MQSeries: Message Oriented Middleware

MQSeries™ messaging software enables business applications to exchange information across more than twenty five different operating system platforms in a way that is straightforward and easy for programmers to implement.

Contents

Why Use MQSeries?

IBM's MQSeries simplifies the task of connecting your applications across unlike environments.

Programs communicate using the MQSeries API, an easy-to-use, high-level program interface which shields your programmers from the complexities of different operating systems and underlying networks. You focus on the business logic, while MQSeries manages your connections to the computer systems.

Features

- Advanced program-to-program connectivity for your enterprise.
- Enables application integration throughout your organisation.
- Protects your information. Assured once-only message delivery - even when programs or networks fail.
- Support of most major platforms - both IBM and non IBM.

In an open-systems environment, communications coding is complex and difficult to manage. MQSeries simplifies things, giving you time to develop the solutions that deliver your business mission.

[Return to Contents]

Application programs and messaging

The IBM MQSeries range of products provides application programming services that enable application programs to communicate with each other using messages and queues. This form of communication is referred to as asynchronous messaging. It provides assured, once-only delivery of messages. Using MQSeries means that you can separate application programs, so that the program sending a message can continue processing without having to wait for a reply from the receiver. If the receiver, or the communication channel to it, is temporarily unavailable, the message can be forwarded at a later time. MQSeries also provides mechanisms for generating acknowledgements of messages received.

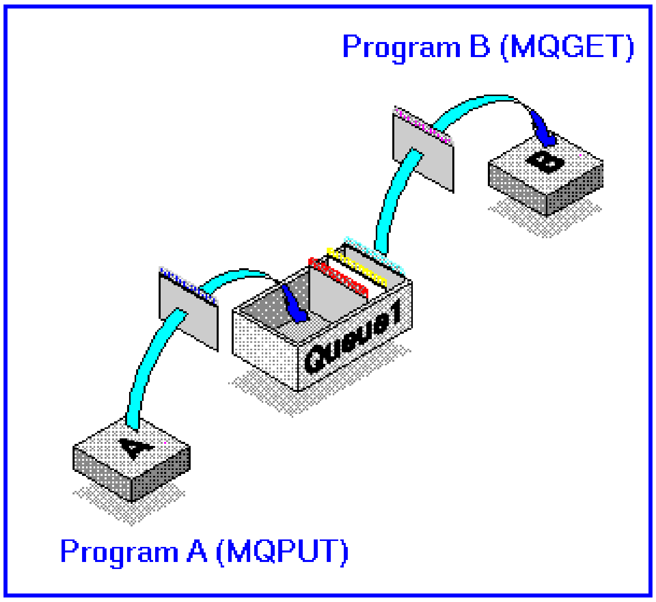
The programs that comprise an MQSeries application can be running on different computers, on different operating systems, and at different locations. The applications are written using a common programming interface known as the Message Queue Interface (MQI), so that applications developed on one platform can be transferred to another.

The figure below shows that when two applications communicate using messages and queues, one application puts a message on a queue, and the other application gets that message from the queue.

Programs connected to the same queue manager

This figure shows how messaging works in the simple case where the program putting the message and the program getting the message are both on the same computer and connected to the same queue manager.

Program A puts messages on the queue; program B gets messages from the queue. In this case, the programs and the queue manager are running on the same workstation.



[Return to Contents]

Queue managers

In MQSeries, queues are managed by a component called a queue manager. The queue manager provides messaging services for the applications and processes the MQI calls they issue. The queue manager ensures that messages are put on the correct queue or that they are routed to another queue manager.

Before applications can send any messages, you must create a queue manager and some queues. MQSeries provides the utilities to help you do this and to create any other MQSeries objects that you need for your applications.

How applications identify themselves to queue managers
Any MQSeries application must make a successful connection to a queue manager before it can make any other MQI calls. When the application successfully makes the connection, the queue manager returns a connection handle. This is an identifier that the application must specify each time it issues an MQI call. An application can connect to only one queue manager at a time, so only one connection handle is valid (for that particular application) at a time. When the application has connected to a queue manager, all the MQI calls it issues are processed by that queue manager until it issues another MQI call to disconnect from that queue manager.

Opening a queue
Before your application can use a queue for messaging, it must open the queue. If you are putting a message on a queue, your application must open the queue for putting. Similarly, if you are getting a message from a queue, your application must open the queue for getting. You can specify that a queue is opened for both getting and putting, if required. The queue manager returns an object handle if the open request is successful. The application specifies this handle, together with the connection handle, when it issues a put or a get call. This ensures that the request is carried out on the correct queue.

Putting and getting messages
When the open request is confirmed, your application can put a message on the queue. To do this, it uses another MQI call on which you have to specify a number of parameters and data structures. These define all the information about the message you are putting, including the message type, its destination, which options are set, and so on. The message data (that is, the application-specific contents of the message your application is sending) is defined in a buffer, which you specify in the MQI call. When the queue manager processes the call, it adds a message descriptor, which contains information that is needed to ensure the message can be delivered properly. The message descriptor is in a format defined by MQSeries; the message data is defined by your application (this is what you put into the message data buffer in your application code).

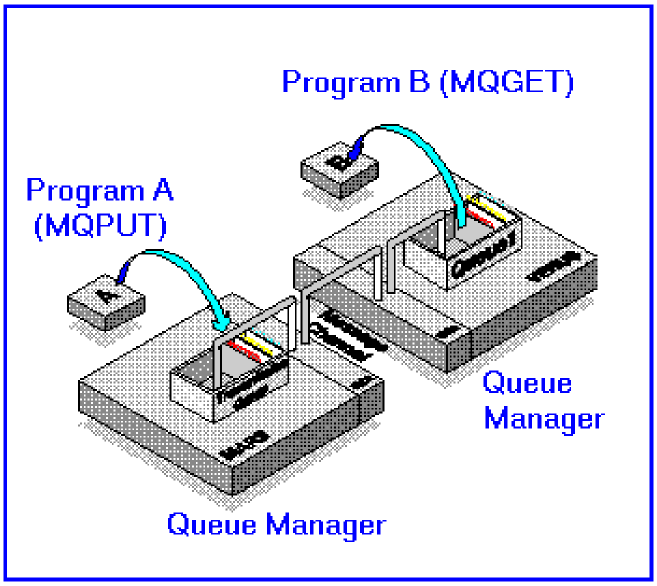
The program that gets the messages from the queue must first open the queue for getting messages. It must then issue another MQI call to get the message from the queue. On this call, you have to specify which message you want to get.

Messaging using more than one queue manager
This arrangement is not typical for a real messaging application because both programs are running on the same computer, and connected to the same queue manager. In a commercial application, the putting and getting programs would probably be on different computers, and so connected to different queue managers.

The figure below shows how messaging works where the program putting the message and the program getting the message are on the different computers and are connected to different queue managers.

In this situation, you also need to create message channels to carry MQSeries messages between the queue managers.

Programs connected to different queue managers



This figure shows how messaging works in the case where the program putting the message and the program getting the message are on different computers and are connected to different queue managers.

Program A makes an MQPUT call specifying not a local queue but a local definition of a remote queue. This definition identifies a queue on another queue manager. The queue manager to which program A is connected puts the message on a special queue called a transmission queue. The message is then automatically sent along the channel connecting the two queue managers. (If the channel is running). The receiving queue manager puts the message on the queue that was originally specified by program A. This message is stored on the queue until it is retrieved by program B, after program B has issued an MQGET call.

[Return to Contents]

Commercial Messaging

For true commercial messaging, middleware needs to provide more than assured delivery. MQSeries is designed to meets the requirements of real business systems:

Transactional Integrity
MQSeries supports transactional messaging, which means that operations on

messages can be grouped into 'units of work'. A unit of work is either committed in its entirety, or backed-out, so that it's as if none of the operations took place. This means that data is always in a consistent state.

For example, an application gets a message off a queue, processes the data in the message and creates five further messages, to be sent to other applications, all within one unit of work. If the output queue for the last of the five messages is temporarily full, then the application can backout the entire unit of work, so the input message that it got from the queue will be returned to the queue, and the first four messages that were put onto output queues will be removed. After the backout, it's as if the application had never got the message in the first place. The application can then try the processing again, at a later time.

MQSeries can also coordinate units of messaging work with other transactional work, like database updates, so that message data and database data remain completely in-sync at all times.

Security
Security is one of the most important aspects for a distributed system and MQSeries provides a flexible security framework that allows the appropriate security architecture to be implemented to meet your specific requirements.

There are several aspects to the MQSeries security model:

- *MQSeries Administration Commands*
  Administration of MQSeries is performed using a number of commands. For instance, create queue manager, start and stop queue manager. Control of individual user's access to these commands is via normal operating system controls, like access control lists.

- *Access to Queue Manager Objects*
  MQSeries allows you to control application access to queue manager objects, like the queue manager itself, and its queues. MQSeries on MVS conforms to the SAF interface and so you can use an external security manager, like RACF or TopSecret, to control application access to a queue manager. MQSeries on most other platforms provides a security manager, which allows you to specify access controls for queue manager objects. If the security manager doesn't do exactly what you need, then you can write your own, because MQSeries provides a documented interface for security management.

- *Channel Security*
  To protect the information flowing between MQSeries queue managers, MQSeries supports exit points into which you can link security modules. For example, when two queue managers initiate communications via an MQSeries channel, each queue manager can verify the identity of the other, before exchanging any data. Similarly, during message transmission, data can be encrypted before it's sent over the network and decrypted when it arrives at the receiving end.

- *Application Security*
  The MQI programming interface provides facilities for applications to identify themselves, both by platform and application, and by principle (or user identifier). This information is propagated with the message, and only privileged applications can change it. Applications can therefore use this information to make extra security checks on messages that they receive.

Triggering
The asynchronous nature of message queuing may mean that applications are idle for periods of time when there are no messages to process. To avoid having idle processes consume system resources while there is no work to do, MQSeries provides a mechanism to 'trigger' applications to start when certain conditions are met.

Triggering works by defining a specific condition for an application's queue, which when met, will cause the queue manager to send a trigger message to an MQSeries queue called an 'initiation queue'. The trigger message is processed by a special application called a trigger monitor, which reads the trigger message from the initiation queue and uses the information in the message to decide which application to start to process the messages on the application's queue.

By using a trigger monitor you can have a single process that initiates many application processes to handle messages arriving on many different queues, as required.

Thin Clients
The facilities provided by MQSeries queue managers provide robust and reliable queuing, but they also consume resources on the computer where they run, for example, disk space to store the messages on queues. Some applications are designed to be lightweight, but need to take advantage of MQSeries' asynchronous messaging facilities.

With MQSeries thin clients, applications can use the same MQI programming interface, but without the need for a queue manager running on their local system. The thin client sends the MQI call to a server machine over an MQSeries channel. The work is performed by the server on behalf of the client, and the results are passed back to the client. It's very much like Remote Procedure Calls (RPC), but using MQSeries channels.

[Return to Contents]

Where to Find More Information

http://www.software.ibm.com/ts/mqseries

[Return to Contents]

MQSeries Platform Coverage

[Return to Contents]