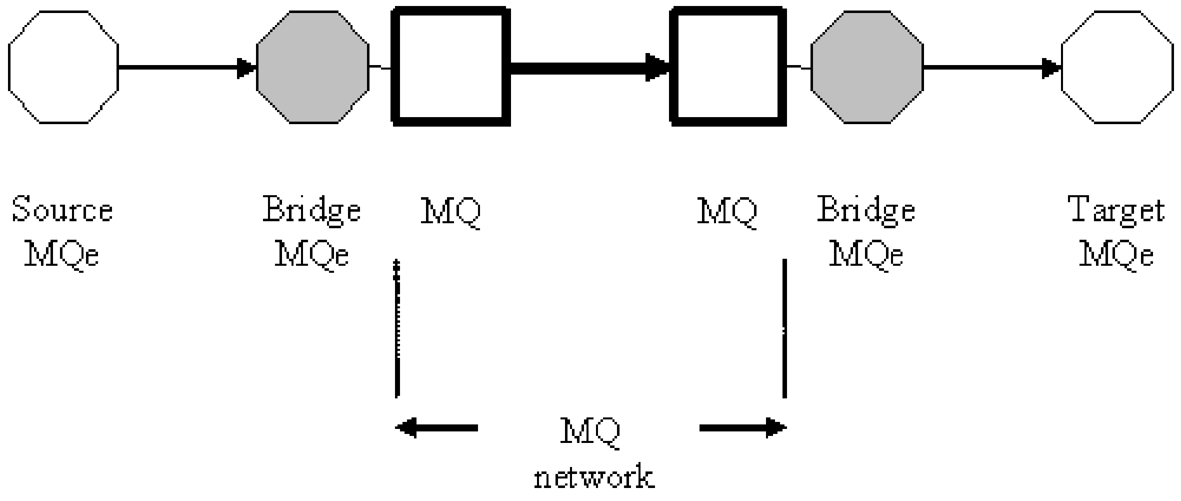


Figure 1. MQe message tunnelling

The MQe queue manager that passes the messages to an MQ queue manager is configured as a bridge queue manager. Likewise, the MQe queue manager that receives messages from an MQ queue manager is also configured as a bridge queue manager. Bridge queue managers are sometimes referred to as gateway queue managers. The details of bridge configuration as described elsewhere; it is sufficient here to understand that a single configured bridge can handle messages travelling in both directions (for example, to and from MQ) and that the important determinants in its behavior are: (a) the transformer class property value (set in the bridge configuration), and (b) for messages passing from MQe to MQ, the class of the MQe message. In many cases the requirement is not to transport messages across a MQ network, but to use messages from a MQe source application to drive a target MQ application, typically returning the results in a reply message. The configuration involved is of the form shown below, though the target application can run on any MQ queue manager, including the one immediately connected to the MQe bridge:

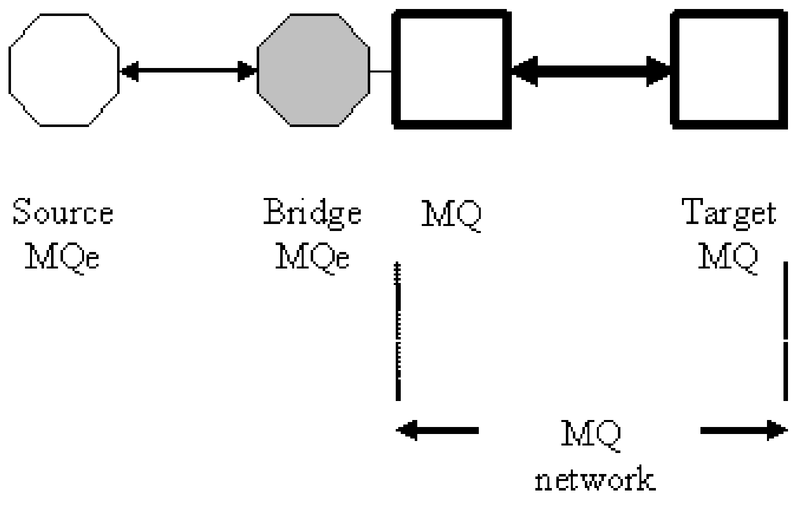


Figure 2. Driving MQ applications

As before, the important determinants in the behavior of the bridge are:

- The transformer class property value (set in the bridge configuration)
- For messages passing from MQe to MQ, the class of the MQe message

The class of message used by the source must allow complete control over the generated MQ message being received by the MQ application. Typically the com.ibm.mqe.mqemqmessage.MQeMQMsgObject class is used for this purpose; however, if JMS is being used as the messaging model at both the source and the target, then the com.ibm.mqe.MQeMsgObject message class is used, although with very specific content present.

An additional requirement is to support the needs of publish and subscribe applications, where an IBM® broker is the destination MQ application and manages both the subscriptions and the publications. The MQe source is liable to be just one of many sources inputting to the broker. There are two configurations possible to implement a publish/subscribe network, however only one is now recommended. This is:

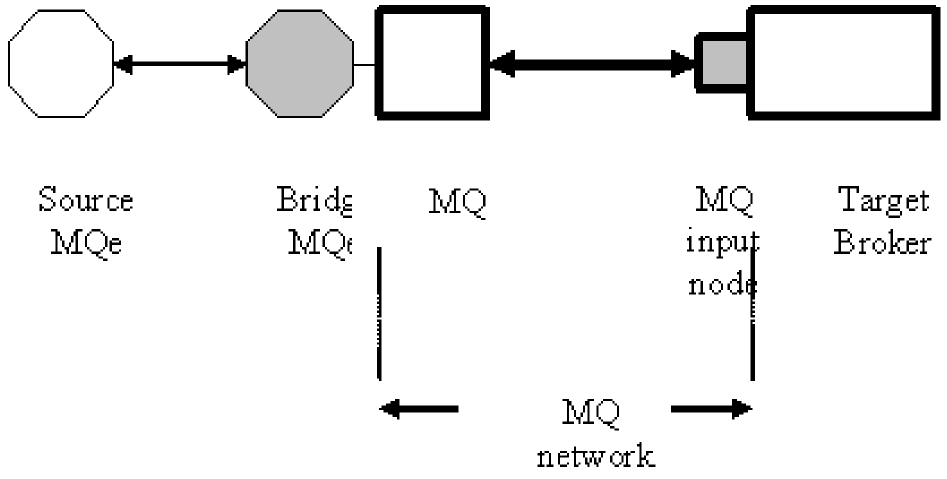


Figure 3. Publish/subscribe via the MQe bridge

Yet again, the important determinants in the behavior of the bridge are: (a) the transformer class property value (set in the bridge configuration), and (b) for messages passing from MQe to MQ, the class of the MQe message. Here, the choice of message used by the source, must allow a publish/subscribe MQ message to be output by the bridge and subsequently received by the broker. The com.ibm.mqe.mqemqmessage.MQePubSubMsgObject class is used for this purpose. For responses back from the broker, the bridge transformer class detects from the MQ message content that it is a publish/subscribe response and therefore returns a com.ibm.mqe.mqemqmessage.MQePubSubMsgObject class message back to the source.

In previous versions of MQe the following configuration was supported:

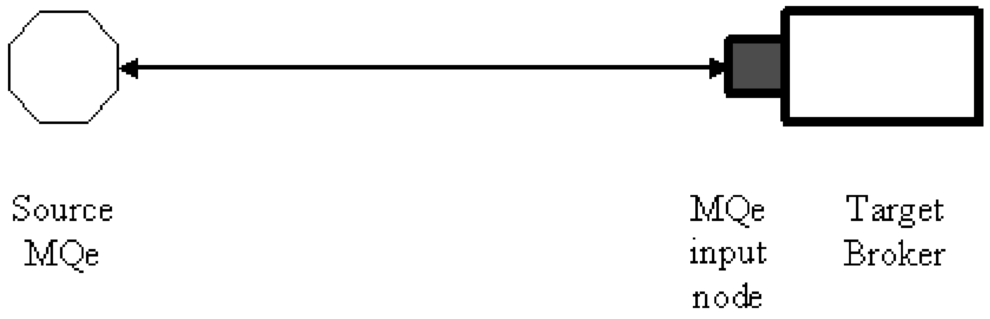


Figure 4. Publish/subscribe via the MQe broker input node

The source sent a class com.ibm.broker.mqimqe.wrapper.MQeMbMsgObject or a com.ibm.mqe.MQeMsgObject message to the MQe input node on the broker. This configuration is no longer recommended; migration scenarios from earlier implementations are discussed later in this section.

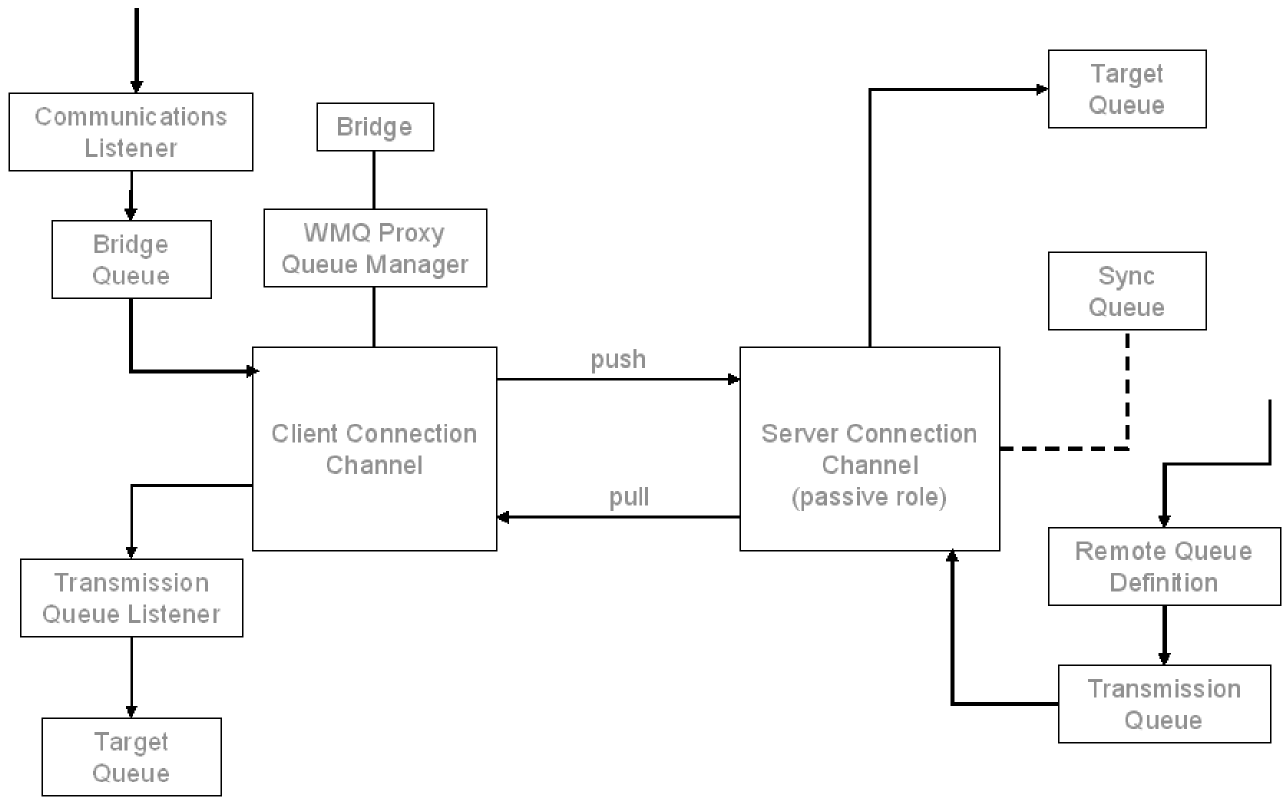
Bridge configuration

The following summary of the MQe bridge is provided to aid understanding of the configurations and tasks described in this document.

The MQe bridge (or gateway) is a specially configured MQe queue manager that acts as a link between an MQe network and an MQ network. For these purposes, a WebSphere Message Broker queue manager is regarded as just another MQ queue manager in the MQ network. A bridge comprises the following elements linked together in a hierarchical, tree-like relationship:

- Bridge
- MQ proxy
- Client connection
- Listener

The following diagram shows both MQe and MQ objects required for a bridge and the flow of the messages.



At each level in the hierarchy, multiple elements can exist (e.g. multiple MQ proxies per bridge, multiple client connections per MQ proxy, etc). The bridge object reflects the overall hierarchy and establishes some default behavior. The MQ proxy identifies the target MQ queue manager; the client connection describes the details of the mechanisms through which the MQe and MQ communicate. Finally, the listener provides the additional information needed to move messages from MQ to MQe. The bridge can thus be regarded as providing the queue manager-level addressability between the two networks.

One element of the bridge configuration involves the specification of the transformer class to be used; this transformer will be invoked whenever an MQe message is to be sent to MQ, or whenever a MQ message is to be sent to MQe.

Once a bridge has been configured it is necessary, on that same MQe queue manager, to add details of the MQ queues that are to be accessible from MQe. This involves the definition of bridge queues, each definition identifies one target queue on MQ. Bridge queues thus provide the queue-level addressability between the two networks.

MQe bridge transformer classes

The bridge transformer class controls the conversion of messages in the bridge, i.e. messages being received from an MQe queue manager and sent to an MQ queue manager, and the movement of messages in the reverse direction. A number of transformers are provided, however some of these are historic and are now deprecated. The recommended transformers are given below, together with their class relationship:

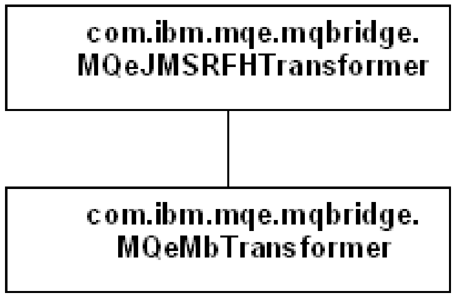


Figure 5. Bridge transformer class hierarchy

Recommended transformers by task are shown in bold in the table below; other transformers that can be used (and produce identical results) are shown in normal type:

Table 1. MQe bridge transformers by task

Task	**Transformer**
MQe message tunnelling	**com.ibm.mqe.mqbridge.MQeJMSRFHTransformer** com.ibm.mqe.mqbridge.MQeBaseTransformer com.ibm.mqe.mqbridge.MQeGeneralRFHTransformer
JMS usage with MQe/MQ	**com.ibm.mqe.mqbridge.MQeJMSRFHTransformer**
Driving MQ applications	**com.ibm.mqe.mqbridge.MQeJMSRFHTransformer** com.ibm.mqe.mqbridge.MQeBaseTransformer com.ibm.mqe.mqbridge.MQeGeneralRFHTransformer
Publish/subscribe (via MQePubSubMsgObject)	**com.ibm.mqe.mqbridge.MQeJMSRFHTransformer**
Publish/subscribe (via MQeMbMsgObject)	com.ibm.mqe.mqbridge.MQeMbTransformer

The features of the recommended transformer classes are:

Table 2. Bridge transformer class features

Transformer class	Features (MQe to MQ conversion)
com.ibm.mqe.mqbridge. MQeJMSRFHTransformer	Incoming MQe messages of class MQePubSubMsgObject are converted as pub/sub messages if they contain pub/sub data, in order to drive the broker; otherwise they are treated as MQeMQMsgObject messages. Messages of class MQeMQMsgObject are processed to set the MQMD and payload of the output MQ message, such that they can drive MQ applications. All other messages are inspected to see if they contain JMS content and, if so, they are converted accordingly; otherwise they are tunnelled through the MQ network.
com.ibm.mqe.mqbridge. MQeMbTransformer	Adds support for the MQeMbMsgObject. Any incoming message of MQeMbMsgObject class is converted to the equivalent MQePubSubMsgObject and then passed to the super-class transformer. All other input message classes are not affected.
Transformer class	**Features (MQ to MQe conversion)**
com.ibm.mqe.mqbridge. MQeJMSRFHTransformer	Incoming MQ messages containing a tunnelled MQe messages are converted to a message of class MQeMsgObject. MQ messages containing JMS data are converted into MQeMsgObject messages with JMS data. MQ messages containing replies from a pub/sub broker are converted into MQePubSubMsgObject messages. All other messages are converted into MQeMQMsgObject messages.
com.ibm.mqe.mqbridge. MQeMbTransformer	Adds support for the MQeMbMsgObject. Any MQe message output of class MQePubSubMsgObject, generated by the super-class transformer, is converted to the equivalent MQeMbMsgObject message. All output message of other classes are not affected

MQe message object classes

Messages from the MQe source to the bridge (or to the MQe input node on the broker) are processed according to the message class. The class hierarchy of the message classes provided by MQe is shown by the shaded boxes in the diagram below. Note that all message classes inherit from the com.ibm.mqe.MQeFields class, which provides a generic ability to get/put data values into the message objects.

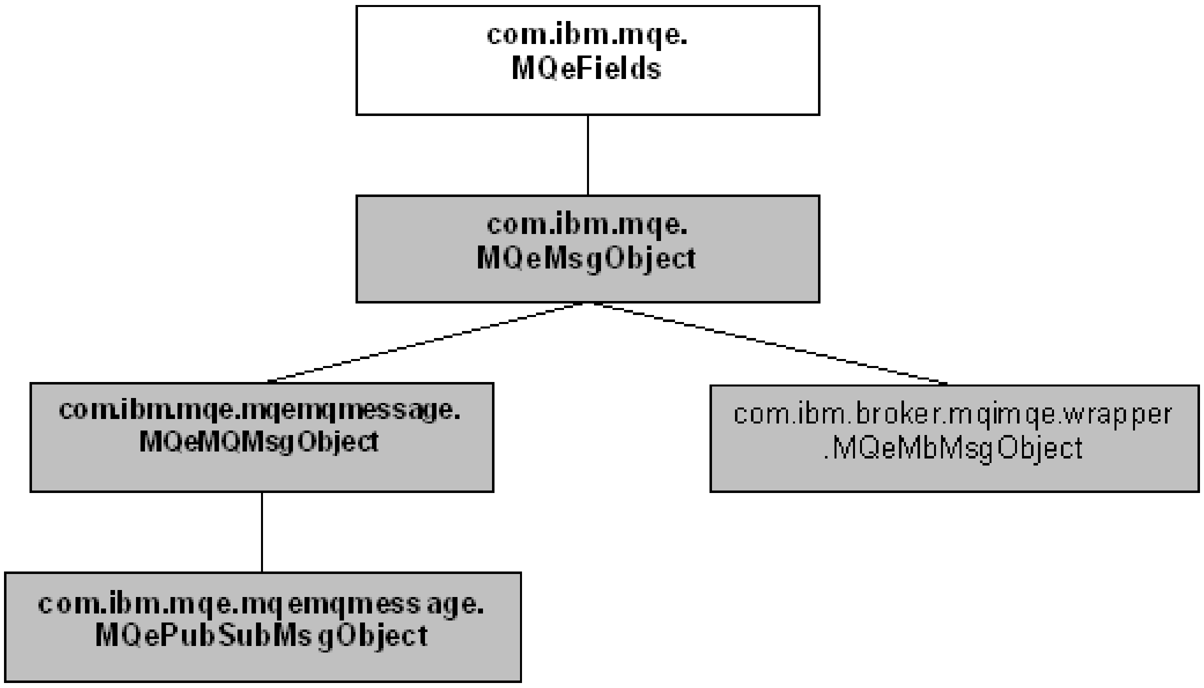


Figure 6. Message class hierarchy

The features of the various classes are:

Table 3. MQe message class features

Message object class	Features
com.ibm.mqe.MQeMsgObject	Base MQe message class. MQeFields methods used to get/put user data.
com.ibm.mqe.mqemqmessage. MQeMQMsgObject	Adds the ability to set the contents of the MQMD and the payload of the MQ message that will be generated by the bridge.
com.ibm.mqe.mqemqmessage. MQePubSubMsgObject	Adds specific pub/sub methods to drive the behavior of the broker.
com.ibm.broker.mqimqe.wrapper. MQeMbMsgObject	Adds the ability to set the equivalent data to that held in the MQMD of an MQ message. MQeFields methods using defined constants are used to implement pub/sub functionality.

The recommended message classes appropriate to various tasks are shown in the table below:

Table 4. MQe message classes by task

Task	Message object class
MQe message tunnelling	**com.ibm.mqe.MQeMsgObject** (or any subclass appropriate to the application)
JMS usage with MQe/MQ	**com.ibm.mqe.MQeMsgObject** (with specifically formatted content)

Table 4. MQe message classes by task (continued)

Task	Message object class
Driving MQ applications	**com.ibm.mqe.mqemqmessage.MQeMQMsgObject** (or any subclass appropriate to the application) **Note:** The default character set for MQeMQMsgObject is 1200. When used with the Websphere Message Broker, this should be altered to 1208. **com.ibm.mqe.mqemqmessage.MQePubSubMsgObject** (or any subclass appropriate to the application)
Publish/subscribe (via MQePubSubMsgObject)	**com.ibm.mqe.mqemqmessage.MQePubSubMsgObject** (or any subclass appropriate to the application)
Publish/subscribe (via MQeMbMsgObject)	**com.ibm.broker.mqimqe.wrapper.MQeMbMsgObject**

Writing MQe applications to drive MQ applications

There are two aspects to writing an MQe application to drive an MQ application. These are:

Configuration

1. Configure a bridge to provide access to the target MQ queue manager from the MQe network, specifying the transformer class com.ibm.mqe.mqbridge.MQeJMSRFHTransformer. Also on the bridge queue manager, configure two bridge queues:
   - One to act as a proxy for the broker’s control queue SYSTEM.BROKER.CONTROL.QUEUE.
   - One to act as a proxy for the input queue of the appropriate message flow.
2. On the source MQe queue manager, configure connections to the MQe bridge queue manager and the target MQ application queue manager (this latter connection being via the bridge queue manager).
3. On the source MQe queue manager, configure a remote proxy queue to the target MQ application queue.
   - One to act as a proxy for the broker’s control queue SYSTEM.BROKER.CONTROL.QUEUE.
   - One to act as a proxy for the input queue of the appropriate message flow.

Application development

1. Construct an MQe message that will be transformed by the bridge into an MQ message suitable for the broker. This message will have a standard MQMD header and an RFH2 header, as well as a suitable payload.
2. Send the subscription and un-subscription messages to the broker’s control queue. Send publication messages to the input queue of the appropriate message flow.

The message should be of the com.ibm.mqe.mqemqmessage.MQePubSubMsgObject class. Should it be necessary to use a bespoke MQ MQMD, then values e message object (such as setCorrelationId, setMessageId, setExpiry, setFeedback, setReport, setUserId). The publish/subscribe operations are invoked through the methods publish, subscribe and unsubscribe.

Responses from the broker will be received as messages of the same class. The overall success of an operation is given by the getCompletionCode method; more detailed information is provided by getResponses. The broker never returns the publication data, but information on the nature of the original request is available through getActionType, getDestQueueMgr, getDestQueueName, getRetention and getTopics.

For messages that have been constructed, the following methods can be used to return information about the request: getMessage and getMessageData.

More information on the configuration details appropriate to a bridge is provided in the MQe publications, including the handling of replies back from the target MQ application.

Writing MQe pub/sub applications

There are two aspects to writing an MQe application to drive an MQ broker application. These are:

Configuration

1. Configure a bridge to provide access to the target MQ queue manager from the MQe network specifying the transformer class com.ibm.mqe.mqbridge.MQeJMSRFHTransformer. Also on the bridge queue manager, configure two bridge queues:
   - One to act as a proxy for the broker's control queue `SYSTEM.BROKER.CONTROL.QUEUE`.
   - One to act as a proxy for the input queue of the appropriate message flow.
2. On the source MQe queue manager, configure connections to the MQe bridge queue manager and the target broker queue manager (this latter connection being via the bridge queue manager).
3. On the source MQe queue manager, configure two remote proxy queues:
   - One to act as a proxy for the broker's control queue `SYSTEM.BROKER.CONTROL.QUEUE`.
   - One to act as a proxy for the input queue of the appropriate message flow.

Application development

1. Construct an MQe message that will be transformed by the bridge into an MQ message suitable for the broker. This message will have a standard MQMD header and an RFH2 header, as well as a suitable payload.
2. Send the subscription and un-subscription messages to the broker's control queue. Send publication messages to the input queue of the appropriate message flow.

The message should be of the com.ibm.mqe.mqemqmessage.MQePubSubMsgObject class. Should it be necessary to use a bespoke WMQ MQMD, then values e message object, (for example, setCorrelationId, setMessageId, setExpiry, setFeedback, setReport, setUserId, and so on). The publish/subscribe operations are invoked through the methods publish, subscribe and unsubscribe.

Responses from the broker will be received as messages of the same class. The overall success of an operation is given by the getCompletionCode method; more detailed information is provided by getResponses. The broker never returns the publication data, but information on the nature of the original request is available through getActionType, getDestQueueMgr, getDestQueueName, getRetention and getTopics.

For messages that have been constructed, the following methods can be used to return information about the request: getMessage and getMessageData.

More information on the configuration details appropriate to a bridge is provided in the MQe publications, including the handling of replies back from the target MQ application.

Examples

All examples below are based on those in the *MQI Programming Guide* in Appendix C, ″MQSeries® Everyplace® Nodes″. An example of a publish application:

```
try
{
    System.out.println("local QM name: " + myQMgr.getName());

     MQePubSubMsgObject mqeMsg = new MQePubSubMsgObject();
     mqeMsg.publish("Weather", true, "Hello");

     System.out.println("..Put message to QM/queue: " +
            brokerQueueManager + "/" + flowInputQueue);
     myQM.putMessage(brokerQueueManager, flowInputQueue, mqeMsg, null, 0);
     System.out.println("Finished");
}
catch (Exception e)
{
    e.printStackTrace();
     System.out.println("Failed! + e);
}
```

An example of a subscribe application:

```
try
{
    System.out.println("local QM name: " + myQMgr.getName());

     MQePubSubMsgObject mqeMsg = new MQePubSubMsgObject();
     String[] topics = new String[]{"Topic1", "Topic2", "Topic3"};
     mqeMsg.subscribe(topics, "ServerQM1", "Inbox");

     System.out.println("..Put message to QM/queue: " +
           brokerQueueManager + "/" + controlQueue);
     myQM.putMessage(brokerQueueManager, "SYSTEM.BROKER.ADMIN.QUEUE", mqeMsg, null, 0);
     System.out.println("Finished");
}
catch (Exception e)
{
    e.printStackTrace();
     System.out.println("Failed! + e);
}
```

An example of a un-subscribe application:

```
try
{
    System.out.println("local QM name: " + myQMgr.getName());

     MQePubSubMsgObject mqeMsg = new MQePubSubMsgObject();
     String[] topics = new String[]{"Topic1", "Topic2", "Topic3"};
     mqeMsg.unsubscribe(topics, "ServerQM1", "Inbox");

     System.out.println("..Put message to QM/queue: " +
           brokerQueueManager + "/" + controlQueue);
     myQM.putMessage(brokerQueueManager, "SYSTEM.BROKER.ADMIN.QUEUE", mqeMsg, null, 0);
     System.out.println("Finished");
}
catch (Exception e)
{
    e.printStackTrace();
     System.out.println("Failed! + e);
}
```

Migrating MQe applications that use the MQe node on the broker

The use of the MQe node on the broker is no longer recommended. Applications should be migrated to use the MQe bridge instead; the broker then receives an MQ message rather than an MQe message. There are three migration scenarios shown below; they are listed in order, with the most desirable first – however the effort required may be seen to be inversely proportional to the desirability. It is recommended that you read all scenarios carefully and decide on the your approach. As all the scenarios require a MQe bridge, you should download the MQe Server Support SupportPac™ as this will greatly assist in the configuration of the bridge. The MQe Server Support SupportPac is available from: http://www-306.ibm.com/software/integration/support/supportpacs/product.html

Using the MQePubMsgObject

This migration uses the MQe bridge instead of the MQe broker node and avoids the use of any deprecated MQe classes.

The following work is required:

1. Modify the message flow on the broker such that MQ input/output nodes are used instead of MQe input/output nodes.
2. Modify the client application such that all references to the MQeMbMsgObject class are removed and replaced by use of the MQePubSubMsgObject class. The mapping between these two classes and their respective methods is fairly obvious; the MQePubSubMsgObject is easier to use for pub/sub operations because it contains explicit methods for these functions. Setting of any MQ MQMD parameters is very similar in both objects, with minor differences in method names and return types.

   One key difference is that the client application will have sent all its messages to the same queue (i.e. the broker MQe input node); intelligence in the node then determined the correct destination. The application must now send any subscribe and unsubscribe messages to the broker's control queue SYSTEM.BROKER.CONTROL.QUEUE, whilst publication messages must now be sent to the input queue of the appropriate message flow (this flow will contain a Publication node).

   For an example of the application conversion, compare the examples in the *MQI Programming Guide* in Appendix C, "MQSeries Everyplace Nodes", with the converted equivalents in the section "Writing MQe pub/sub applications" on page 9.

   Note: There is a simpler way of modifying the application so that it generates only MQePubSubMsgObject class messages – but it has the disadvantage of perpetuating the use of the deprecated MQeMbMsgObject class in the application. Leave the logic unchanged until the point at which the message is about to be sent. Then use the new method MQeMbMsgObject.toMQePubSubMsgObject which generates the equivalent message object in the preferred class. This object must of course, be sent to the appropriate target queue according to its pub/sub function.

   Likewise for messages being received – an MQePubSubMsgObject class object can be converted to its equivalent MQeMbMsgObject object with the constructor MQeMbMsgObject(MQePubSubMsgObject).
3. Configure an MQe bridge to transfer and transform MQe messages from the MQe network to the MQ network underlying the broker topology. The bridge transformer class must be configured to be com.ibm.mqe.mqbridge.MQeJMSRFHTransformer. Note that two bridge queues must also be configured: one to direct messages to the broker's control queue SYSTEM.BROKER.CONTROL.QUEUE; the other to direct messages to the input queue of the appropriate message flow.

Using the MQeMbMsgObject with the bridge, retaining an efficient message flow

This migration makes minimal changes to the application, yet still uses the MQe bridge instead of the MQe broker node. The downside is that it requires the use of deprecated MQe classes.

The following work is required:

1. Modify the message flow on the broker such that MQ input nodes are used instead of MQe input nodes.
2. Modify the client application such that messages are sent via the MQe bridge. Any subscribe and un-subscription messages are sent to the broker's control queue SYSTEM.BROKER.CONTROL.QUEUE, whilst publication messages must now be sent to the input queue of the appropriate message flow (this flow will contain a Publication node). In all cases the com.ibm.broker.mqimqe.wrapper.MQeMbMsgObject continues to be used.
3. Configure an MQe bridge to transfer and transform MQe messages from the MQe network to the MQ network underlying the broker topology. The bridge transformer class must be configured to be com.ibm.mqe.mqbridge.MQeMbTransformer. Note that two bridge queues must also be configured:

one to direct messages to the broker’s control queue SYSTEM.BROKER.CONTROL.QUEUE; the other to direct messages to the input queue of the appropriate message flow.

Using the MQeMbMsgObject with the bridge and without application change

This migration makes no changes to the application, but still uses the MQe bridge instead of the MQe broker node. The disadvantages are that it requires the use of deprecated MQe classes and may also result in excessive processing within the broker message flow.

The following work is required:

1. Modify the message flow on the broker such that MQ input nodes are used instead of MQe input nodes. Add a filter node into the flow such that (a) any subscribe and un-subscription messages are sent to the broker’s control queue SYSTEM.BROKER.CONTROL.QUEUE, and (b) publication messages are sent to the input queue of the appropriate message flow.

   An example of the ESQL needed for the filter node:

```
BEGIN

        IF   InputRoot.MQRFH2.psc.Command = 'RegSub' OR
             InputRoot.MQRFH2.psc.Command = 'DeregSub' OR
             InputRoot.MQRFH2.psc.Command = 'ReqUpdate' THEN
             RETURN TRUE;
      END IF;

        IF   InputRoot.MQRFH2.psc.Command = 'Publish' OR
          InputRoot.MQRFH2.psc.Command = 'DeletePub' THEN
          RETURN FALSE;
        END IF;

        RETURN UNKNOWN;
END;
```

2. Configure an MQe bridge to transfer and transform MQe messages from the MQe network to the MQ network underlying the broker topology. The bridge transformer class must be configured to be com.ibm.mqe.mqbridge.MQeMbTransformer. One bridge queue only must also be configured, such that it directs all messages to the input queue of the appropriate message flow.
3. If the node’s destination mode was set to `Reply To Queue` and your destination queue manager and destination queue are different than your reply to queue manager and reply to queue, you will need to set a Java™ system property to prevent the destination fields from being copied into the reply to fields in the MD of the MQ message. The system property `"com.ibm.mqe.mqbridge.MQeMbTransformer.keepReplyTo` can be set to any value. To set using a Java program, add the `-Dproperty=value` VM argument. If you are using the MQeExplorer or MQeScript executables from the MQeServerSupport SupportPac, you will need to create a .sp file. Refer to the *MQeServerSupport Installation Guide* for more information.