Technical Introduction to IBM WebSphere MQ

(Course code WM100 / VM100)

Student Notebook

ERC 1.0

WebSphere Education

Trademarks

IBM® is a registered trademark of International Business Machines Corporation.

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

AIX®	CICS®	DB2®
Domino®	e-business on demand®	Everyplace®
IMS™	iSeries®	Lotus®
Maximo®	MVS™	MVS/ESA™
OS/390®	OS/400®	RACF®
Redbooks®	SupportPac™	System/390®
VSE/ESA™	VTAM®	WebSphere®
z/OS®	zSeries®	

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft and Windows 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.

Linux® is a registered trademark of Linus Torvalds in the United States, other countries, or both.

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

April 2008 edition

Contents

Trademarks

The reader should recognize that the following terms, which appear in the content of this training document, are official trademarks of IBM or other companies:

IBM® is a registered trademark of International Business Machines Corporation.

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

AIX®	CICS®	DB2®
Domino®	e-business on demand®	Everyplace®
IMS™	iSeries®	Lotus®
Maximo®	MVS™	MVS/ESA™
OS/390®	OS/400®	RACF®
Redbooks®	SupportPac™	System/390®
VSE/ESA™	VTAM®	WebSphere®
z/OS®	zSeries®	

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft and Windows 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.

Linux® is a registered trademark of Linus Torvalds in the United States, other countries, or both.

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

Course description

Technical Introduction to IBM WebSphere MQ

Duration: 1 day

Purpose

This course is designed to provide a technical introduction to WebSphere MQ.

Audience

Technical personnel who wish to find out more about WebSphere MQ

Prerequisites

Before taking this course, you should have previous experience in working with software. Skills and experience in one or more of the following specific areas will enable you to derive more benefit from the course:

- Communications and networking
- System and network management
- System design
- Application development
- Transaction processing
- Database management
- Client/server solutions
- Platform knowledge (IBM and non-IBM)
- Open systems

Objectives

After completing this course, you should be able to:

- Compare and contrast WebSphere MQ with other forms of program-to-program communication
- Identify the impact of WebSphere MQ on application design

- Describe the basic components and structure of WebSphere MQ (for example, a message, a queue, a queue manager, a channel, a cluster, and so forth)
- Describe the function of various calls in the Message Queue Interface (MQI)
- Describe the tasks that need to be performed in order to manage a queue manager and its connections with other queue managers and with WebSphere MQ client applications
- Describe the transactional support within WebSphere MQ
- Describe those features of WebSphere MQ that contribute to system security
- Explain how WebSphere MQ can be used as part of the communications infrastructure to connect application environments such as the World Wide Web, and enterprise transaction and database systems
- Explain the functionality and example application of publication/subscription provided with IBM WebSphere MQ V7.

Curriculum relationship

This course is a prerequisite to the other WebSphere MQ courses.

Agenda

Day 1

Welcome
Unit 1 — Introduction to WebSphere MQ
Unit 2 — Programming with WebSphere MQ
Unit 3 — Messages — additional information
Unit 4 — Intercommunication
Unit 5 — System administration
Unit 6 — Transactional support
Unit 7 — Security
Unit 8 — Linking, bridging, and the WebSphere MQ family

Unit 1. Introduction to WebSphere MQ

What this unit is about

This unit provides an overview of IBM WebSphere MQ. It introduces the business requirements that led to the development of IBM WebSphere MQ, positions it against other forms of program-to-program communication, and discusses the function it provides for building business application solutions.

What you should be able to do

After completing this unit, you should be able to:

- Explain the positioning of messaging and queuing in today's business environment
- Provide a high-level description of WebSphere MQ functions
- Describe the breadth of coverage of WebSphere MQ products

How you will check your progress

Accountability:

- Instructor questions
- Checkpoint questions

References

www.ibm.com/software/integration/wmq
IBM WebSphere MQ Homepage

www.ibm.com/software/integration/wmq/library
IBM WebSphere MQ Library

www.ibm.com/software/integration/wmq/quicktour
IBM WebSphere MQ Quick Tour

GC33-0805 *An Introduction to Messaging and Queuing*

Unit objectives

After completing this unit, you should be able to:

- Explain the positioning of messaging and queuing in today's business environment
- Provide a high-level view of WebSphere MQ functions
- Describe the breadth of coverage of WebSphere MQ products

Figure 1-1. Unit objectives WM100 / VM1001.0

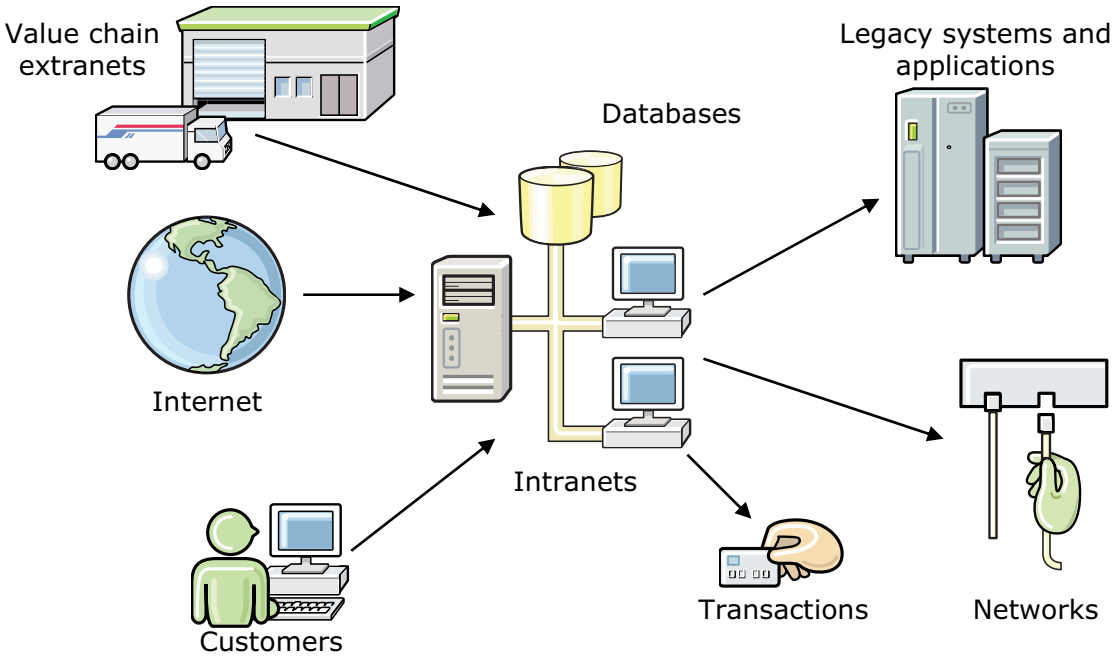
Notes:

The intent of this course is to provide an introductory technical overview of IBM WebSphere MQ. The topics covered will allow you to gain a solid foundation to begin building WebSphere MQ skills.

The purpose of this unit is to identify and understand the types of business requirements solved by IBM WebSphere MQ, and to introduce the manner in which IBM WebSphere MQ meets those requirements.

Today's enterprise IT environment

IT environments are increasingly heterogeneous and complex



The role of modern middleware is to integrate and simplify

Figure 1-2. Today's enterprise IT environment WM100 / VM1001.0

Notes:

Perhaps the biggest challenge is that today's IT environments are becoming increasingly diverse. As the focus on integration increases, it moves horizontally and extends beyond the boundaries of the individual enterprise to include partners, suppliers, and customers. It is this diversity in IT environments that is increasing the complexity of the challenge that a company like yours has to deal with. That is where middleware comes in.

Middleware is the infrastructure software that simplifies the problem of horizontal integration. The infrastructure has to integrate people, data, and applications across and beyond the enterprise to provide benefits throughout the value chain. It is infrastructure middleware that provides the operational resilience that is essential to make the technology transitions that allow your business to react quickly to the necessary business changes and to adapt dynamically in this new business world.

IBM WebSphere reference architecture

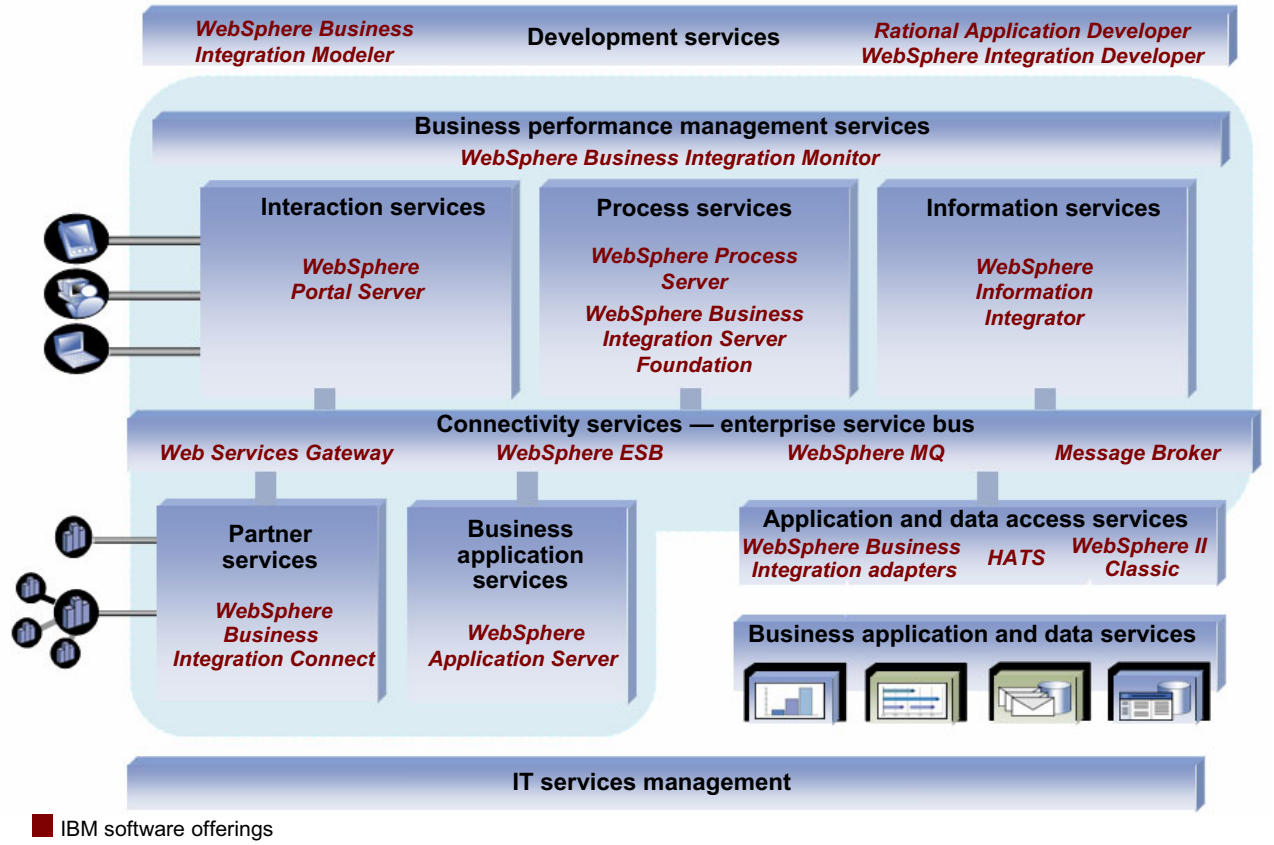
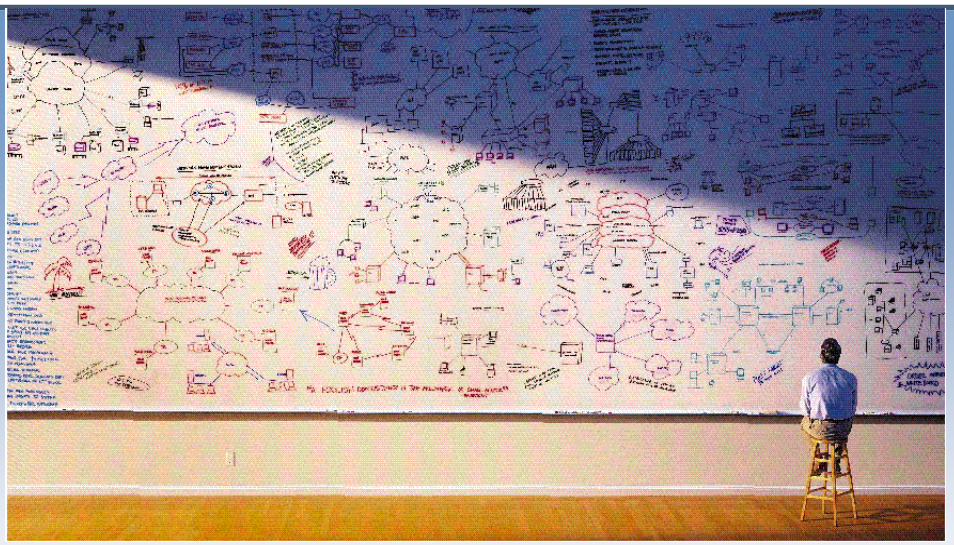


Figure 1-3. IBM WebSphere reference architecture WM100 / VM1001.0

Notes:

The WebSphere platform provides the software foundation for On Demand Business. IBM WebSphere MQ started as a transactional application server and has grown with customer needs to provide software products delivering a breadth of business integration. While it emphasizes the value of the entire platform, IBM also allows the customer to start simply, using just the functionality required at the time, and then to progressively add function as needed. The IBM middleware platform is implemented as a service-oriented architecture (SOA), which IBM calls the IBM WebSphere Integration Reference Architecture.

Why are interfaces so expensive to build and maintain?



- **Application interface logic is intertwined with business logic.**
- **Tightly integrated interfaces are difficult to change.**
- **The more interfaces, the more complex the application — interface logic may exceed business logic.**
- **In such circumstances, reuse becomes difficult and impractical.**

Figure 1-4. Why are interfaces so expensive to build and maintain? WM100 / VM1001.0

Notes:

Why are interfaces so expensive to build and maintain?

There are several reasons: Application interface logic is typically intertwined within application business logic. This is usually due to the nature of the applications themselves. The programming models do not enable the separation of interface logic from the applications themselves.

The more tightly integrated the interface, the more difficult the application is to change.

The more interfaces within a program, the more complex the application becomes. Over time, and with enough separate connections, the interface logic can, in fact, exceed the business logic. In such circumstances, reuse becomes difficult and impractical.

SOA is the methodology and architecture for solving this problem.

Service-oriented architecture — SOA

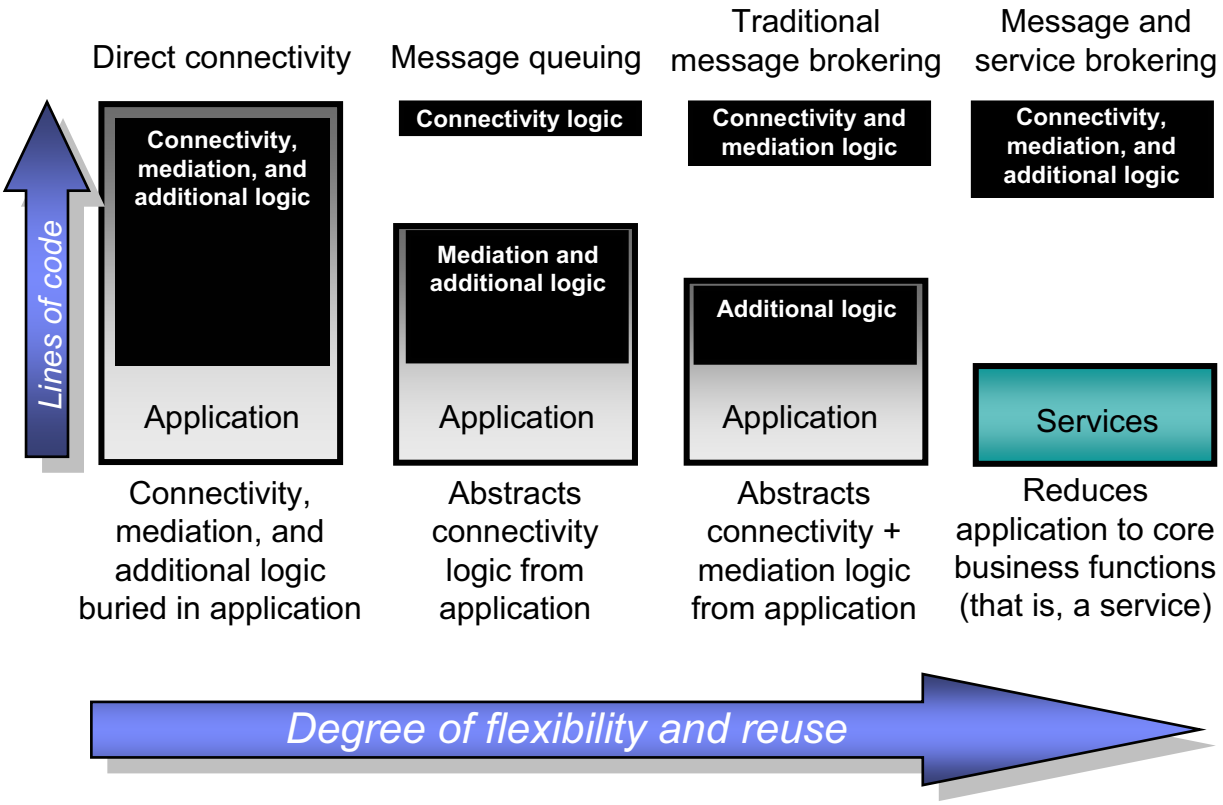


Figure 1-5. Service-oriented architecture — SOA WM100 / VM1001.0

Notes:

SOA is not a new concept. It is the next step of a connectivity evolution that has been going on for some time, and indeed, much of the problem has already been solved with existing technologies that you may be using today.

The objective of SOA is to reduce the service down to the bare business logic.

Program-to-program communication

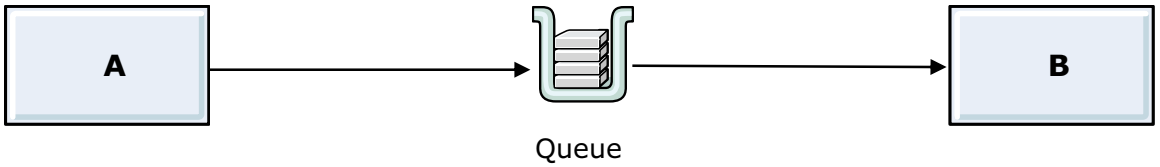


Figure 1-6. Program-to-program communication WM100 / VM1001.0

Notes:

IBM WebSphere MQ is a means of program-to-program communication.

The figure depicts the basic mechanism by which this communication takes place. Program A prepares a message and puts it on a queue. Program B then gets the message from a queue and processes it.

Both Program A and Program B use an application programming interface (API) to put messages on a queue and get messages from a queue. The IBM WebSphere MQ API is called the *Message Queue Interface* (MQI).

Note that when Program A puts a message on the queue, Program B may not be executing. The queue stores the message safely until Program B starts and is ready to get the message. Likewise, at the time when Program B gets the message from the queue, Program A may no longer be executing. Using IBM WebSphere MQ, there is no requirement for two programs communicating with each other to be executing at the same time.

IBM WebSphere MQ provides several functions that are mandatory for successful message transport:

- Assured delivery
- Once-only delivery
- Asynchronous delivery

Synchronous application design model

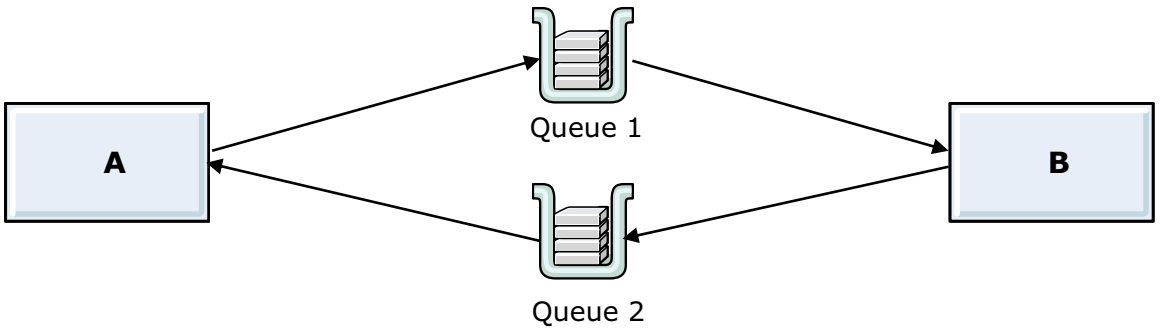


Figure 1-7. Synchronous application design model WM100 / VM1001.0

Notes:

The figure shows how Program B can send a message to Program A using the same mechanism. The message may be a reply to a message it received from Program A. Typically, Program B uses a different queue to send messages to Program A. Using a separate queue is not strictly necessary, but doing so leads to a simpler application design and simpler programming logic.

If Program A sends a message to Program B and expects a reply, one option is for Program A to put a message on Queue 1 and then wait for the reply to appear on Queue 2. This is called the *synchronous model* for two-way communication between programs.

Using the synchronous model, Program A and Program B would normally be executing at the same time. However, if Program B fails, Program A might potentially have to wait a long time for a reply. How long Program A should wait before continuing with other processing is a design issue.

Extended asynchronous application design model

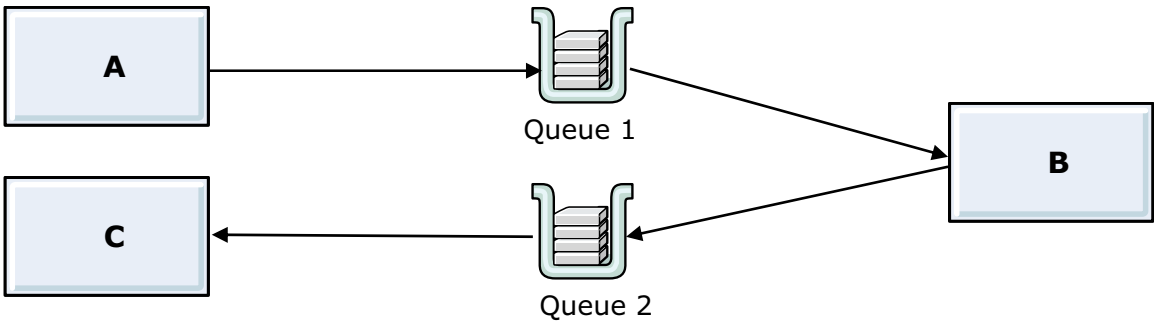


Figure 1-8. Extended asynchronous application design model WM100 / VM1001.0

Notes:

Using the extended asynchronous model, Program A puts messages on Queue 1 for Program B to process, but it is Program C, acting asynchronously to Program A, which receives replies from Queue 2 and processes them. Typically, Program A and Program C would be part of the same application.

The asynchronous model is a natural model for IBM WebSphere MQ. Program A can continue to put messages on Queue 1 and is not blocked by having to wait for a reply to each message. It can continue to put messages on Queue 1 even if Program B fails. In that case, Queue 1 stores the messages safely until Program B is restarted.

In a variation of the asynchronous model, Program A could put a sequence of messages on Queue 1, optionally continue with other processing, and then return to get and process the replies from Queue 2 itself.

Time independence

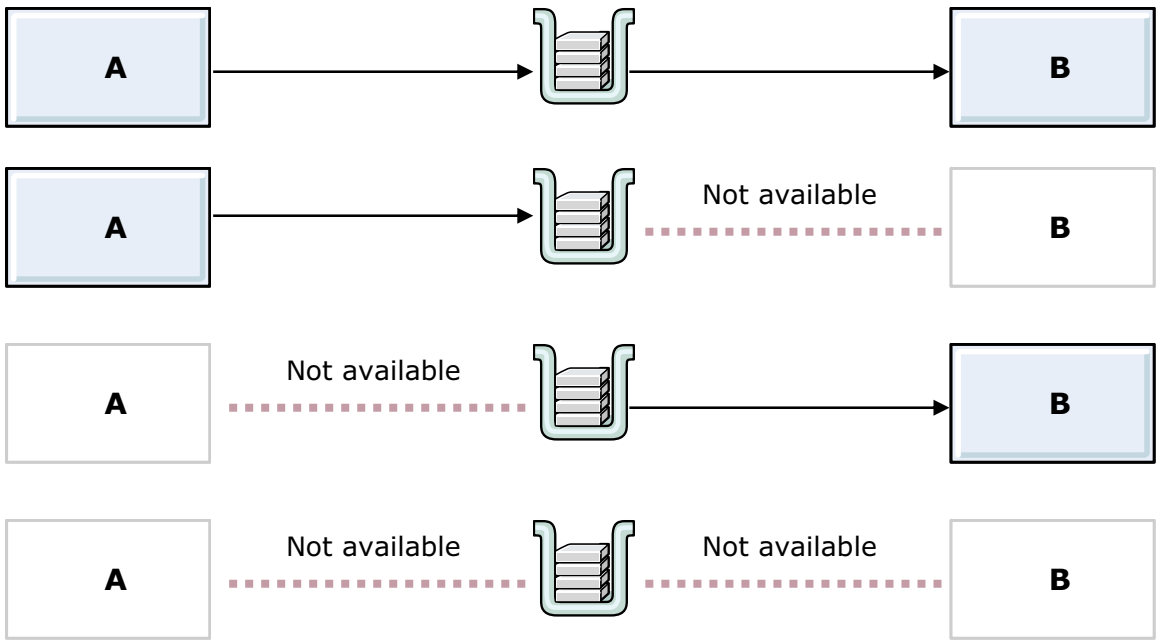


Figure 1-9. Time independence WM100 / VM1001.0

Notes:

This figure demonstrates that Program A puts messages on the queue and Program B gets them when it is ready. If Program B is busy or is not available, the messages are stored safely in the queue until it is ready to get them.

If, at the point when B becomes ready, Program A has completed its processing or has failed, it does not matter. Program B can still get the messages and process them. Indeed, there may be times when neither program is available, but any outstanding messages are still stored safely in the queue.

This property of IBM WebSphere MQ, in which communicating applications do not have to be active at the same time, is known as *time independence*.

Three styles of communication

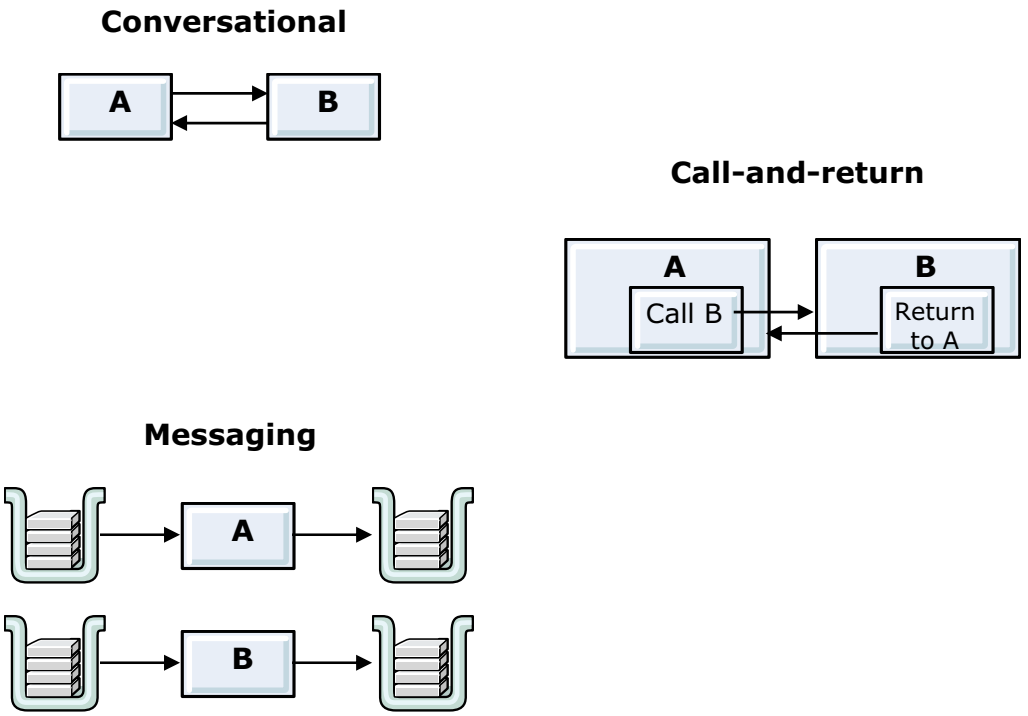


Figure 1-10. Three styles of communication WM100 / VM1001.0

Notes:

Conversational or transaction-oriented communication is characterized by two or more programs executing simultaneously in a cooperative manner in order to perform a transaction. They communicate with each other through an architected interface. While one program is waiting for a reply from another program with which it is cooperating, it may continue with other processing. APPC, CPI-C, and the sockets interface to TCP/IP are examples of this style of communication.

The *call-and-return* style is similar, but when one program calls another program, the first program is blocked and cannot perform any other processing until the second program returns control to it. Remote procedure call (RPC) is an example of this style of communication.

In the *messaging* style, communicating programs can execute independently of each other. An executing program receives input in the form of messages and also outputs its results as messages. A message that is the output from one program becomes the input to another program, but there is no requirement that the latter must be executing when the former outputs the message. Contrast this with the conversational and call-and-return styles where all cooperating partners must be executing at the same time. IBM WebSphere MQ uses this messaging style.

WebSphere MQ eliminates application network concerns

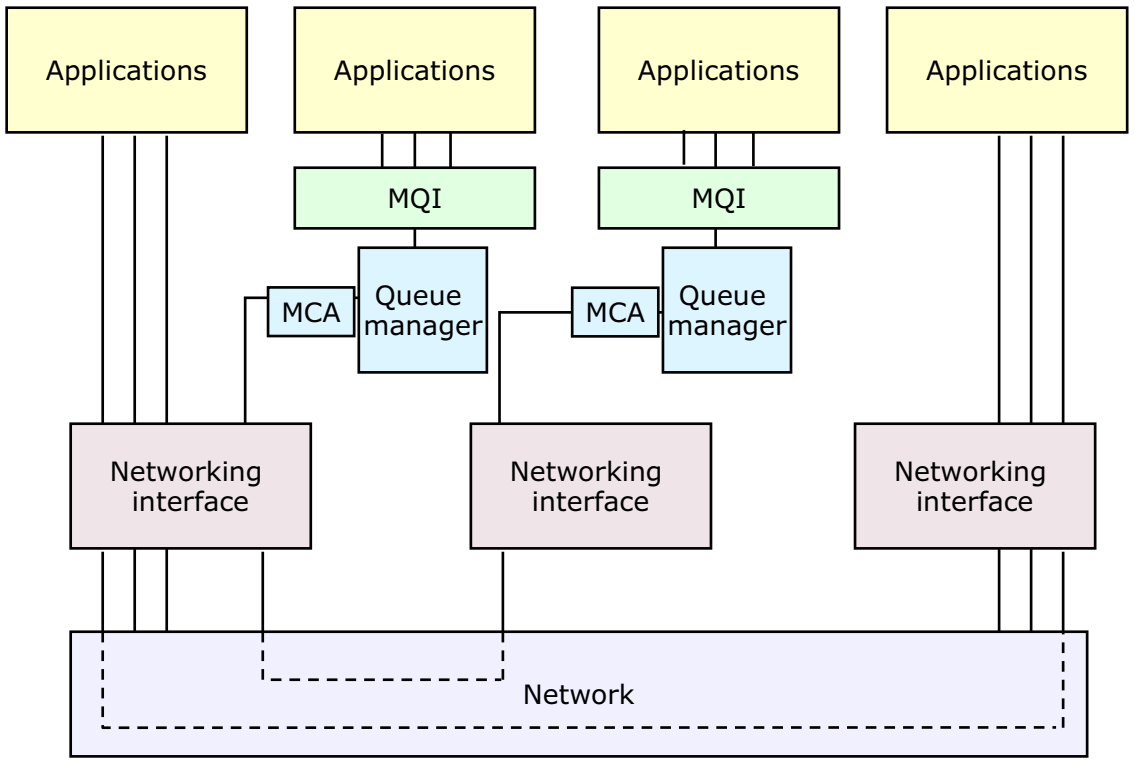


Figure 1-11. IBM WebSphere MQ eliminates application networking concerns WM100 / VM1001.0

Notes:

The conversational style of program-to-program communication relies on a communications connection existing across a network for each pair of applications. In reality, a communications connection manifests itself as a TCP connection, an SNA LU6.2 (system network architecture logical unit) conversation, a NetBIOS session, and so on.

In IBM WebSphere MQ, an application sends a message to another application by using the Message Queue Interface (MQI) functions provided by the queue manager to which it is connected. Therefore, the required communications connection is between a pair of message channel agents (MCAs), each connected to its respective queue manager, not between a pair of applications.

Note how the Message Queue Interface shields applications and their developers from the complexities of the network. The message channel agents supplied with IBM WebSphere MQ contain all the communications programming that is necessary.

Local and remote queues concept

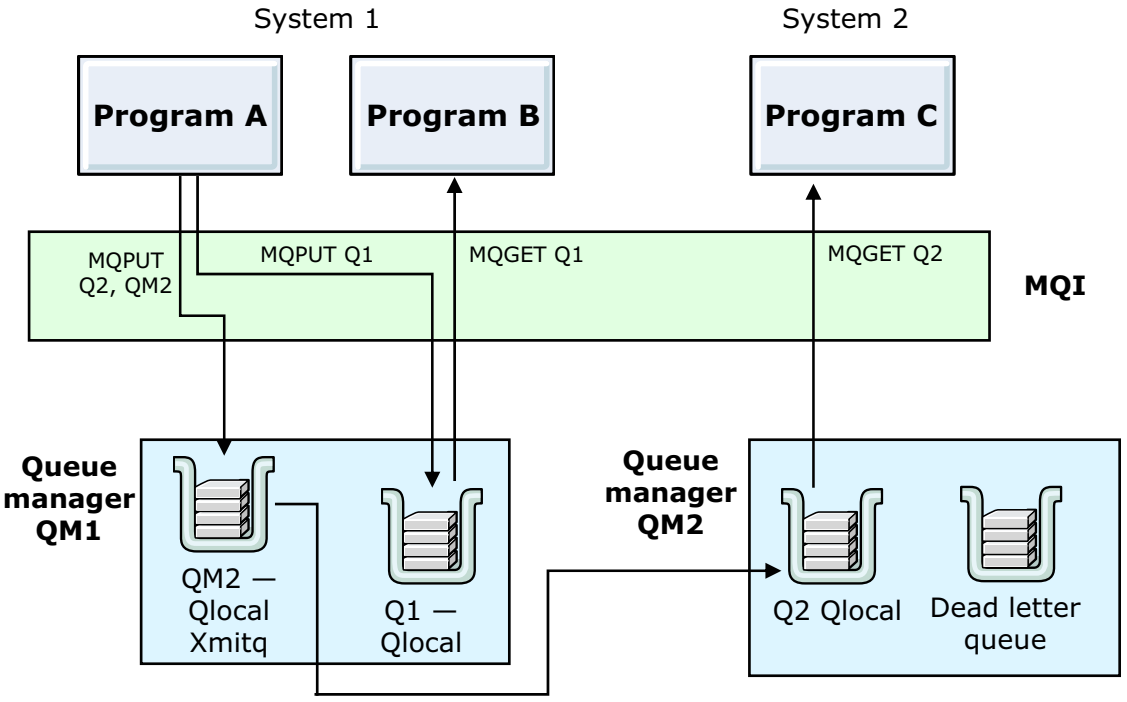


Figure 1-12. Local and remote queues concept WM100 / VM1001.0

Notes:

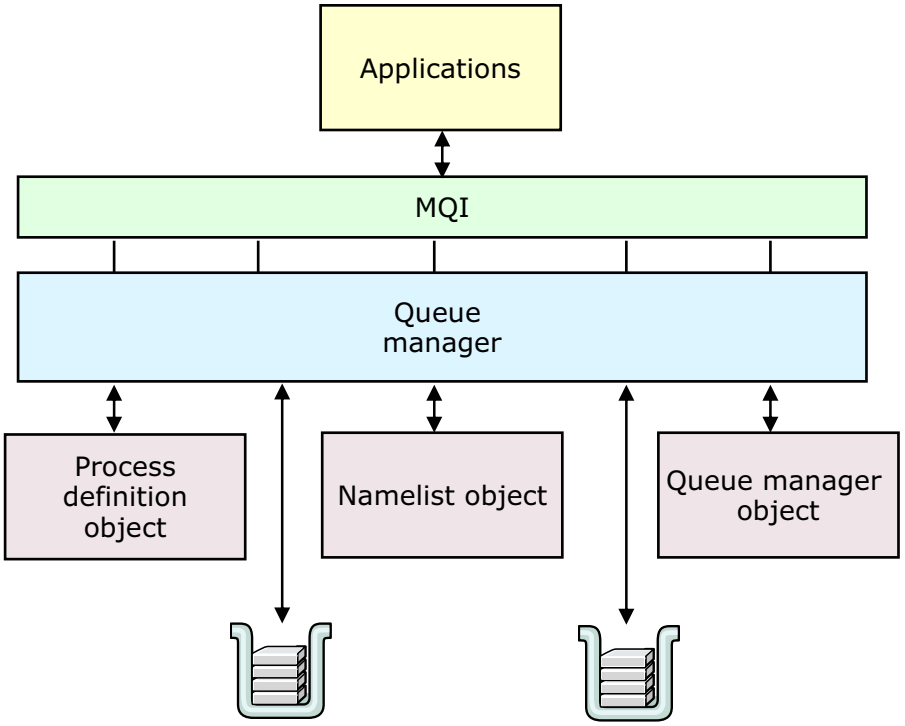
When an application opens a queue, the queue manager determines whether the ultimate destination queue is owned by the queue manager to which the application is connected (a *local* queue), or whether it is one owned by another queue manager (a *remote* queue).

When the application subsequently puts a message on a queue that is local, the queue manager places the message directly on that queue. However, if the queue is remote, the queue manager places the message instead on a special local queue called a *transmission* queue.

It is then the task of message channel agents, supplied components of IBM WebSphere MQ software, to get the message from the transmission queue and send it over the network to a message channel agent at the receiving end. The receiving MCA then puts the message on the destination queue. Once the message has been safely committed on the destination queue, it is removed from the transmission queue.

If the receiving MCA cannot put the message on the destination queue for any reason, the message either will be placed in the dead letter queue associated with that queue manager, or the message will be discarded, depending on the options specified by the sending application in the message descriptor.

MQI calls



- Major calls
  - MQCONN
  - MQCONNX
  - MQDISC
  - MQOPEN
  - MQCLOSE
  - MQPUT
  - MQPUT1
  - MQGET
  - MQSUB
  - MQSUBRQ
- Minor calls
  - MQBEGIN
  - MQCMIT
  - MQBACK
  - MQINQ
  - MQSET

Figure 1-13. MQI calls WM100 / VM1001.0

Notes:

The component of IBM WebSphere MQ software that owns and manages queues is called a *queue manager*.

A queue manager also provides the *Message Queue Interface* (MQI) to enable an application to access its queues and the messages they contain. The MQI is a simple application programming interface that is consistent across all platforms supported by IBM WebSphere MQ. The MQI effectively protects applications from having to know how a queue manager physically manages messages and queues.

An application must first connect to a queue manager before it can access any of its resources. To do this, it issues an **MQCONN** or **MQCONNX** call. When the application no longer needs to be connected to the queue manager, it issues an MQDISC call.

To access a queue, an application must first issue an **MQOPEN** call. When it no longer requires access to the queue, the application issues an **MQCLOSE** call.

Once a queue is opened, an application uses an **MQPUT** call to put a message on the queue and an **MQGET** call to get a message from the queue. An MQPUT with a `Topic`

`String` is also used to publish information which can be establish for subscribers of the published information. The MQCLOSE call releases access to the queue. The **MQPUT1** call enables an application to open a queue, put one message on the queue, and close the queue, all in a single call.

The **MQPUT1**, **MQCMIT**, and **MQBACK** calls enable an application to put and get messages as part of a unit of work.

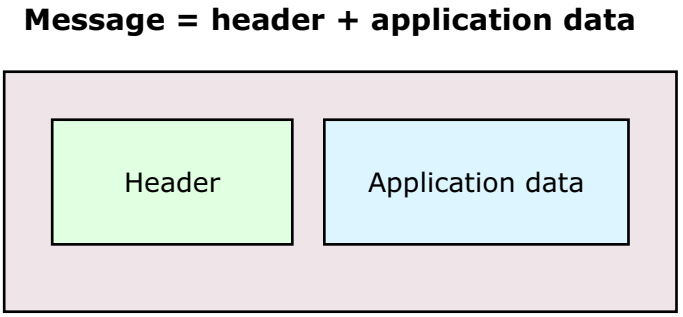
A queue is an example of a IBM WebSphere MQ object. However, there are other types of WebSphere MQ objects, such as a process definition, a namelist, and queue manager objects.

Introduced in IBM Web Sphere MQ V7, using MQPUT with ***topics***, or ***topic strings*** enable the publication of information. IBM WebSphere MQ enable the subscription to publication information via the MQSUB, and MQSUBR which will be outlined in Unit 8 - Linking

Every IBM WebSphere MQ object has a set of attributes which describe the object. Each attribute has a name and a value. The definition of a IBM WebSphere MQ object specifies the values of its attributes. Every IBM WebSphere MQ object also has a name, which is considered to be one of its attributes. An application can use an MQINQ call to inquire on the values of the attributes of an object. It can use an MQSET call to set the values of certain attributes of a queue.

IBM WebSphere MQ has two additional application programming interfaces: the Java interface for use in Java applications, and the Java Message Service (JMS), which allow programmers to write event-based messaging applications.

Message composition



- Set by application and queue manager
- Header
  - MQMD
- Application data
  - Any sequence of bytes
  - Meaningful only to the sending and receiving applications
  - Not meaningful to the queue manager

Figure 1-14. Message composition WM100 / VM1001.0

Notes:

A message is composed of two parts: the various *headers* used by IBM WebSphere MQ, and the *application data*.

All WebSphere MQ messages always have a header called the MQ *message descriptor* (MQMD). The message descriptor contains control information about the message that is used by both the queue manager and the receiving application. It contains a number of fields that you review later in the course.

The application data is meaningful only to the applications that send or receive the message. It is possible to have a message with only a header and no application data.

There is no restriction on the content of the application data, but there is a maximum allowable length of 100 MB for a physical message. However, later in the course you will see that messages larger than that size can still be processed.

Parallel processing — application design

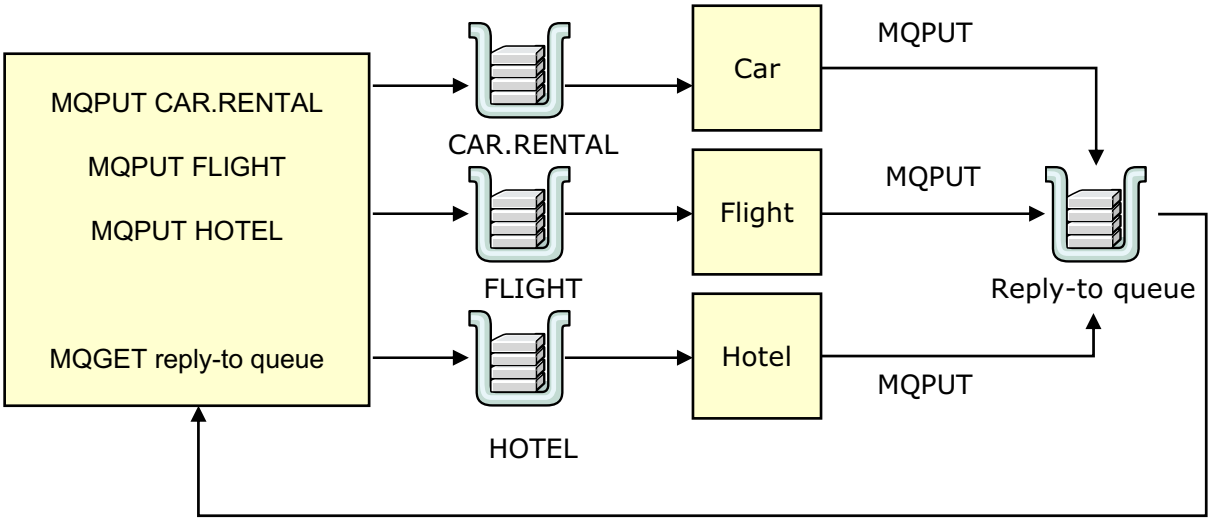


Figure 1-15. Parallel processing — application design WM100 / VM1001.0

Notes:

To book a vacation with a travel agent may require a number of tasks. In the scenario depicted in the figure, an agent needs to reserve a flight and a hotel room, and rent a car. All of these tasks must be performed before the overall business transaction can be considered complete.

Using IBM WebSphere MQ, a request message can be put on each of three queues which serve the car rental application, the flight reservations application, and the hotel reservations application. Each application can perform its respective task in parallel with the other two and then put a reply message on the reply-to queue. The agent's application can then get the three replies and produce a consolidated answer.

This model allows several requests to be sent by an application without the application having to wait for a reply to one request before sending the next. All the requests can then be processed in parallel. Designing the system in this way can improve the overall response time.

Triggering

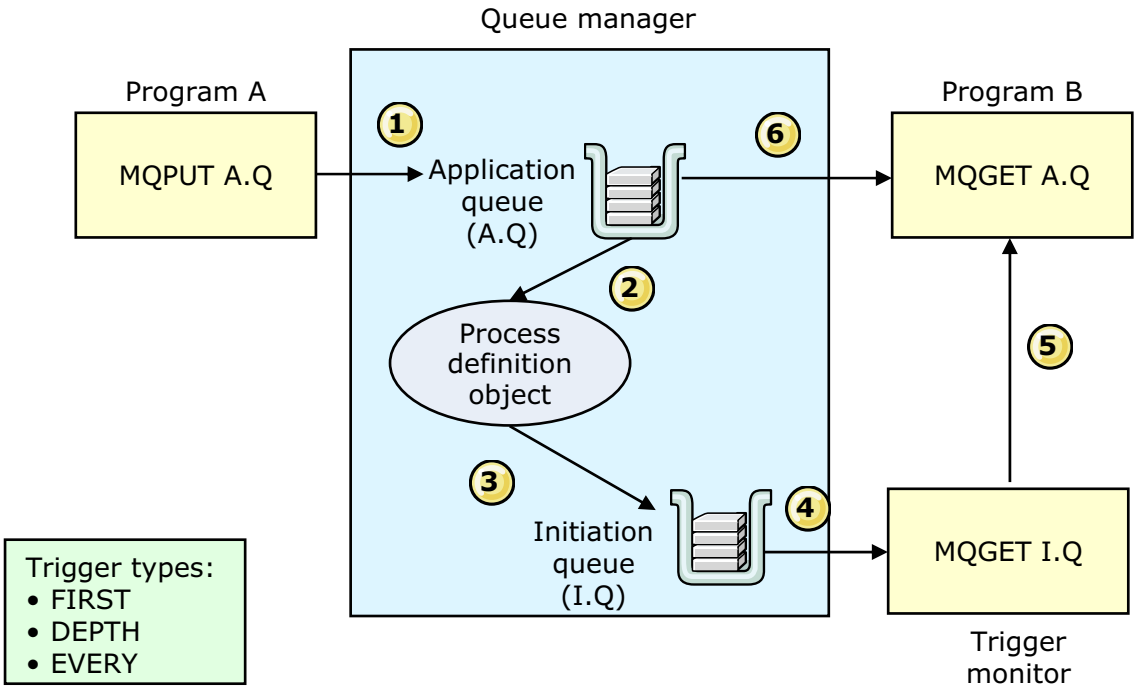


Figure 1-16. Triggering WM100 / VM1001.0

Notes:

In IBM WebSphere MQ, it is possible to arrange for an application to be started automatically when a message is put on a queue and certain conditions are satisfied. This facility is known as *triggering*.

The figure depicts the sequence of actions involved in triggering.

1. Program A puts a message on an application queue which is enabled for triggering.
2. If the conditions for triggering are met, a *trigger event* occurs, and the queue manager examines the process definition object referenced by the application queue. The process definition object identifies the application to be started, namely Program B.
3. The queue manager creates a *trigger message* whose fields contain information copied from certain attributes of the process definition object and the application queue, including the name of the application queue. The queue manager puts the trigger message on an *initiation queue*.
4. A long-running program called a *trigger monitor* gets the trigger message from the initiation queue, and examines its contents.

5. The trigger monitor starts Program B, passing it information from the trigger message as a parameter, including the name of the application queue.
6. Program B opens the application queue and gets messages from it.

Triggering can occur based on selected conditions:

- The first time a message arrives on a queue
- When the number of messages on the queue reaches a predefined number
- Every time a message arrives on a queue

Client/server — application model

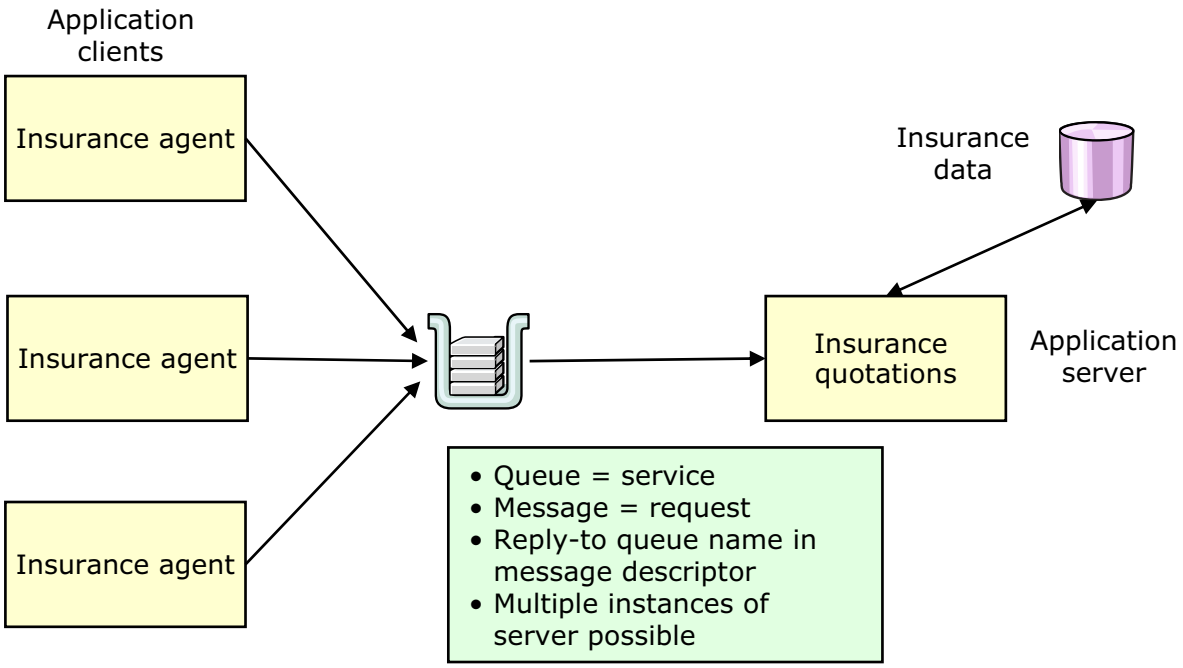


Figure 1-17. Client/server — application model WM100 / VM1001.0

Notes:

The server application (insurance quotations) can handle requests from multiple client applications. Each application client is identical, and requests information from the application server. The message descriptor in each of the incoming messages identifies the appropriate *reply-to queue* for each request, so that the application server knows where to route the reply message to.

WebSphere MQ client

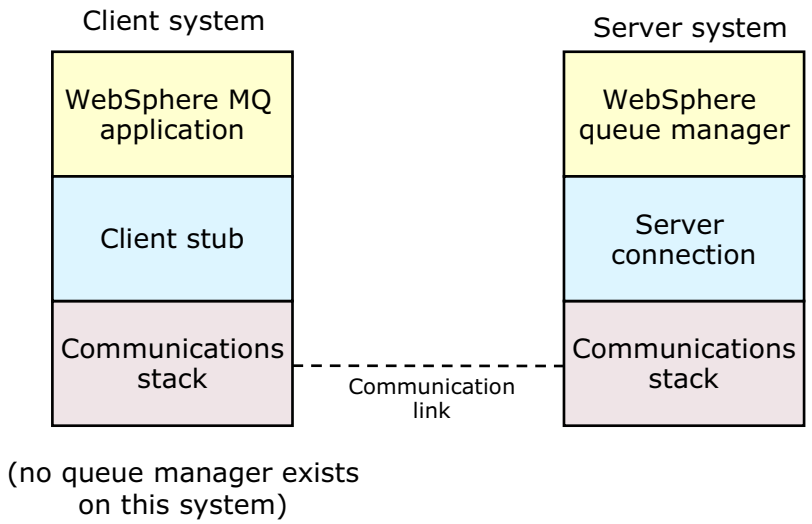


Figure 1-18. WebSphere MQ client WM100 / VM1001.0

Notes:

A *WebSphere MQ client* is a component of IBM WebSphere MQ that allows an application running on a system where no queue manager exists to issue MQI calls to a queue manager running on another system.

The *client stub* receives the input parameters of an MQI call from the application and sends them over a communications connection to the *server connection.* The server connection is on the same system as the queue manager. The server connection then issues the MQI call to the queue manager on behalf of the application. After the queue manager has completed the MQI call, the server connection sends the output parameters of the call back to the client stub, which then passes them to the application.

Most MQI calls and options are available to the client application. The application simply issues an **MQCONN** (or **MQCONNX** where supported) call to connect to a queue manager. Special consideration of MQ client and database units of work will be discussed later.

Not surprisingly, a reliable communications connection is required.

Data integrity

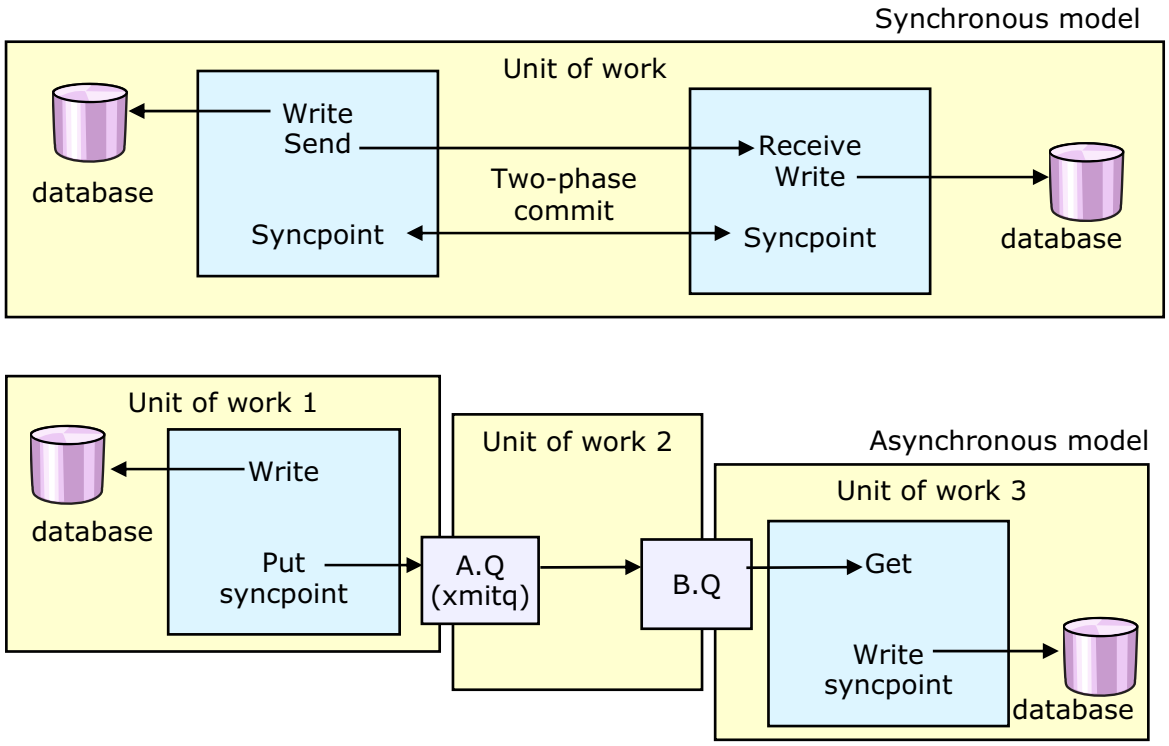


Figure 1-19. Data integrity WM100 / VM1001.0

Notes:

Some implementations of the conversational style of program-to-program communication support the implementation of a distributed unit of work using a two-phase commit protocol. However, this kind of function is only necessary when there is an absolute business requirement to maintain two or more distributed databases in sync. If no such requirement exists, using a single distributed unit of work can be resource intensive and complex, particularly if many processes are involved. IBM WebSphere MQ, on the other hand, offers a simple solution involving multiple units of work acting asynchronously.

The IBM WebSphere MQ solution is depicted in the lower half of the figure.

- The first application writes to a database, puts a message on queue `A.Q`, and then issues a syncpoint to commit the changes to the two resources. The message contains data which is to be used to update a second database on a separate system. As the queue is a remote queue, the message gets no further than the transmission queue within this unit of work. When the unit of work is committed, the message becomes available for retrieval by the sending MCA.

- In the second unit of work, the sending MCA gets the message from the transmission queue and sends it to the receiving MCA on the system containing the second database. The receiving MCA then puts the message on the destination queue. All this is performed reliably because of the assured delivery property of IBM WebSphere MQ. When this unit of work is committed, the message becomes available for retrieval by the second application.
- In the third unit of work, the second application gets the message from the destination queue and updates the database using the data contained in the message.

It is the transactional integrity of units of work 1 and 3, and the “once and once only” and assured delivery properties of IBM WebSphere MQ used in unit of work 2, which ensure the integrity of the complete business transaction.

If the business transaction is a more complex one, multiple units of work may be involved.

Note: Using the IBM WebSphere MQ Client requires the use of an XA compliant transaction manager.

Security

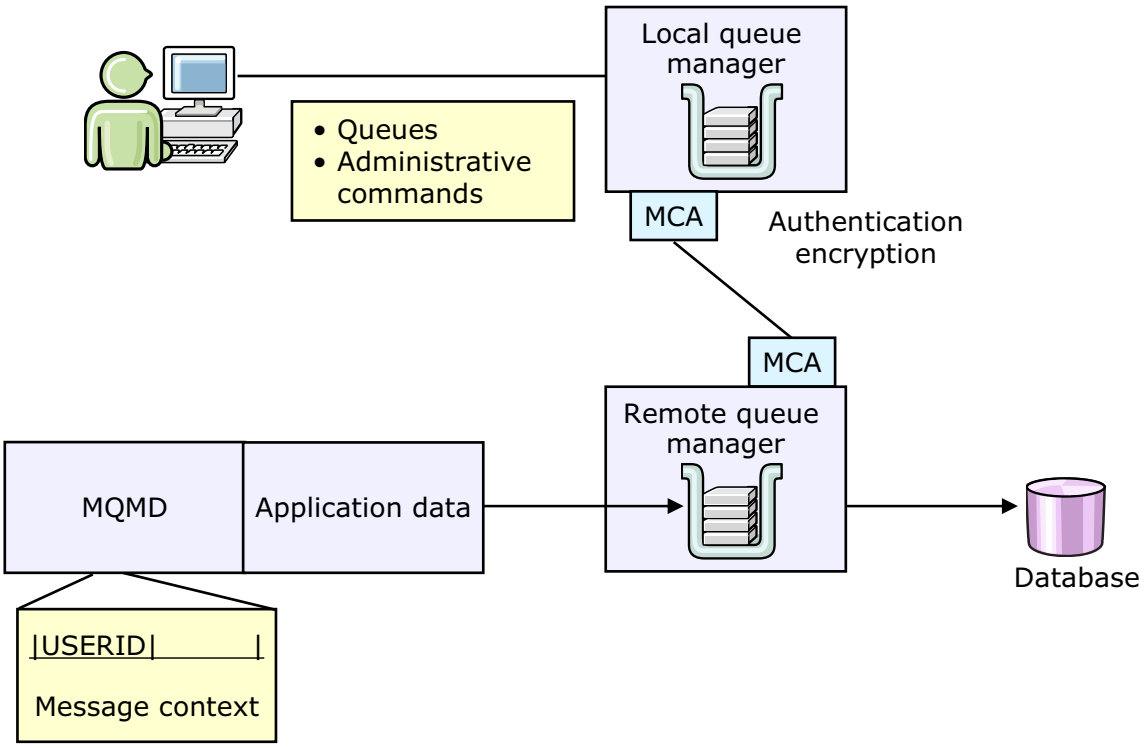


Figure 1-20. Security WM100 / VM1001.0

Notes:

Security is an important aspect of system management. You will look at the security features of IBM WebSphere MQ in more detail later, but here is a brief overview of some of the security features that IBM WebSphere MQ provides:

- A queue manager can check whether a user is authorized to enter commands that are used to manage the queue manager.
- A queue manager can check whether a user or an application is authorized to access a WebSphere MQ resource, such as a queue, for a specified operation.
- An MCA can authenticate a partner MCA before allowing messages to flow.
- A message can be encrypted before it is sent by an MCA to its partner MCA. At the receiving end, the message can be decrypted.
- A message descriptor normally contains a user ID field and other information about the originator of the message. This information is called the *message context*. Information in the message context can be used to authenticate a message, and to check whether

the sender of the message has the authority to access a WebSphere MQ resource on the system on which the message is received.

The user ID might also be used by the application to check whether the sender of the message has the authority to access a non-WebSphere MQ resource, such as a database; but whether this is possible depends on the security features provided by the respective resource manager.

Checkpoint

Exercise — Unit 1 checkpoint

1. IBM WebSphere MQ uses an interface for programs to access resources called:

   a. The program-to-program API
   b. The Message Queue Interface
   c. The synchronous model
   d. Triggering

2. **T**/**F**: IBM WebSphere MQ only supports messaging and queuing in an asynchronous environment.

3. A message consists of:

   a. Application data
   b. A WebSphere MQ trailer
   c. A security header
   d. A message descriptor
   e. All the above

4. All WebSphere MQ messages will have a header. It is the:

   a. MQXQH (transmission header)
   b. MQDLK (dead letter header)
   c. MQMD (message descriptor)
   d. MQTH (trigger header)

5. **T**/**F**: In WebSphere MQ triggering, the queue manager starts the triggered program.

Unit summary

Having completed this unit, you should be able to:

- Explain the positioning of messaging and queuing in today's business environment
- Provide a high-level view of WebSphere MQ functions
- Show the breadth of coverage of WebSphere MQ products

Figure 1-21. Unit summary WM100 / VM1001.0

Notes:

This unit has discussed the functions and value that IBM WebSphere MQ provides, at an overview level. Five major benefits of IBM WebSphere MQ were highlighted:

- There is a common application programming interface, the MQI, that is consistent across all the supported platforms.
- IBM WebSphere MQ can transfer data with assured delivery; messages do not get lost, even in the event of a system failure. Just as important, there is no duplicate delivery.
- The communicating applications do not have to be active at the same time. For example, a sending application can still be putting messages on a queue even though the receiving application is not active.
- Message-driven processing is a style of application design. An application is divided into discrete functional modules that can communicate with each other by means of messages. In this way, the modules can execute on different systems, be scheduled at different times; or they can act in parallel.

- Application development is made faster by shielding the developer from the complexities of the network.

Now it is time to go a layer deeper and gain a better understanding of the details of some of these functions.

Unit 2. Programming with WebSphere MQ

What this unit is about

This unit describes the functions provided by the Message Queue Interface (MQI) and some of the more common details of its use.

What you should be able to do

After completing this unit, you should be able to:

- Explain several calls provided by the message queue interface (MQI)
- Explain the use of several calls at a high level
- Provide a detailed description of the MQI
- Describe the Java interface and Java Message Services in WebSphere MQ

How you will check your progress

Accountability:

- Instructor questions
- Checkpoint questions

References

www.ibm.com/software/integration/wmq/library

IBM WebSphere MQ Library

SC34-6595 *WebSphere MQ Application Programming Guide*

SC34-6596 *WebSphere MQ Application Programming Reference*

SC34-6591 *WebSphere MQ Using Java*

Unit objectives

After completing this unit, you should be able to:
- Explain each message queue interface (MQI) call
- Provide a high-level understanding of the use of each call
- Describe the details of the MQI
- Note some WebSphere MQ functions
- Describe the Java interface and Java Message Service (JMS)

Figure 2-1. Unit objectives WM100 / VM1001.0

Notes:

This unit introduces each of the message queueing interface (MQI) calls and provides a high-level understanding of their use. On completion, you should be able to list each of the calls and describe their functions. The unit also introduces the Java and Java Message Service application programming interfaces.

2.1. Programming with WebSphere MQ — the MQ interface

The message queue interface (MQI) philosophy

- Simple call interface
- Limited number of calls
- Rich in function
- Sensible default and initial values
- Supplied “include” files and “copy” files for the definitions of structures and constants

Figure 2-2. The MQI philosophy WM100 / VM1001.0

Notes:

The design principles behind the MQI are reflected in the interface. It is a simple call interface with a limited number of calls and a rich set of options for each call. Sensible default and initial values ensure that it is easy and quick to get applications up and running.

The MQI uses a number of structures and constants. IBM WebSphere MQ supplies “include” files and “copy” files containing the definitions of these structures and their fields, and also the definitions of meaningful symbolic names to represent the constants within program logic.

Message Queue Interface (MQI)

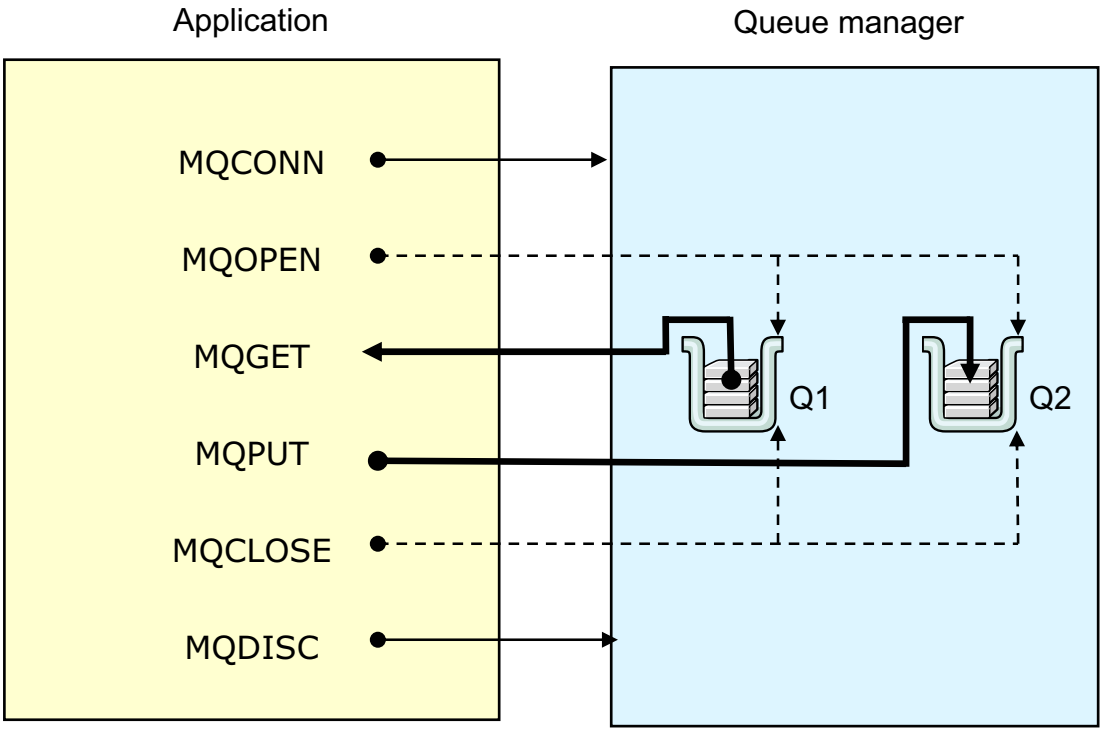


Figure 2-3. Message Queue Interface (MQI) WM100 / VM1001.0

Notes:

The MQI is a simple call interface. The most common calls are:

MQCONN	Connects to a queue manager
MQOPEN	Establishes access to a WebSphere MQ object
MQPUT	Places a message on a queue or publish to a new topic
MQGET	Retrieves a message from a queue
MQCLOSE	Releases access to a WebSphere MQ object
MQDISC	Disconnects from a queue manager

Some less common calls are:

MQPUT1	Opens a queue manager, places a message on the queue, and closes the queue manager
MQINQ	Queries the attributes of a WebSphere MQ object
MQSET	Changes the attributes of a queue

MQCONNX	Connects to a queue manager using the specified options
MQBEGIN	Begins a unit of work coordinated by the queue manager
MQCMIT	Commits all message PUT and GET operations made since the last syncpoint
MQBACK	Backs out all message PUT and GET operations made since the last syncpoint
MQSUB	Register subscriptions to a previously published topic
MQSUBR	Request a retained publication

Message queue interface (MQI) call notation

- WebSphere MQ Application Programming Reference

```
MQPUT1 (Hconn, ObjDesc, MsgDesc, PutMsgOpts,
BufferLength, Buffer, CompCode, Reason)
```

- C equivalent

```
MQPUT1 (Hconn, &ObjDesc, &MsgDesc,
&PutMsgOpts, BufferLength, Buffer, &CompCode,
&Reason);
```

- COBOL equivalent

```
CALL "MQPUT1" USING HCONN, OBJDESC, MSGDESC,
PUTMSGOPTS, BUFFERLENGTH, BUFFER, COMPCODE,
REASON.
```

Figure 2-4. MQI call notation WM100 / VM1001.0

Notes:

The *WebSphere MQ Application Programming Reference* defines the MQI calls. The figure shows an example of the notation used in that publication and others.

MQPUT1 is an example of an MQI *call*. The items in parentheses following it, namely *Hconn*, *ObjDesc*, and so forth, are referred to as its *parameters*.

Examples of an MQPUT1 call in both C and COBOL are also shown on the visual. Other languages supported by IBM WebSphere MQ include System/390 assembler, C++, PL/I, RPG, Java, Visual Basic, and .NET.

MQCONN call

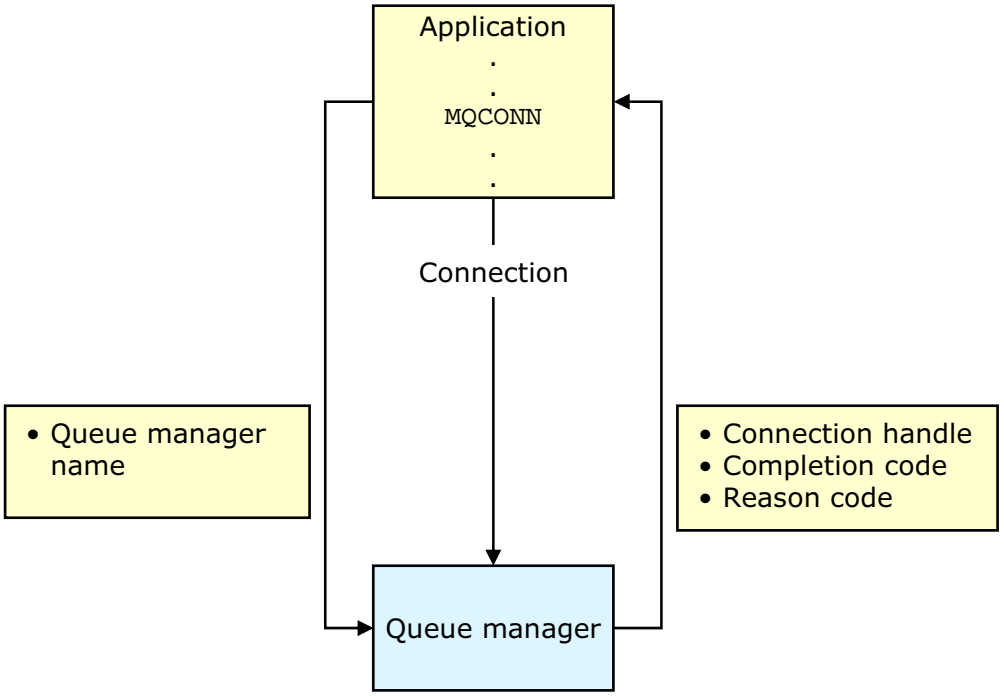


Figure 2-5. MQCONN call WM100 / VM1001.0

Notes:

Call syntax:

```
MQCONN (QMgrName, Hconn, CompCode, Reason)
```

Before issuing any other MQI calls, an application must first connect to a queue manager by issuing an MQCONN call. The queue manager to which an application connects is known as the *local* queue manager. In general, you connect to a specific queue manager or to a default queue manager. An application normally needs authority to connect to a queue manager.

The MQCONN call has one input parameter and three output parameters. They are:

Queue manager name (input)
The name of the queue manager to which the application wishes to connect.

Connection handle (output)
The connection handle represents the connection to the queue manager

and must be specified as a parameter on all subsequent MQI calls to that queue manager.

Completion code (output)
Every MQI call returns a completion code to enable the application to determine whether the call completed successfully, completed partially, or failed.

Reason code (output)
Every MQI call returns a reason code to provide more information to the application when a call completes partially or fails.

The application uses the **MQDISC** call to disconnect from a queue manager. When the MQDISC call is completed, the connection handle ceases to be valid.

Call syntax:

```
MQDISC (Hconn, CompCode, Reason)
```

MQCONNX call

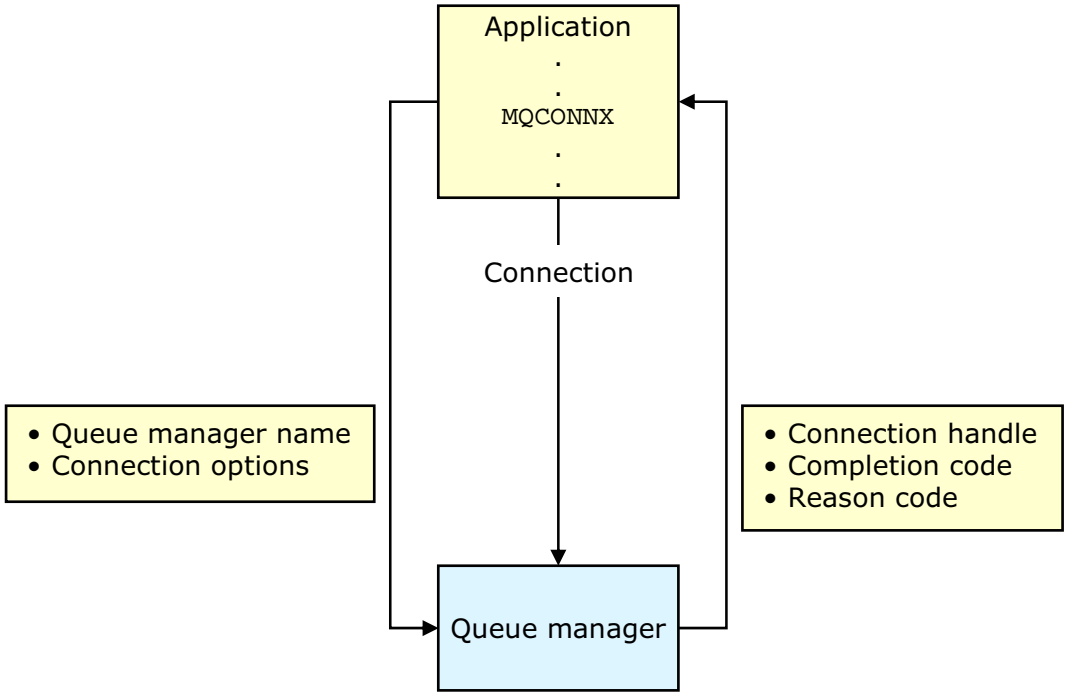


Figure 2-6. MQCONNX call WM100 / VM1001.0

Notes:

Call syntax:

```
MQCONNX (QMgrName, ConnectOpts, Hconn, CompCode, Reason)
```

The MQCONNX is another means of connecting to the queue manager available on some platforms. MQCONNX is similar to MQCONN, but it allows for a “connection options” parameter to be passed. These connections options allow you to control the binding options for the connection.

Those parameters of the MQCONNX call that have not been described previously are:

Connection options (input/output)
The binding options to be used when opening the connection

When an application connects to a queue manager using the MQCONN call, the queue manager code that is executed to service each subsequent MQI call runs as a separate

unit of execution from that of the application. When an application connects to a queue manager using the MQCONNX call, it can specify different binding options. One such option is for “fast path binding.” Fast path binding causes the queue manager code that executes each subsequent MQI call to run within the same unit of execution as the application. The advantage of such an arrangement is that fewer system resources are required to run the application. The disadvantage is that the integrity of the queue manager can be compromised, as there is no protection from the application overwriting the storage used by the queue manager. In order to use fast path binding, the application must be run as a *trusted application*.

The MQCONNX call can be used by clients to connect to a specific server and override any settings that would otherwise be in effect for that connection.

The MQCONNX call allows you to create a shared (thread-independent) connection that can be used by all threads in a process.

MQCONNX can be used to allow an application to serialize its calls to a queue manager.

Remote queue manager

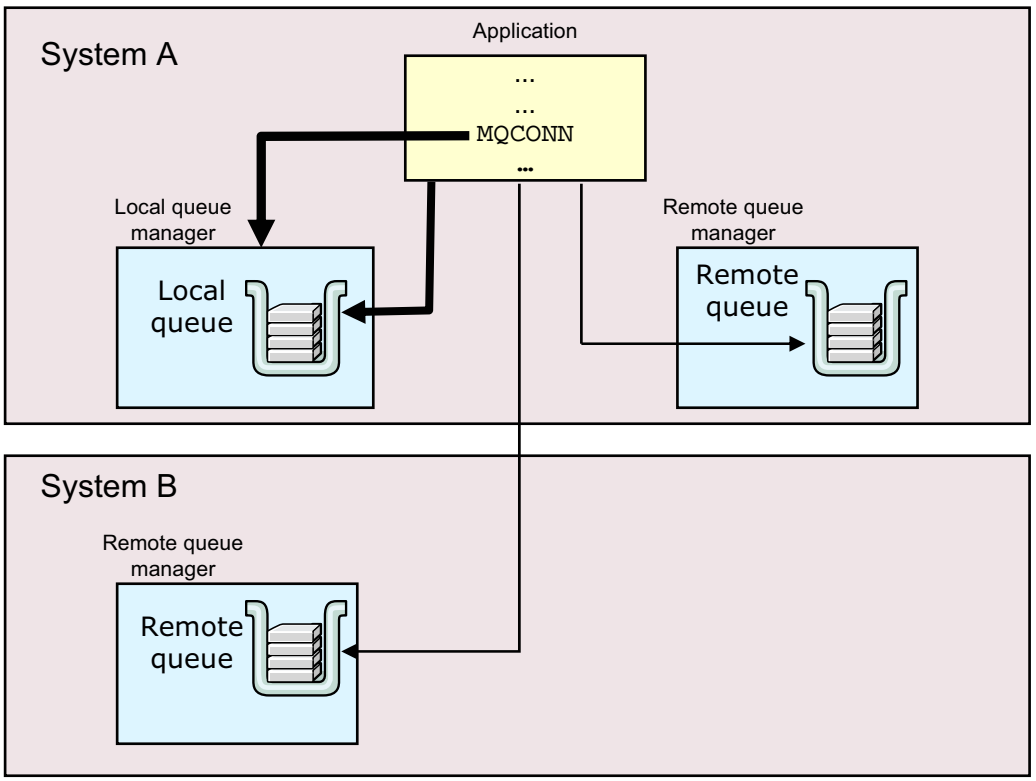


Figure 2-7. Remote queue manager WM100 / VM1001.0

Notes:

The figure shows two systems. System A is running two queue managers. The application connects to one of the queue managers, which then becomes its *local* queue manager. Access to a queue owned by the local queue manager is direct and synchronous.

The other queue manager on System A is termed a *remote* queue manager even though it is on the same system as the queue manager to which the application is connected. Any message originating from the application that is destined for a queue owned by the remote queue manager, is delivered asynchronously by message channel agents (MCAs).

System B is also running a remote queue manager. Likewise, messages destined for queues owned by this queue manager are delivered asynchronously by MCAs.

MQOPEN call

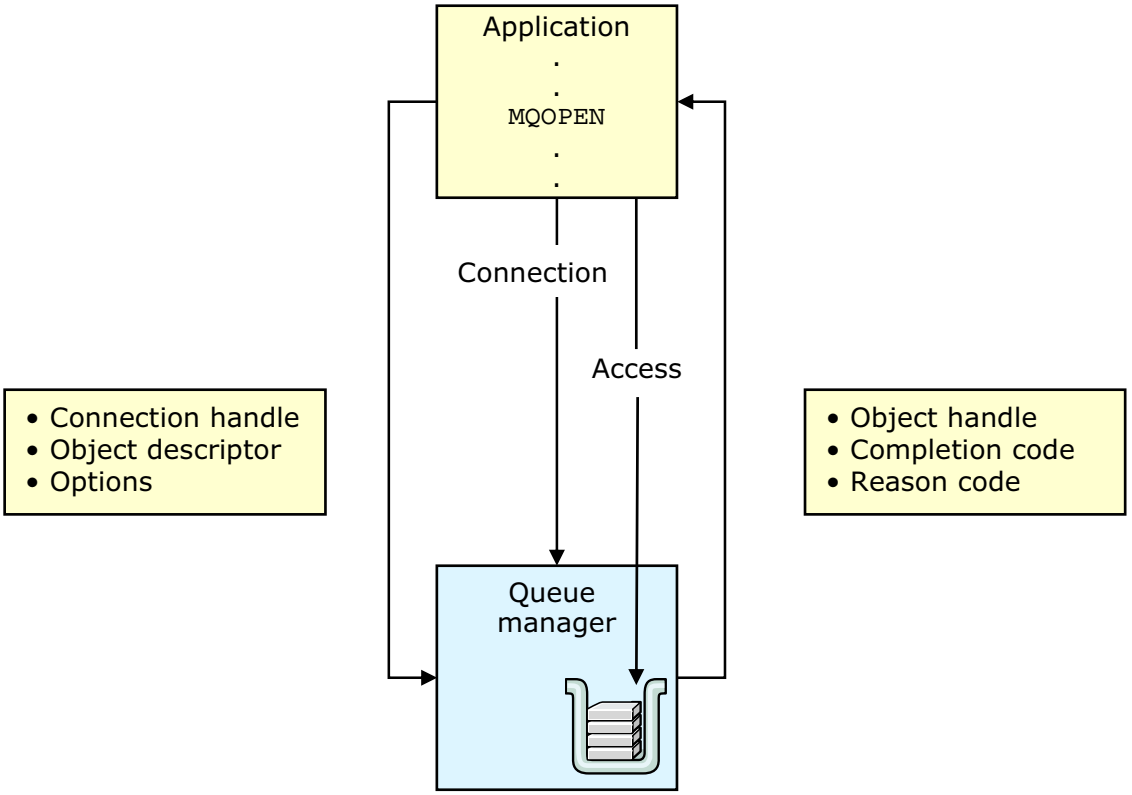


Figure 2-8. MQOPEN call WM100 / VM1001.0

Notes:

Call syntax:

```
MQOPEN (Hconn, ObjDesc, Options, Hobj, CompCode, Reason)
```

Before an application can put messages on a queue or get messages from a queue, it must first open the queue by issuing an MQOPEN call.

The figure shows a local queue being opened using either a local queue, an alias queue, or model queue object.

MQOPEN may be used to open other types of IBM WebSphere MQ objects such as a process definition, a namelist, or the queue manager object. You will see later in this unit why an application may need to open an object other than a queue.

Those parameters of the MQOPEN call that have not been described previously are:

Object descriptor (input/output)

This is one of the structures defined in the MQI. It identifies the object being

opened. It contains a number of fields that specify, among other things, the name of the object being opened and the type of object.

Options (input**)**

The application uses this parameter to specify which operations it wishes to perform on the object being opened, or to control the action of MQOPEN. Examples include opening a queue for putting messages on it, opening a queue for browsing messages only, and opening a queue to inquire on the values of some or all of its attributes. An application may specify more than one option by adding the values together or by combining them using the bitwise OR operation.

An application will normally need authority to open an object for each of the requested operations. These authorities are checked at the time the object is opened.

The application uses the **MQCLOSE** call to relinquish access to a queue, or any other type of IBM WebSphere MQ object. After the call, the object handle ceases to be valid.

Call syntax:

```
MQCLOSE (Hconn, Hobj, Options, CompCode, Reason)
```

Queue definition and independence

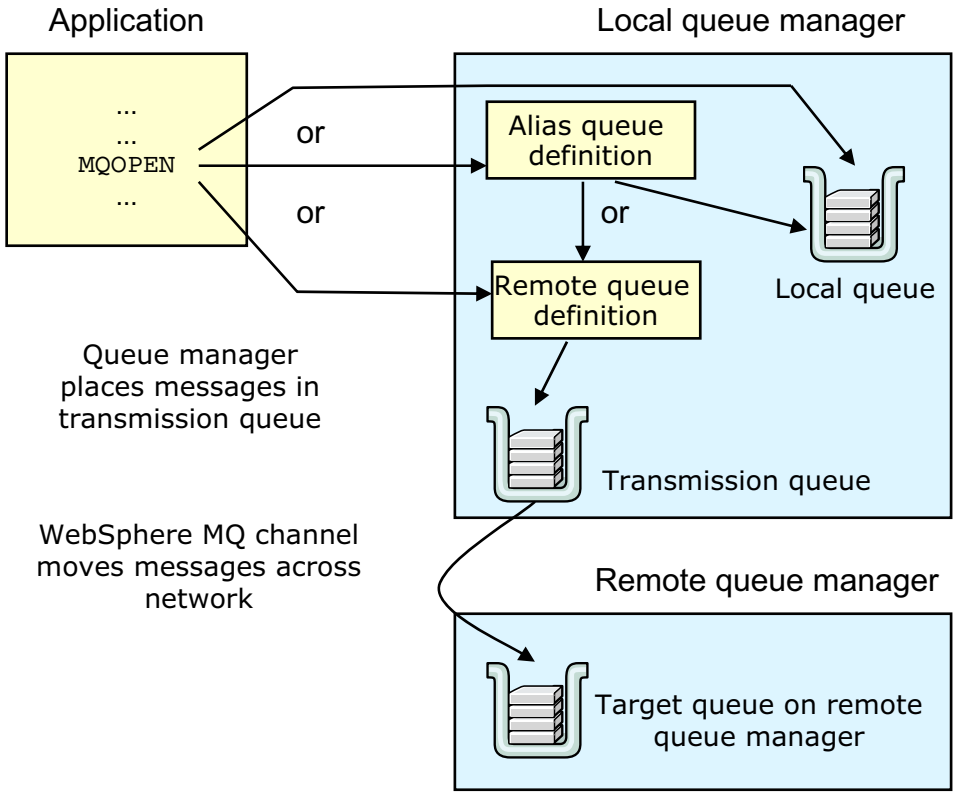


Figure 2-9. Queue definition and independence WM100 / VM1001.0

Notes:

There are four types of queue that an application can open:

- Local queues
- Remote queues (local definitions of remote queues; cannot be opened for input)
- Alias queues
- Model queues (used to create queues dynamically)

With the exception of model queues, the syntax of the MQOPEN call is the same, irrespective of the type of queue being opened. For output queues, this allows the application architect or administrator to move a queue from one queue manager to another without having to rewrite the program.

The queue manager is responsible for resolving the name of the object being opened.

Model queue definition

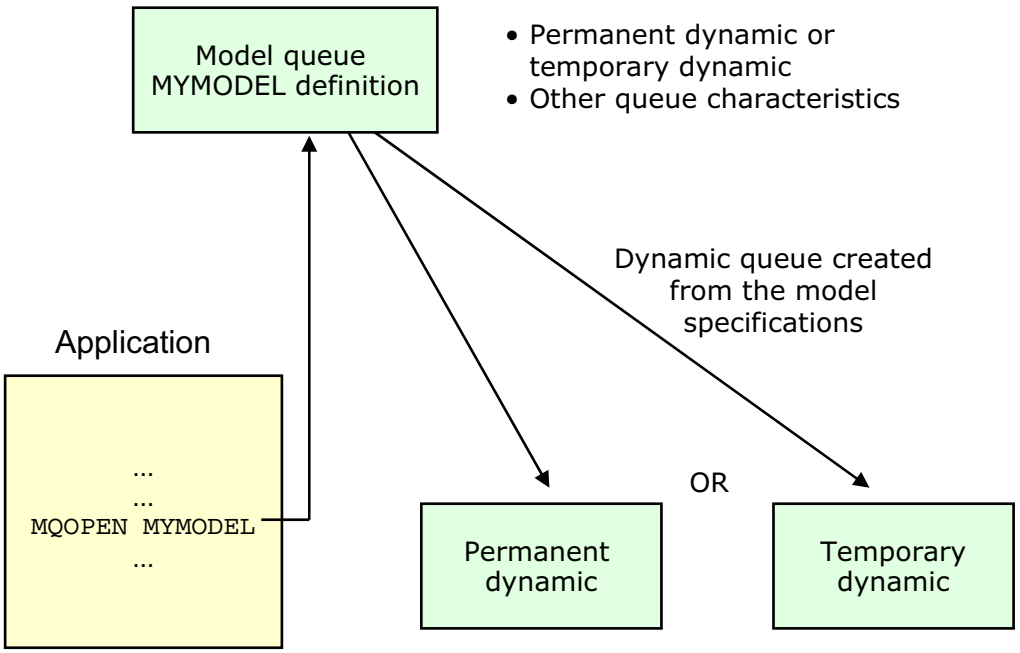


Figure 2-10. Model queue definition WM100 / VM1001.0

Notes:

A model queue is a template that is used to define other queues. That definition is a template which is used to define other queues. When the name of a model queue is specified in the object descriptor for an MQOPEN, a queue with the attributes of the model is dynamically created. The model itself has no other purpose. If the characteristics of the new queue were to be displayed, the queue would appear to be a local queue.

Temporary dynamic queues last only until the execution of the program that created it ends (normally or abnormally) or until the creating program closes it. There is no way to retain a temporary dynamic queue beyond that point. Temporary dynamic queues may not hold persistent messages.

Permanent dynamic queues are created in exactly the same way, but they are not automatically deleted. They must be explicitly deleted (by specifying a “close” option on delete, or by the administrator issuing a delete command on the queue). Once created, there is nothing specific that IBM WebSphere MQ does to keep track of dynamically created, permanent dynamic queues.

Which type of dynamic queue is chosen is a matter of application design.

MQPUT call

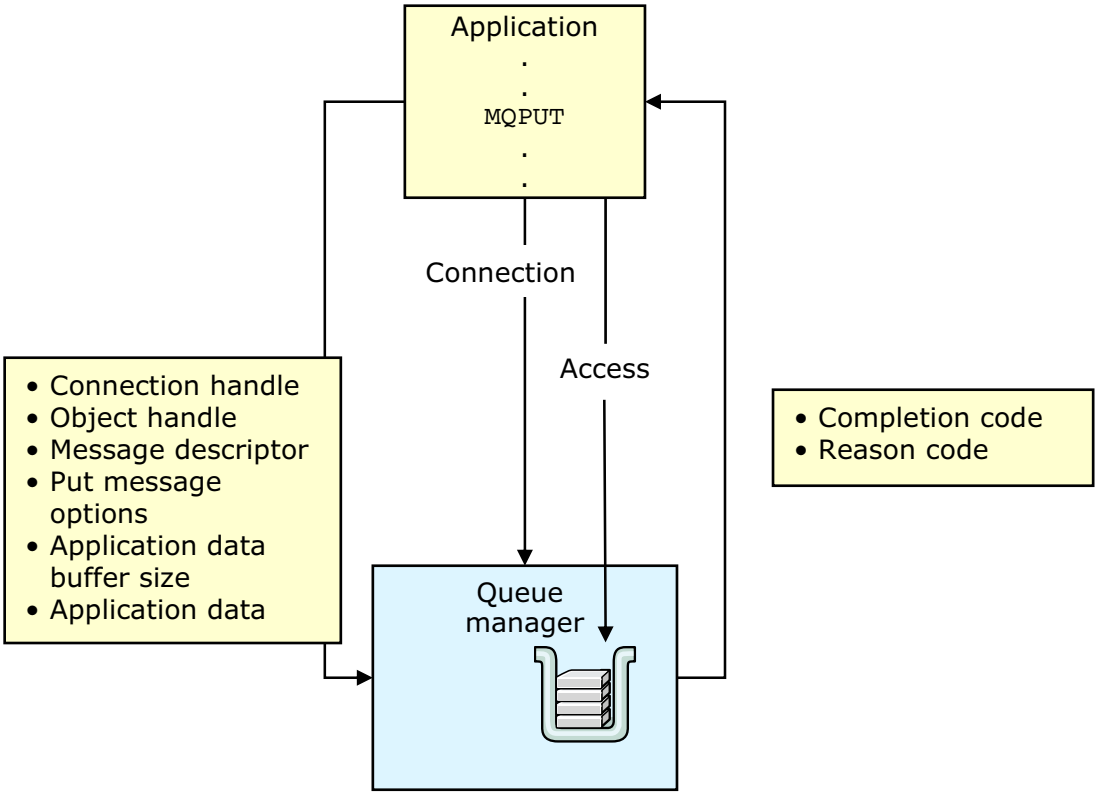


Figure 2-11. MQPUT call WM100 / VM1001.0

Notes:

Call syntax:

```
MQPUT (Hconn, Hobj, MsgDesc, PutMsgOpts, BufferLength,
Buffer, CompCode, Reason)
```

Once an application has opened a queue for output, it may call MQPUT to put a message on the queue. The object handle returned by the MQOPEN call is used to identify the queue to be used by the MQPUT call.

Although the figure depicts an application putting a message on a local queue, MQPUT can also be used to put a message to a remote queue; that is, a queue owned by a queue manager to which the application is not connected. To do this, the application must first call MQOPEN to open a local definition of the remote queue, or an alias queue resolving to a local definition of the remote queue, and then use the object handle returned by the MQOPEN call.

Those parameters of the MQPUT call that have not been described previously are:

MQ message descriptor (input/output)

The message descriptor is another structure defined in the MQI. It provides information about the message accompanying the application data being put on the queue, or returned to the application along with the application data when retrieved from a queue. Some of the fields are set by the application that puts the message on the queue, while others are set by the queue manager on behalf of the application.

The message descriptor is discussed in more detail in the next unit.

Put message options (input/output)

This is another structure defined in the MQI. It contains a number of fields that control the action of MQPUT.

Buffer length (input/output)

The size of the application data in the buffer.

Buffer (input)

The application data in the message.

MQGET call

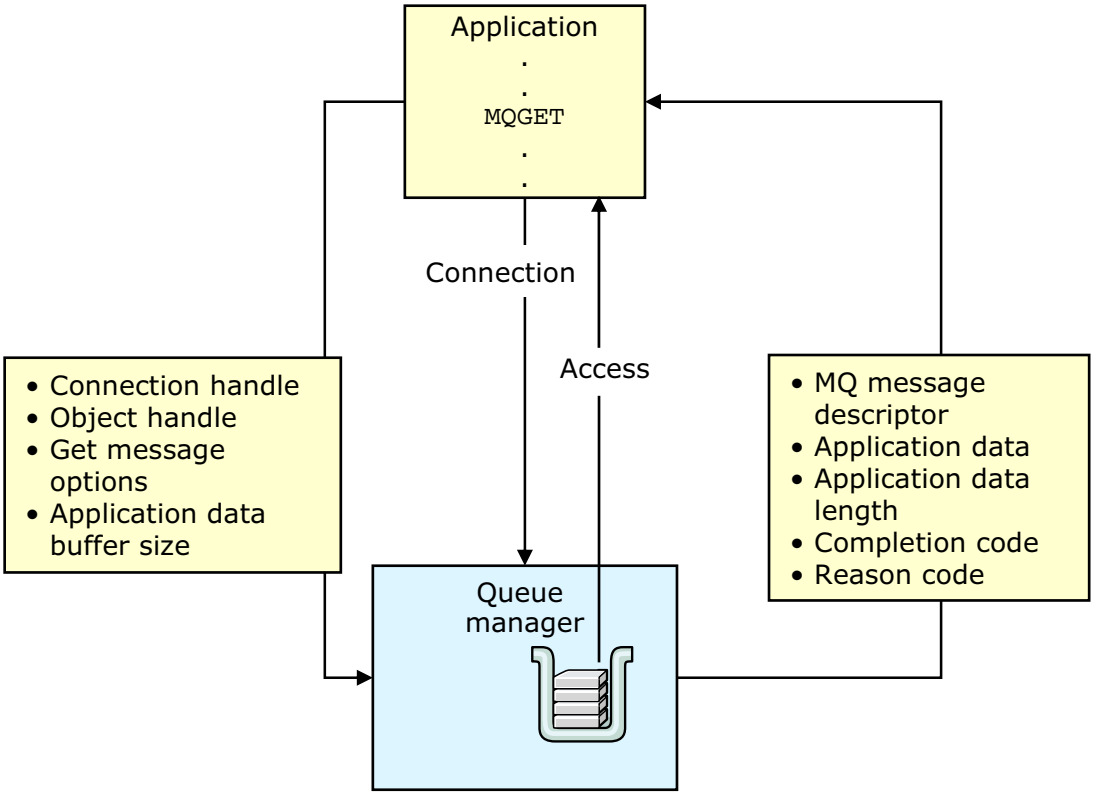


Figure 2-12. MQGET call WM100 / VM1001.0

Notes:

Call syntax:

```
MQGET (Hconn, Hobj, MsgDesc, GetMsgOpts, BufferLength,
Buffer, DataLength, CompCode, Reason)
```

An application retrieves a message from a queue by issuing an MQGET call. An application can only issue an MQGET call after it has successfully opened a queue for input (or for browse). The object handle returned by the MQOPEN call is used to identify the queue on the MQGET call.

An application may only get messages from a queue that is owned by the queue manager to which it is connected.

Those parameters of the MQGET call that have not been described previously are:

Get Message Options (input/output)
This is another structure defined in the MQI. It contains a number of fields

that control the action of MQGET. This structure allows great control over message retrieval. You look at some of those control fields next.

Data length (output)

The length of data returned from the call. If **data length** is less than **buffer length**, the message is truncated.

MQGET get message options

- Wait
- Set signal
- Browse from start of queue
- Browse from current position in queue
- Get message under browse cursor
- Within syncpoint control
- Outside of syncpoint control
- Accept truncated message
- Convert application data

Figure 2-13. MQGET get message options WM100 / VM1001.0

Notes:

The MQGET get message options structure contains a number of fields that control the action of MQGET. One of the fields, called *options*, allows an application to specify one or more of the options listed in the figure and others that are not listed here. An application can specify more than one option by adding the values together or by combining them using the bitwise OR operation.

The *Options* field in the get message options structure enables an application to specify one or more of the following.

Wait If the queue is empty, control is not returned to the application. Instead, the application waits until a suitable message arrives.

Set signal If the queue is empty, control is returned to the application. When a suitable message arrives on the queue, the queue manager sends a signal to the application program. In the meantime, the application can continue with other processing and, at some point, either test whether the signal has been set by the queue manager, or wait until the signal is set. (Note: set signal is not available on all platforms.)

Browse from start of queue

An application may simply read the contents of a message on a queue without actually removing it from the queue. This is known as *browsing* a message. This option allows an application to browse the first suitable message on a queue.

Browse from current position in queue

Instead of starting from the beginning of a queue, an application may browse the next suitable message from its current position in the queue.

Get message under browse cursor

Having browsed a message, an application may then get the message from the queue, thus removing it from the queue.

Within syncpoint control

An application may get a message under syncpoint control, in which case the message is not actually removed from the queue until the unit of work is committed. This option is discussed in a later unit.

Outside of syncpoint control

An application may get a message outside of syncpoint control. In this case, the message is removed from the queue immediately and cannot be made available again by backing out the unit of work. This option is discussed in Unit 6.

Accept truncated message

If a message is too large to fit in the buffer area provided by an application, the application may elect to accept as much of the message as the buffer can hold.

Often an application knows that a message over a certain size must be an error. In this case, in order to be able to get the other messages in the queue, the application might accept the message in truncated form and put it on a queue that is being used to store messages that require further investigation: a dead letter queue, for example.

Convert application data

If an application gets a message that originated on a different system, the application data in the message may use character and numeric representations that are different from the ones being used by the application. The use of this option on an MQGET call causes the application data in a message to be converted into the character and numeric representations being used by the receiving application. This option is discussed further in a later unit.

MQPUT1 call

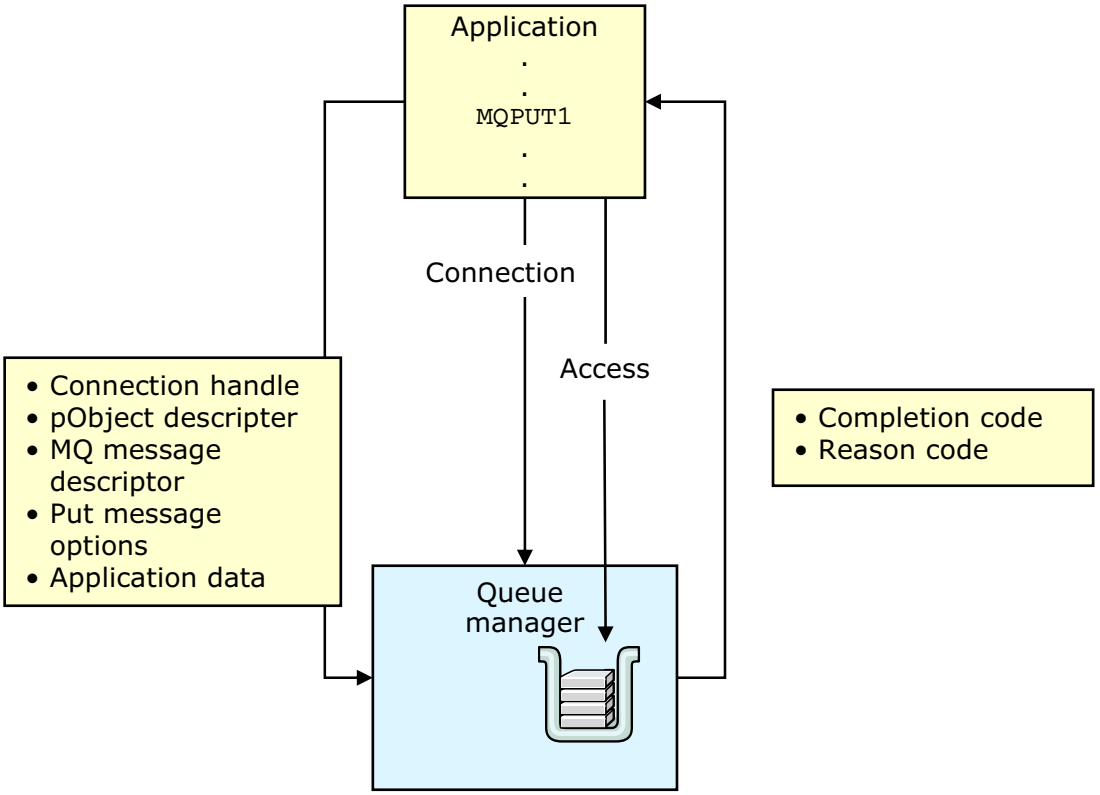


Figure 2-14. MQPUT1 call WM100 / VM1001.0

Notes:

Call syntax:

```
MQPUT1 (Hconn, ObjDesc, MsgDesc, PutMsgOpts, BufferLength,
Buffer, CompCode, Reason)
```

The MQPUT1 call is a combination of the MQOPEN, MQPUT, and MQCLOSE calls. MQPUT1 opens a queue, puts a single message on the queue, and then closes the queue, all in a single call. The call syntax is identical to that of MQPUT.

If there is only one message to be put on a queue, the use of MQPUT1 is more efficient than explicitly issuing an MQOPEN call, followed by an MQPUT call, and then an MQCLOSE call. If there is more than one message to be put on a queue, it is less efficient.

The MQPUT1 would be useful in a server application with a large number of client applications sending requests, each with its own reply-to queue. Because the frequency of replies to any one reply-to queue is unpredictable, it may be more appropriate to use MQPUT1 for each reply instead of holding a large number of queues open indefinitely.

MQINQ call

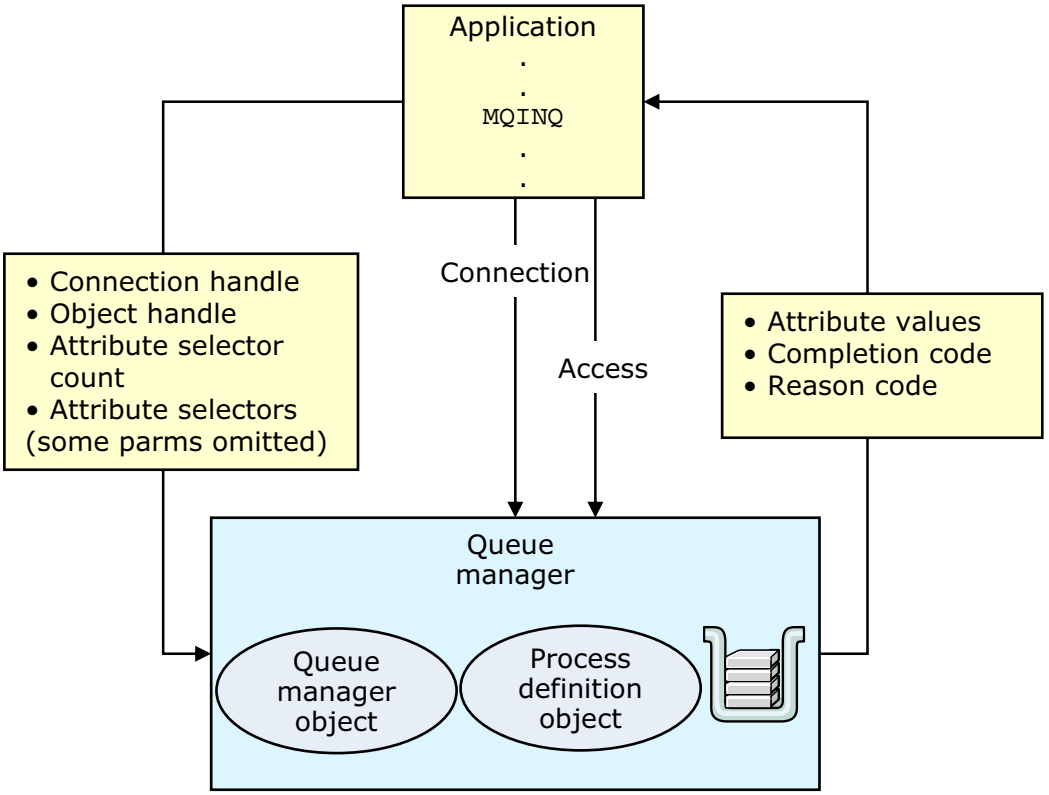


Figure 2-15. MQINQ call WM100 / VM1001.0

Notes:

Notes:

Call syntax:

```
MQINQ (Hconn, Hobj, SelectorCount, Selectors, IntAttrCount,
IntAttrs, CharAttrLength, CharAttrs, CompCode, Reason)
```

An MQINQ call can be used by an application to determine the values of specified attributes of a WebSphere MQ object, such as a queue, a ***topic***, a process, a namelist, or the queue manager object. For example, MQINQ can be used by an application to determine the current depth of a queue, or whether triggering is enabled for a queue.

Before an MQINQ call can be issued, the application must first open the object by using an MQOPEN call with the “open for inquiry” option. This is why MQOPEN might be used to open a process definition object, a namelist, or the queue manager object.

Those parameters of the MQINQ call which have not been described previously are:

Attribute selectors (input)

This is an array of the names of the attributes of the object whose values the application wishes to retrieve.

Attribute values (output)

These are two arrays containing the values of the specified attributes. One array contains the values that are integers; the other contains the values that are character data.

Note that some of the parameters listed in the call syntax have been omitted from the description. Selectors are passed to the call as integers or characters, depending on the individual selector; the additional parameters in the call syntax reflect those specific selector parameters.

MQSET call

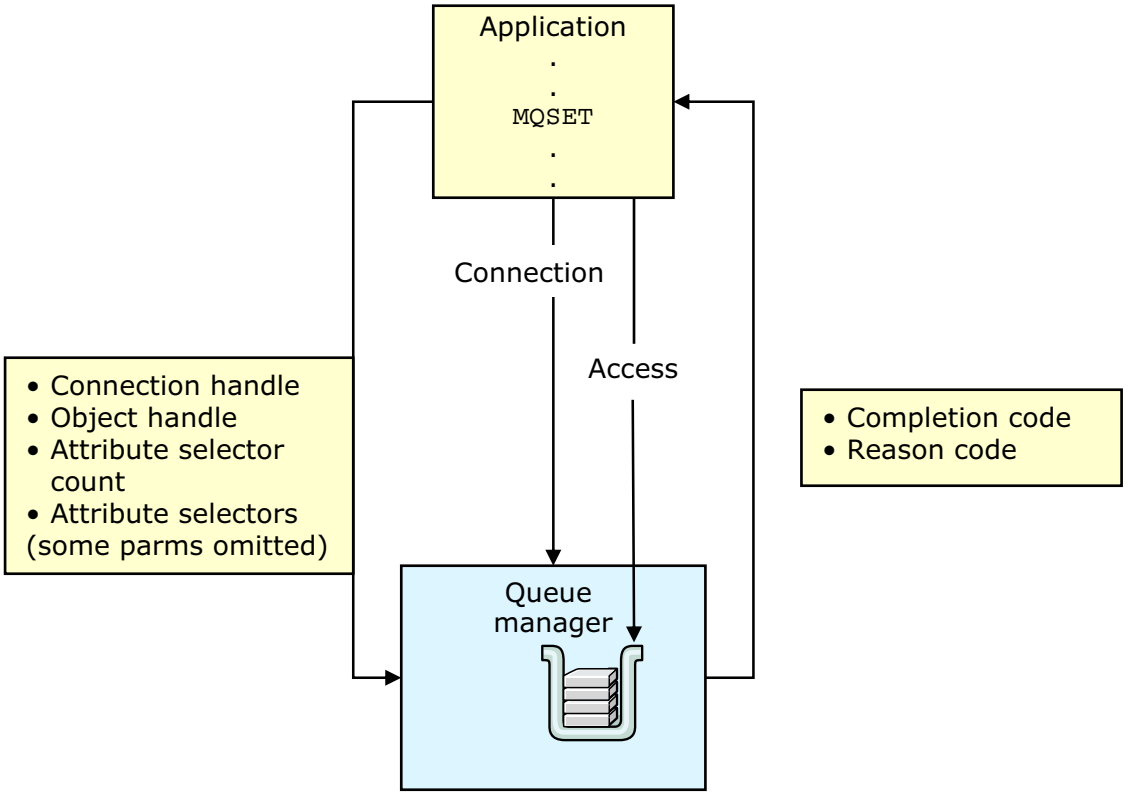


Figure 2-16. MQSET call WM100 / VM1001.0

Notes:

Call syntax:

```
MQSET (Hconn, Hobj, SelectorCount, Selectors, IntAttrCount,
IntAttrs, CharAttrLength, CharAttrs, CompCode, Reason)
```

MQSET can be used to modify a subset of attributes of a local queue. The attributes that can be set are:

- Whether put requests are allowed for the queue (*InhibitPut*)
- Whether get requests are allowed for the queue (*InhibitGet*)
- The attributes associated with triggering (*TriggerControl*, *TriggerType*, *TriggerMsgPriority*, *TriggerDepth*, and *TriggerData*)
- Whether distribution list messages can be put on the queue (*DistLists*) (not applicable for z/OS)

MQSET cannot be used to change any attributes of other objects.

Additional MQI calls

- MQBEGIN
- MQCMIT
- MQBACK
- MQSUB
- MQSUBR

Figure 2-17. Additional MQI calls WM100 / VM1001.0

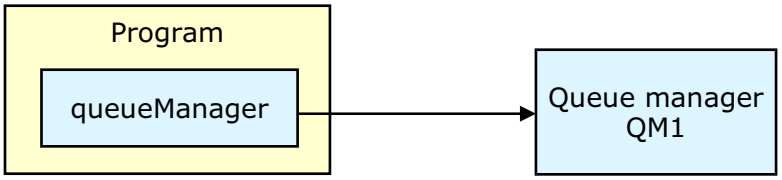
Notes:

MQBEGIN, MQCMIT and MQBACK are the remaining calls. They are discussed in detail in Unit 6, “Transactional Support”. As an overview:

- MQBEGIN — Is used to allow queue managers to coordinate updates that may include other resource managers.
- MQCMIT — If the queue manager is permitted to coordinate its own updates or those of other resource managers, this is the call to finalize those updates and commit the pending GET and PUT operations, as well as in-scope database updates.
- MQBACK — If the queue manager is permitted to coordinate its own updates or those of other resource managers, this call backs out any pending GET and PUT operations, as well as in-scope database updates, to a point of known consistency.

2.2. Programming with WebSphere MQ — Java and JMS

Connecting to a queue manager




```
MQQueueManager queueManager =
  new MQQueueManager(String queueManagerName);
```

Figure 2-18. Connecting to a queue manager WM100 / VM1001.0

Notes:

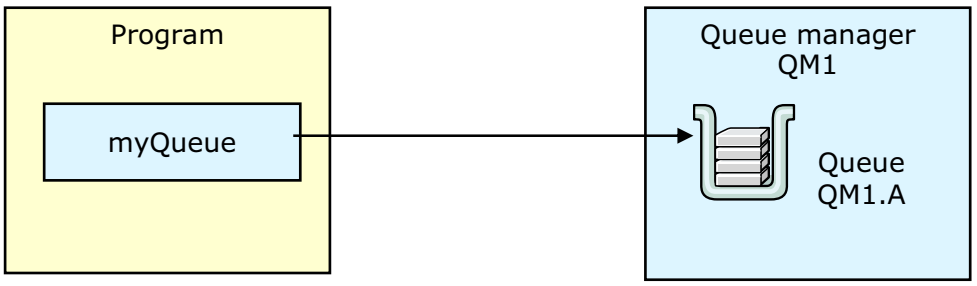
To connect to a queue manager, all you need to do is create a new instance of the MQQueueManager class:

```
MQQueueManager queueManager = new MQQueueManager("QM1");
```

If the queue manager name is left blank (null or ""), a connection is made to the default queue manager.

MQQueueManager is the Java constructor method equivalent of the MQCONN call. There are multiple formats of this call, depending on which parameters are passed.

Accessing queues




```
MQQueue myQueue =
  new MQQueueManager(String MQQueueManager,
                     String queueName,
                     int openOptions,
                     String queueManagerName,
                     String dynamicQueueName,
                     String alternateUserId);
```

Figure 2-19. Accessing queues WM100 / VM1001.0

Notes:

Queues are accessed using the accessQueue method of the MQQueueManager class. The accessQueue method returns a new object of class MQQueue. For example, to open queue QM1.A on queue manager QM1, use the following code:

```
MQQueue myQueue = MQQueueManager.AccessQueue("QM1.A",
   MQC.MQOO_OUTPUT,
   "QM1",
   "dynamicQName",
   "altUserID");
```

AccessQueue is the Java method equivalent of the MQOPEN call.

With WebSphere classes for Java, you can also create a queue object using the MQQueue constructor. The parameters are exactly the same as for the accessQueue method, with the addition of a queue manager parameter. For example:

```
MQQueue queue = new MQQueue(
   queueManager,
   "qName",
   MQC.MQOO_OUTPUT,
   "qMgrName",
   "dynamicQName",
   "altUserID");
```

The MQMessage object

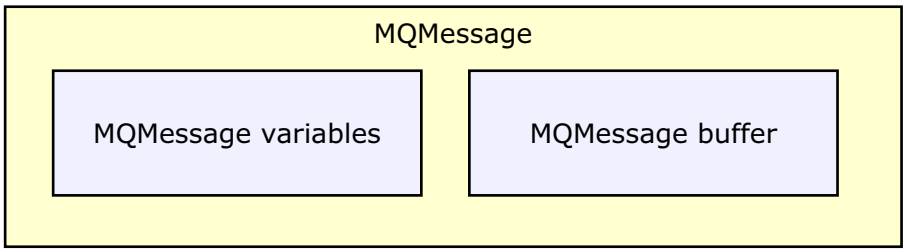


Figure 2-20. The MQMessage object WM100 / VM1001.0

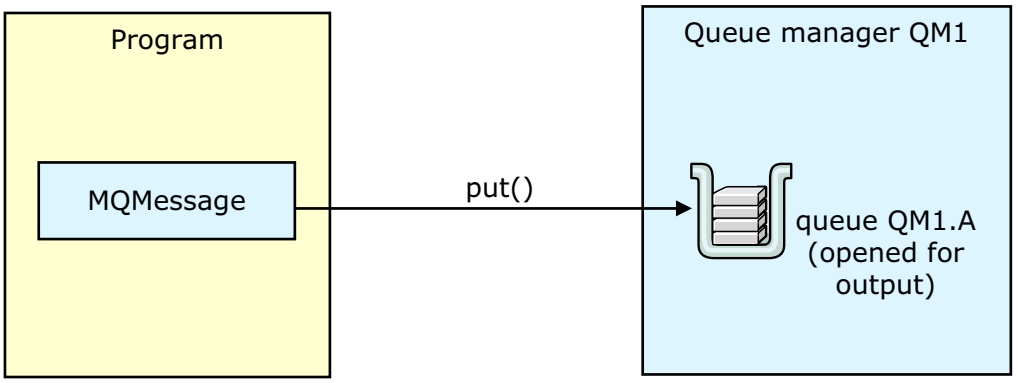
Notes:

The MQMessage object represents the message descriptor as variables and the user message data as the buffer.

A group of *ReadXXXX* methods is provided by the Java API for reading of various data types from a message, along with a similar group of *WriteXXXX* methods for writing data of various data types into a message. The format of numbers and strings used by these read and write methods can be controlled by the encoding and characterSet member variables.

The remaining member variables contain control information that accompanies the application message data when a message travels between sending and receiving applications. The application can set values into the member variable before putting a message to a queue and can read values after retrieving a message from a queue.

Putting messages on the queue




```
myQueue.put(myMessage,myPMO);
```

Figure 2-21. Putting messages on the queue WM100 / VM1001.0

Notes:

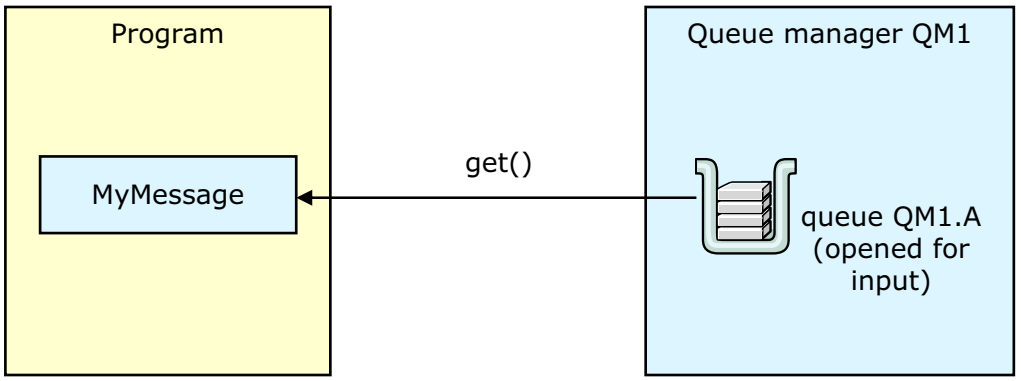
The put method puts a message onto the specified queue. It takes an MQMessage object as the first parameter and the put message options object as the second.

The put method uses these objects to construct the underlying MQI MQPUT call. There are multiple formats of this call, depending on which parameters are passed.

The variables of the MQMessage object are used to construct the message descriptor and are updated on successful completion of the put request.

To put a message on a queue, the queue must first have been opened with the “output” option.

Getting messages from the queue




```
myQueue.get(myMessage,myGetOptions,buffsz);
```

Figure 2-22. Getting messages from the queue WM100 / VM1001.0

Notes:

The get method retrieves a message from the associated queue. The call takes an MQMessage object as the first parameter, a list of get options, and other optional parameters. A number of fields in the MQMessage object are treated as input, in the processing performed by the underlying queue manager in response to the get request.

If the get throws an MQException and sets the associated completion code to “failed,” the fields (also known as member variables) and message buffer of the MQMessage object are unchanged, as no message has been returned. If the get request does not throw an MQException, or does but sets the completion code “warning,” the message fields and the message buffer of the associated MQMessage object are replaced with the message descriptor and message data from the incoming message.

The second parameter of the get method is the name of the associated getMessageOptions object. The fields of this object control the actions of the MQGET request. In particular, the options field controls the type of get request and another field controls the method of message identification.

Other fields within the getMessageOptions object are updated upon successful completion of the get request.

The third parameter of the get request is the size of the largest message that can be returned by this request. If a value is not supplied, the default action is to adjust the size of the buffer of the specified MQMessage object to accommodate the selected message. The use of this parameter can therefore be to restrict the size of the message buffer, and thereby stop a very large message from being returned. This parameter overrides the default size and any value set by the resizeBuffer request.

There are multiple formats of this call, depending on which parameters are passed.

To get a message from a queue, the queue must first have been opened with the “input” option.

Introducing Java Message Service (JMS)

- Java Message Service (JMS) is a set of interfaces and associated semantics that define how a JMS client accesses the facilities of an enterprise messaging product.
- JMS provides a common way for Java programs to create, send, receive, and read messages to and from an enterprise messaging system.
- The JMS API is a standard developed by Sun, IBM, and other enterprise solution vendors.
- WebSphere MQ is a JMS provider that implements JMS for a messaging product.

Figure 2-23. Introducing Java Message Service WM100 / VM1001.0

Notes:

JMS is a collection of Java services used to process messages. It is an open standard designed for communicating with enterprise messaging systems.

Since messaging is peer-to-peer, all users of JMS are referred to generically as clients. A JMS application is made up of a set of application-defined messages and a set of clients that exchange them.

JMS concept

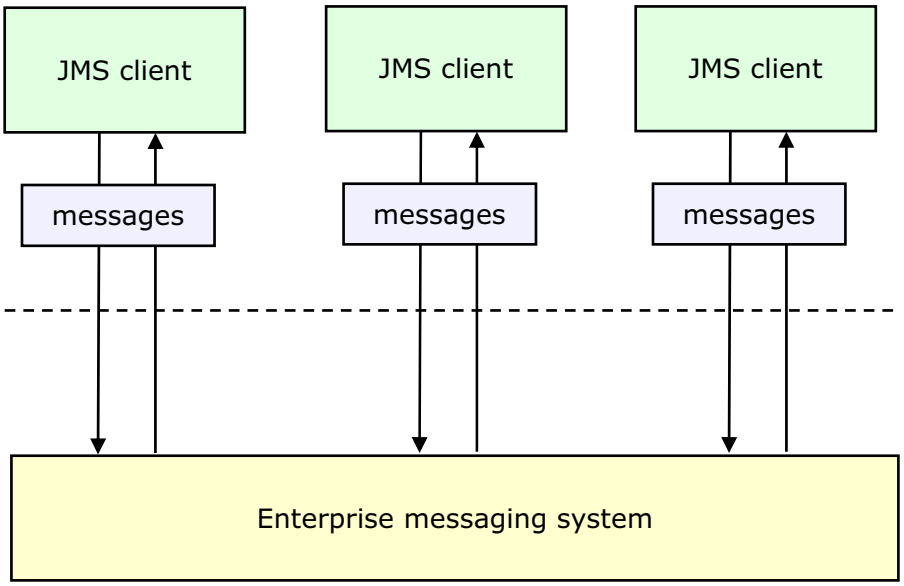


Figure 2-24. JMS concept WM100 / VM1001.0

Notes:

IBM WebSphere MQ classes for Java Message Service (MQ JMS) is a set of Java classes that implement Sun's Java Message Service (JMS) interfaces to enable JMS programs to access IBM WebSphere MQ systems. Both the point-to-point and publish/subscribe models of JMS are supported.

JMS APIs

- The generic JMS model is based around the following interfaces that are defined in Sun's javax.jms package:
  - Connection
    - Provides access to the underlying transport, and is used to create sessions
  - Session
    - Provides a context for producing and consuming messages, including the methods used to create MessageProducers and MessageConsumers
  - MessageProducer
    - Used to send messages
  - MessageConsumer
    - Used to receive messages

Figure 2-25. JMS APIs WM100 / VM1001.0

Notes:

In standard IBM WebSphere MQ terms:

- Connection

  Provides a scope for temporary queues, as well as a place to hold the parameters that control how to connect to IBM WebSphere MQ (for example, the name of the queue manager, and the name of the remote host if using the IBM WebSphere MQ Java client connectivity)

- Session

  Contains an HCONN (connection handle) and therefore defines a transactional scope

- MessageProducer and MessageConsumer

  Contain an HOBJ (queue handle) that specifies a particular queue for writing to or reading from

JMS components

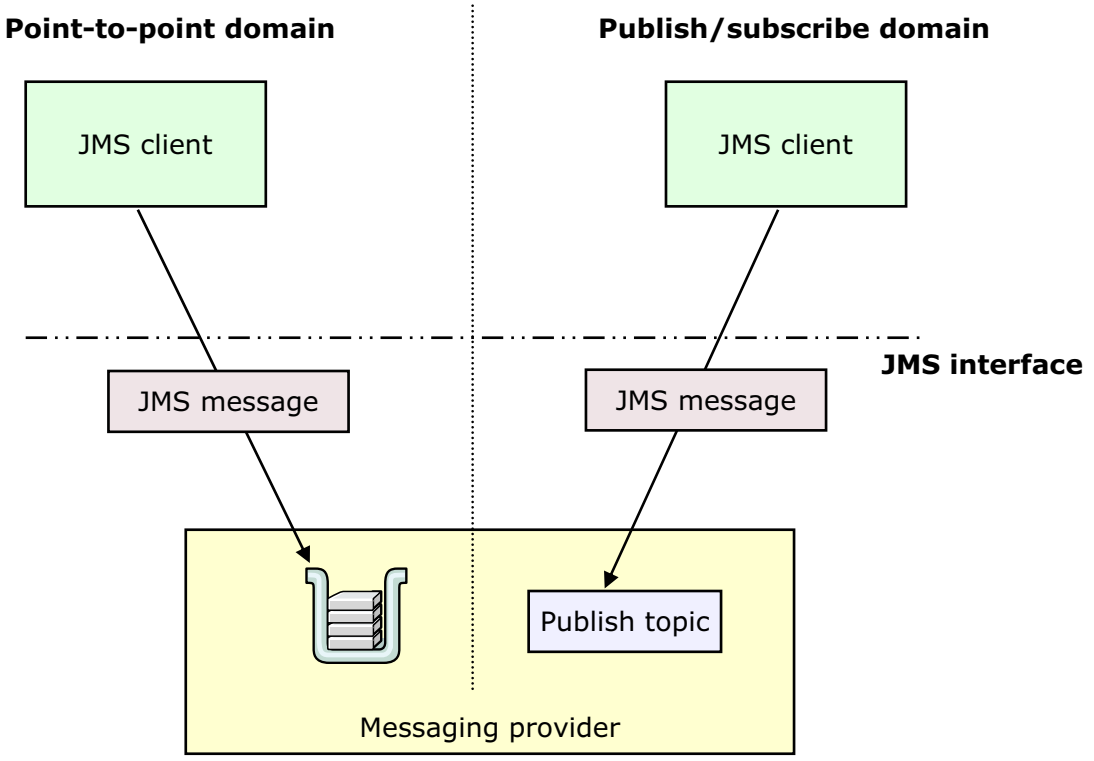


Figure 2-26. JMS components WM100 / VM1001.0

Notes:

JMS provider: A JMS provider is the entity that implements JMS for a messaging product.

JMS messages: JMS defines a set of message interfaces. Clients use the message implementations supplied by their JMS provider. A major goal of JMS is that clients have a consistent API for creating and working with messages, which is independent of the JMS provider.

JMS domains: Messaging products can be broadly classified as either *point-to-point* or *publish/subscribe* systems. Point-to-point products are built around the concept of message queues. Publish/subscribe clients address messages to some node in a content hierarchy. Publishers and subscribers are generally anonymous and may dynamically publish or subscribe to the content hierarchy.

Relationship to other Java APIs

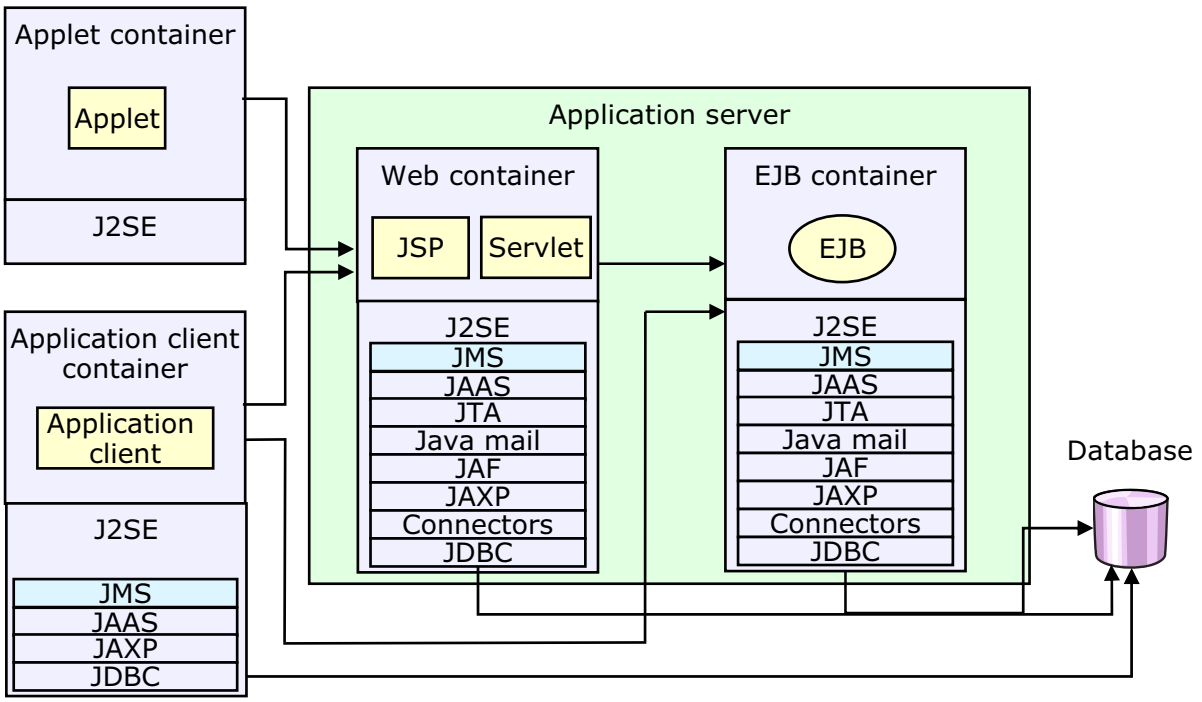


Figure 2-27. Relationship to other Java APIs WM100 / VM1001.0

Notes:

JMS is a mandatory part of J2EE (Java 2 Enterprise Edition), and must be available to application code executing in the Web container (using the JSP and servlet programming models), the EJB container (using the EJB programming model), and the client container (using the J2SE — Java 2 platform, Standard Edition — programming model). The implementation must allow code running in the three containers to intercommunicate using messaging.

WebSphere MQ APIs

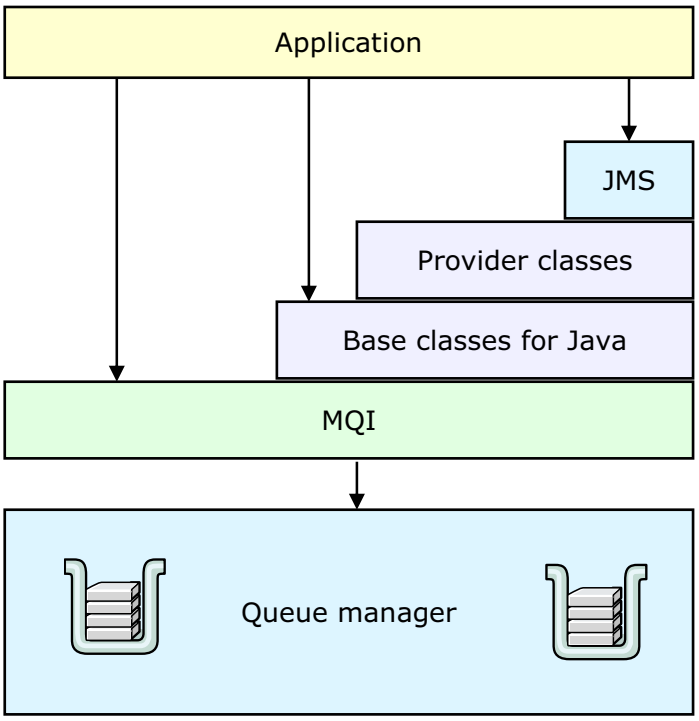


Figure 2-28. WebSphere MQ APIs WM100 / VM1001.0

Notes:

The JMS interface is presented to application developers, but the underlying services are supplied by the IBM WebSphere MQ base classes for Java, which in turn use the services of the standard MQI.

A Java application could also be written using the IBM WebSphere MQ base classes for Java. This approach allows for portability across other messaging product implementations.

JMS and IBM WebSphere MQ base classes are object-oriented implementations of IBM WebSphere MQ functionality. The standard MQI APIs are essentially a procedural language implementation. In some of the supported language bindings, there may be a choice of an object-oriented or a procedural interface. Java only has object-oriented style interfaces.

Checkpoint

Exercise — Unit 2 checkpoint

1. The additional parameter used on the MQCONNX call is:
   a. Queue manager name
   b. Connection handle
   c. Connection options
   d. Reason code
2. **T**/**F** MQSET lets you change all the attributes of queues only.
3. You must use MQOPEN before using (select all that apply):
   a. MQGET
   b. MQINQ
   c. MQPUT1
   d. MQCLOSE
   e. all the above

Unit summary

Having completed this unit, you should be able to:
- Explain each message queue interface (MQI) call
- Provide a high-level understanding of the use of each call
- Describe the details of the MQI
- Describe some WebSphere MQ functions
- Describe the Java interface and Java Message Service (JMS)

Figure 2-29. Unit summary WM100 / VM1001.0

Notes:

This unit offered a brief look at each of the verbs. In the next unit, some of the unique functions available using the MQI are reviewed. It is important to note that, although there are a limited number of MQI verbs, they are very powerful and offer a great deal of flexibility.

With appropriate design consideration, a program can be designed so that it can easily be moved among the various supported platforms with little or no change. This is a big plus in today's ever-changing environment.

Unit 3. Messages: additional information

What this unit is about

This unit describes some of the fields in the message descriptor and additional message options.

What you should be able to do

After completing this unit, you should be able to:

- Describe the fields in the message descriptor and how they relate to the WebSphere MQ environment
- Explain the various orders in which messages can be retrieved from a queue
- Describe message groups and segmentation
- Explain how message groups and segmentation are used
- Describe distribution lists

How you will check your progress

Accountability:

- Checkpoint questions
- Instructor questions

References

www.ibm.com/software/integration/wmq/library

IBM WebSphere MQ Library

SC34-6595 *WebSphere MQ Application Programming Guide*

SC34-6596 *WebSphere MQ Application Programming Reference*

Unit objectives

After completing this unit, you should be able to:

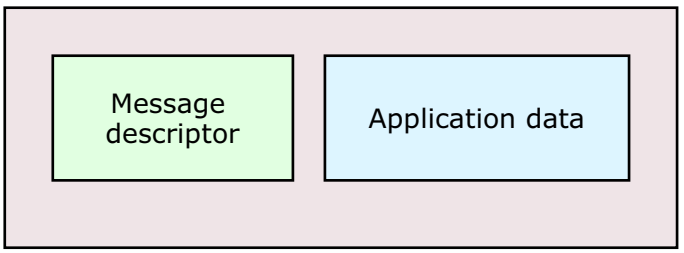
- Describe some of the fields in the message descriptor and how they relate to the MQ environment
- Explain the various orders in which messages can be retrieved from a queue
- Describe message groups and segmentation and why they are used
- Discuss distribution lists

Figure 3-1. Unit objectives WM100 / VM1001.0

Notes:

3.1. Additional information on messages

Some fields in the message descriptor



Message descriptor fields

- Persistence
- Priority
- MsgId
- CorrelId
- Report
- Feedback
- ReplyToQ
- ReplyToQMgr
- Expiry
- GroupId
- MsgSeqNumber
- Offset
- Format
- Encoding
- CodedCharSetId

Publication/subscription message descriptor fields

- SubName
- SubUserData
- SubCorrelId
- PubPriority
- PubAccountingToken
- PubAppIdentityData
  SubLevel
- ResObjString

Figure 3-2. Some fields in the message descriptor WM100 / VM1001.0

Notes:

Recall that a message is composed of two parts: the message descriptor and the application data. The message descriptor contains a number of fields, some of which are shown in this figure. The fields listed here are discussed in more detail below and in the following pages.

Every message contains a message identifier, which is determined by the value of the field *MsgId* in its message descriptor. When an application puts a message on a queue, either the application can supply a message identifier or it can ask the queue manager to generate a unique one.

The correlation identifier is normally used to provide an application with some means of matching a reply with the original message. In a reply message, therefore, the value of the *CorrelId* field is normally copied from the *MsgId* field of the original message.

The last three fields are associated with application data conversion, and the following is a brief explanation of their meanings:

Format The format of the application data in the message

Encoding The representation used for the binary integers, packed decimal integers, and floating point numbers in the application data

CodedCharSetId
The representation used for the character data in the application data

The publish/subscribe message descriptor fields enable the creation of published information at a topic level destined to all applications that subscribe to the topic.

Message persistence

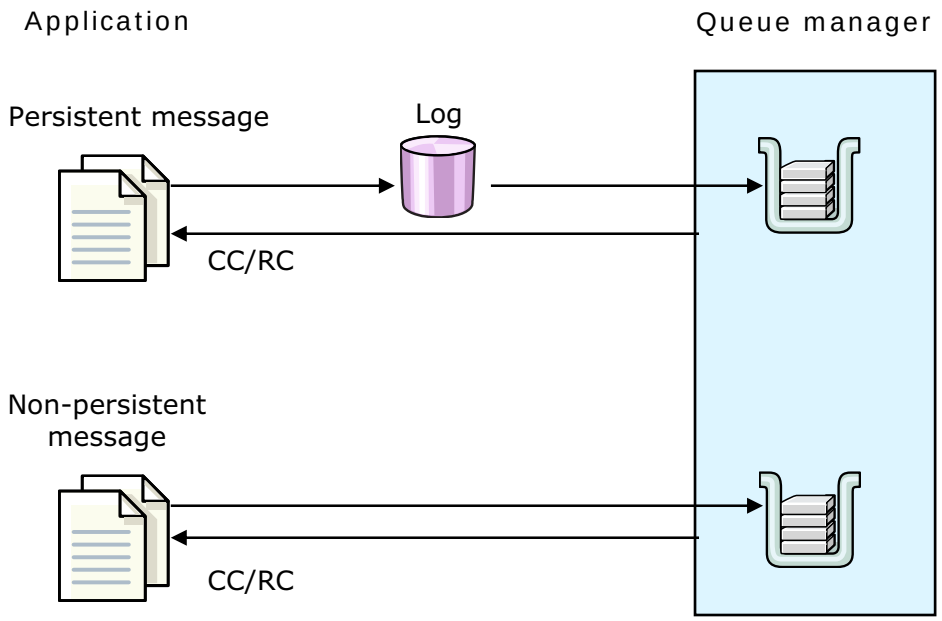


Figure 3-3. Message persistence WM100 / VM1001.0

Notes:

A message is said to be *persistent* if it survives a queue manager restart. This applies whether the queue manager was stopped by an operator command or because of a system failure. Persistent messages must be written to a log. If a queue manager is restarted after a failure, it recovers these persistent messages as necessary from the logged data.

A message is said to be nonpersistent if it does not survive a queue manager restart. This applies even if a queue manager finds an intact nonpersistent message on disk during restart; by default, the queue manager discards it. On non-z/OS queue managers, if a queue is set to NPMCLASS(HIGH), the queue manager does its best to rebuild all messages, even non-persistent ones. On z/OS systems nonpersistent messages are always discarded.

The persistence of a message is determined by the value of the *Persistence* field in its message descriptor.

Both persistent and nonpersistent messages can be stored on the same queue.

Retrieval in priority order

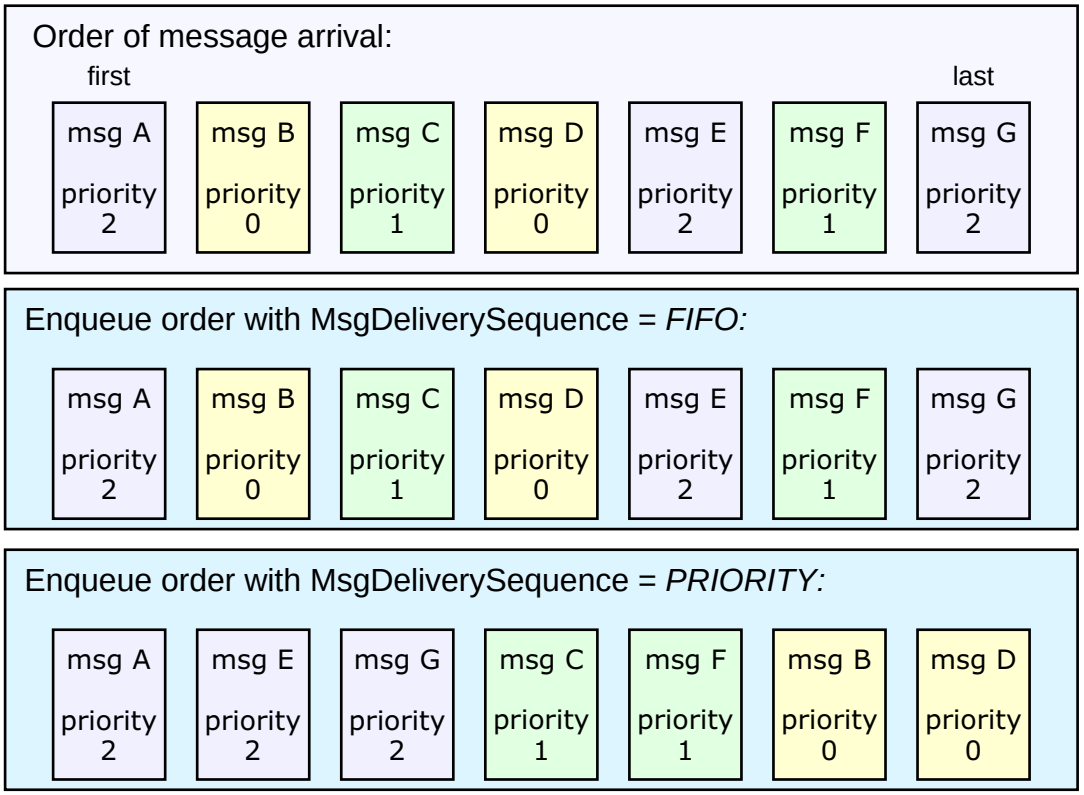


Figure 3-4. Retrieval in priority order WM100 / VM1001.0

Notes:

When a message is put on a queue, the application may set the *Priority* field in the message descriptor to a value in the range 0 (lowest priority) to 9 (highest priority). If no value is specified, the value of the *DefaultPriority* attribute of the queue is assigned to the message.

The value of the *MsgDeliverySequence* attribute of a queue determines how messages are placed in the queue. If the MsgDeliverySequence is **priority**, the message is enqueued according to the value of the priority field in the MQMD, with messages of the same priority being enqueued in FIFO (first-in first-out) order. If the MsgDeliverySequence is **FIFO**, messages are enqueued in FIFO order at the priority specified by the *DefaultPriority* attribute of the queue.

The value of the MsgDeliverySequence attribute of the queue at the time that messages are placed on the queue thus determines the order in which they are retrieved from the queue.

Report field and report message type

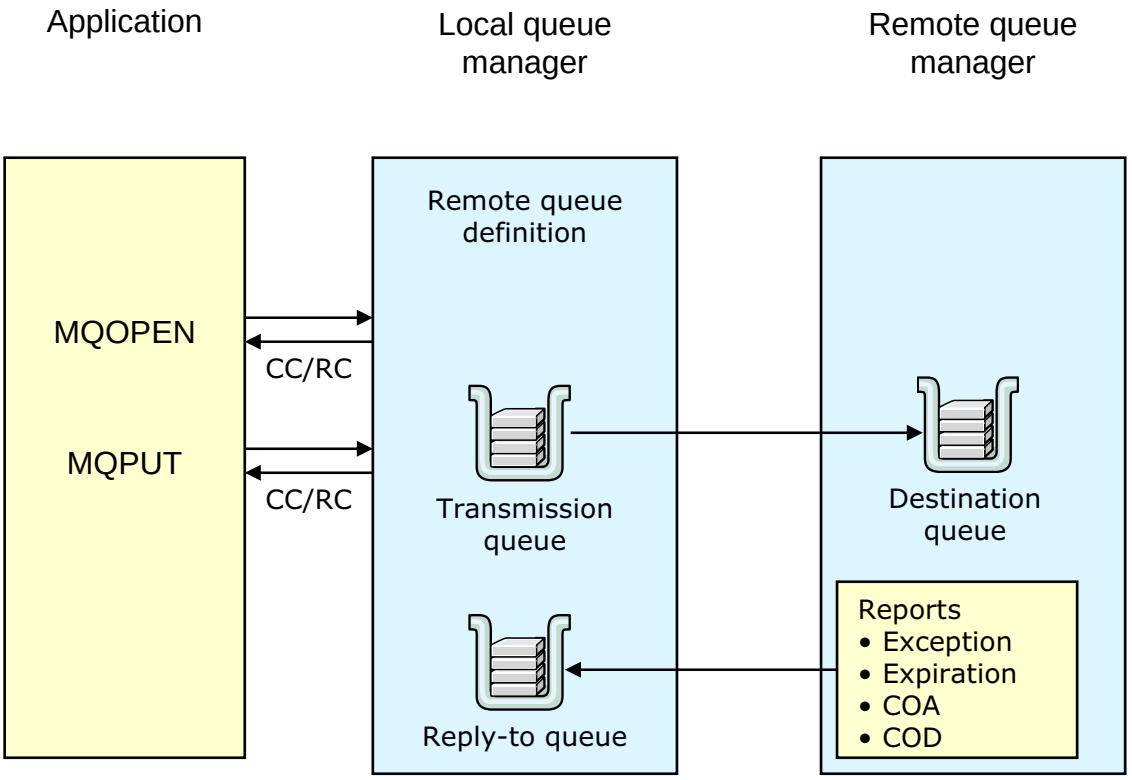


Figure 3-5. Report field and report message type WM100 / VM1001.0

Notes:

If an application issues an MQPUT call for a message destined for a remote queue and a problem is detected during the call, the queue manager reports the problem to the application immediately by means of a completion code and reason code.

If a problem occurs during the delivery of the message, the queue manager can generate a report message and send it to the reply-to queue specified in the message descriptor of the original message. The queue manager can also report on certain other events associated with the delivery of the message, in this way.

An application does not receive report messages by default. It must request them. It does this when it puts a message on a queue by specifying which reports it wishes to receive in the *Report* field in the message descriptor. If more than one type of report is required, the values can be added together or combined using the bitwise OR operation.

The following are some of the types of reports that can be requested:

Exception

The queue manager generates an exception report if a message cannot be

handled in any way; for example, the message cannot be put on the destination queue because the queue is full.

Confirm on arrival (COA)

The queue manager generates a COA report when a message is put on the destination queue.

Confirm on delivery (COD)

The queue manager generates a COD report when a message is delivered to the receiving application.

Expiration

The queue manager generates an expiration report if a message is discarded prior to delivery because its expiry time has passed. You will examine expiry later in this unit.

COA and COD reports can be used to implement non-repudiation function within an application, for example, preventing a customer from claiming that a bill never reached him or a bank from denying that a money transfer arrived. Such functionality may be more trusted when the basic mechanism is provided by the underlying system — IBM WebSphere MQ — rather than by the application itself.

In a report message, the *Feedback* field in the message descriptor indicates the nature of the report (for example, COA report or COD report) or, for an exception report, the reason for the report (for example, the destination queue is full, the message is too big for the destination queue, or the like).

Note that this is a second use of the fields *ReplyToQ* and *ReplyToQMgr* in the message descriptor. Earlier in the course you saw that these fields may also be used by a server application to determine where a reply message should be sent.

Expiry

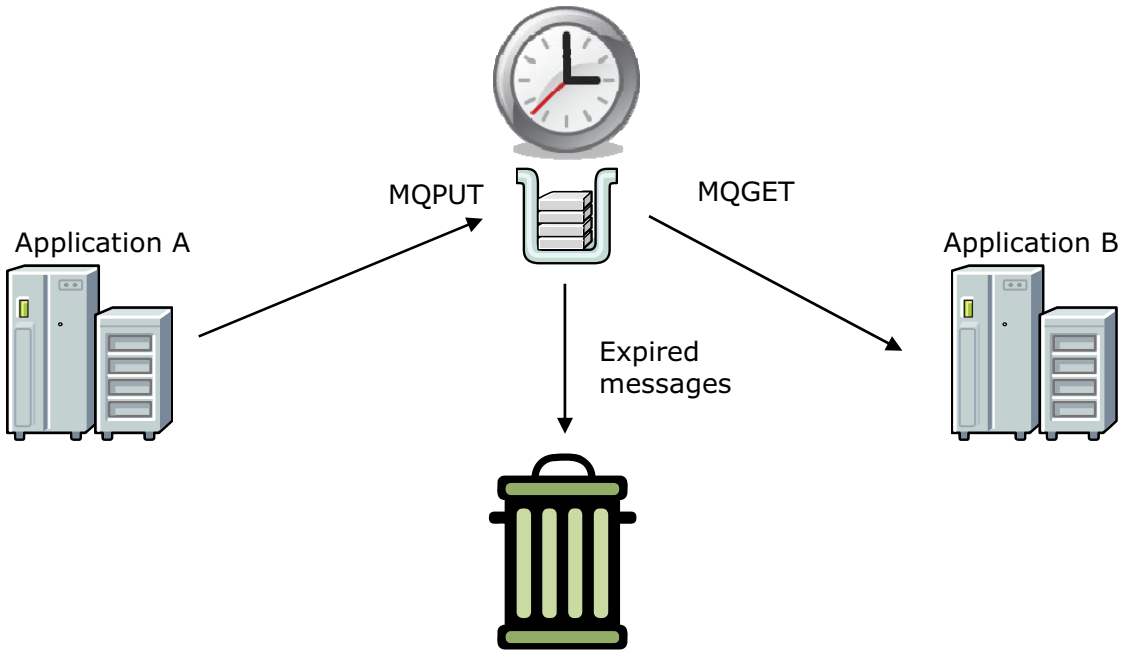


Figure 3-6. Expiry WM100 / VM1001.0

Notes:

When an application puts a message on a queue, it may set an expiration time for the message in the **Expiry** field of the message descriptor. This is a period of time expressed in tenths of a second. It specifies the total amount of time that the message may spend on the destination queue and also on any transmission queues, if the message is put to a remote queue.

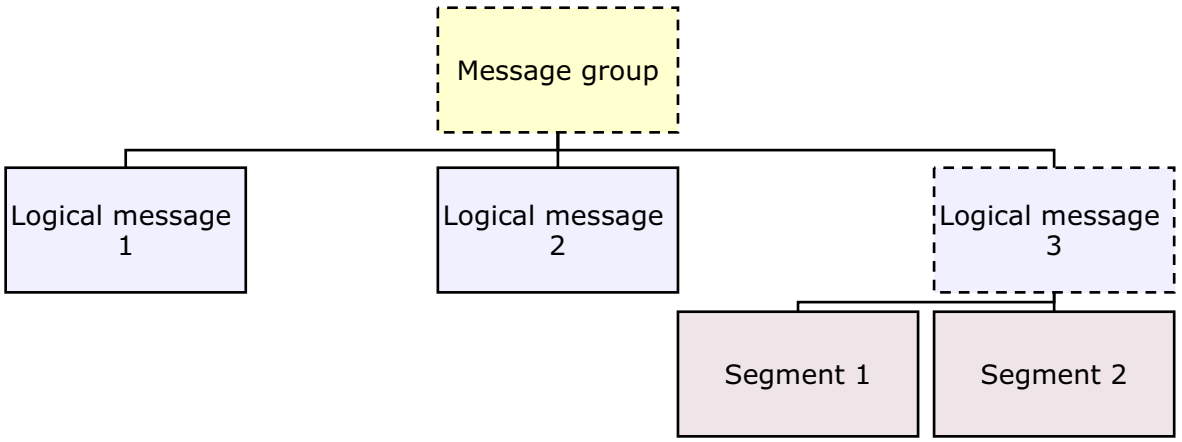
When the receiving application attempts to get the message from the destination queue, the queue manager subtracts the amount of time the message has been on the queue from the value in the **Expiry** field in the message descriptor. If the resulting value is zero or negative, the message is not returned to the application; instead, it becomes eligible to be discarded. The next suitable message in the delivery sequence for the queue is returned to the application instead.

If it has been requested, an expiration report is sent to the reply-to queue as specified in the message descriptor of the expired message.

The queue manager does not immediately purge expired messages. When a message has expired, the queue manager purges it the next time it processes an MQGET call. The WebSphere MQ administrator can also configure the queue manager to scan periodically for expired messages and discard them instead of waiting for an MQGET call to be processed.

There is no need for clocks to be synchronized across a network of queue managers for this to work.

Message groups



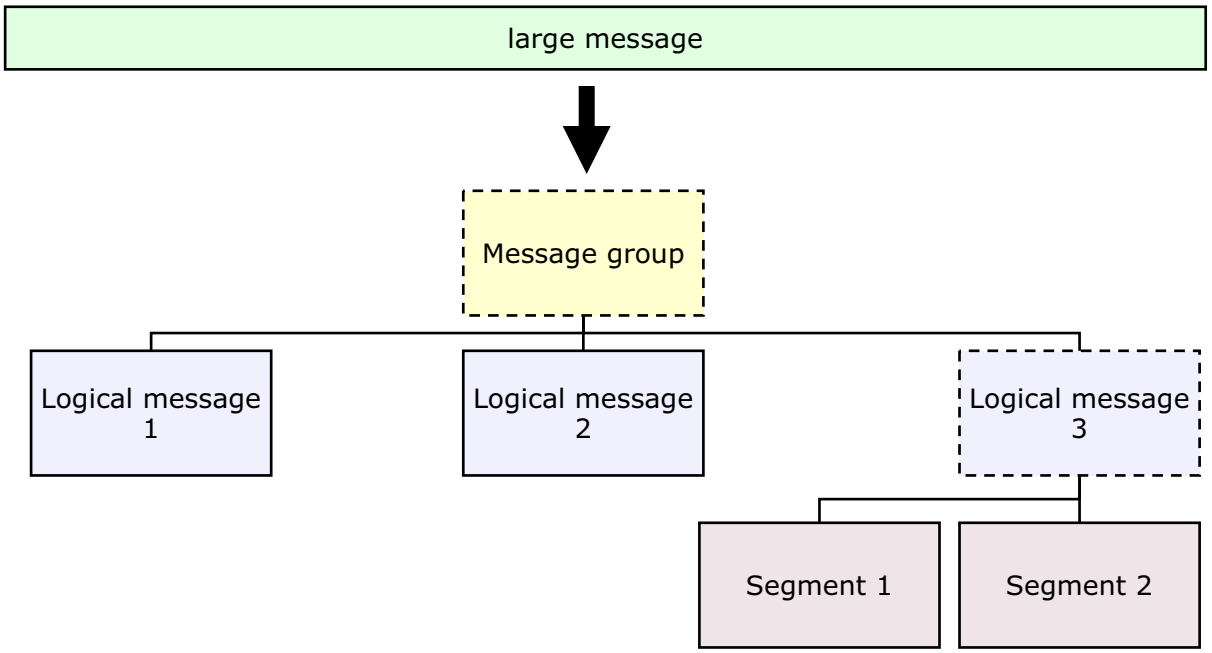
- **Message group**:
  - Consists of one or more logical messages
- **Logical message** is:
  - A physical message (unless it is split into segments)
  - Identified by the GroupId and MsgSeqNumber fields in the message descriptor
- A message group is **needed to**:
  - Ensure ordering on retrieval (where it is not already guaranteed)
  - Allow an application to group together related messages
- Segmentation and reassembly can be done by the application or by the queue manager

Figure 3-7. Message groups WM100 / VM1001.0

Notes:

Messages can occur within logical groups. This allows for individual messages to be processed in a specific order. A *group* is the highest level in the hierarchy, and is identified by a *GroupId*. It consists of one or more messages that contain the same GroupId. These messages can be stored anywhere in the queue, not necessarily in physical consecutive order. Each message within a group consists of one physical message, unless it is split into segments. Each message is logically separate from the others; messages in a group are related to other messages within the same group by their GroupId and MessageSeqNumber fields.

Message segmentation



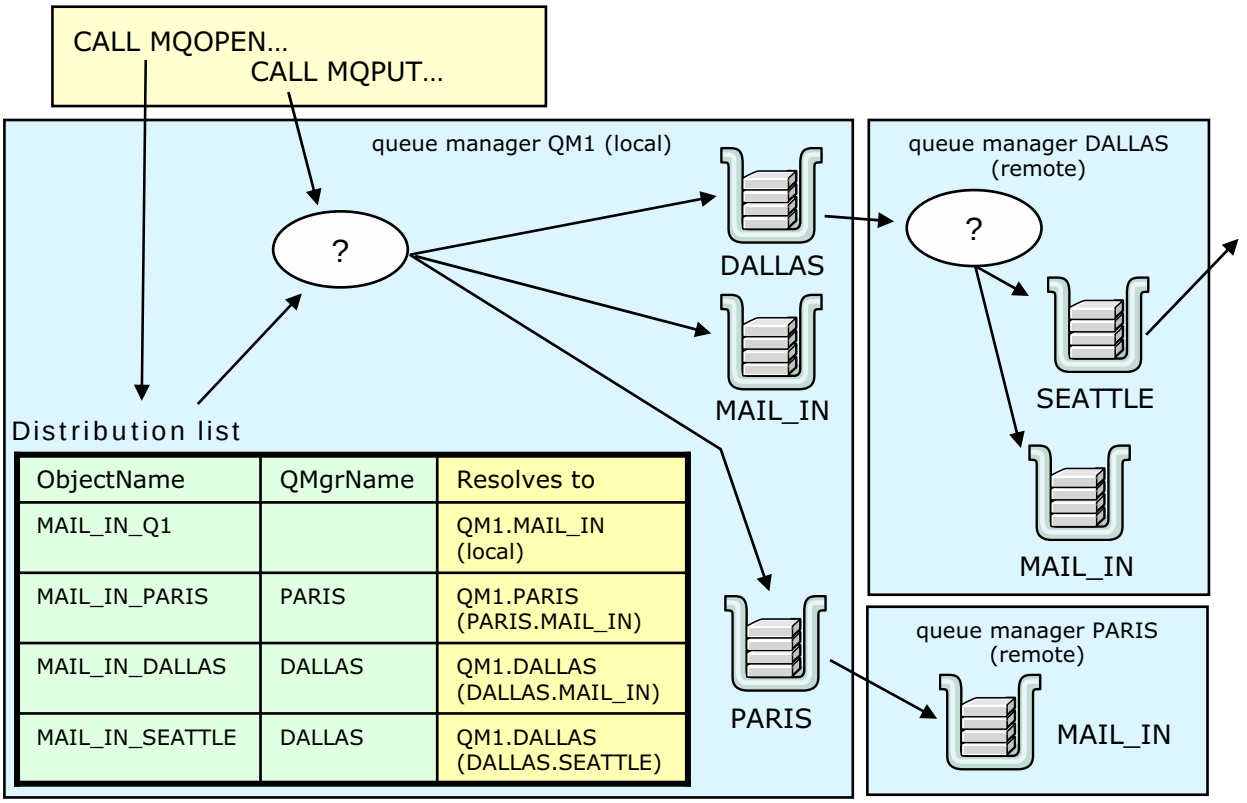
- Message too large for queue, queue manager, or application can be segmented
- Segmentation performed by queue manager or by application

Figure 3-8. Message segmentation WM100 / VM1001.0

Notes:

A message may be too large for a queue or queue manager to handle in a single buffer because of the way that the queue or queue manager is configured. A receiving application may also not be able to handle an entire large message at once. In either case, the message can be segmented into smaller pieces. The sending application can perform the segmentation itself and then issue an MQPUT call for each segment; the receiving application then issues an MQGET call for each segment. An application can also issue a single MQPUT call and allow the queue manager to segment the message on behalf of the application. When the receiving application issues the MQGET to retrieve the message, the queue manager reassembles it before returning it to the application.

Distribution list



ObjectName	QMgrName	Resolves to
MAIL_IN_Q1		QM1.MAIL_IN (local)
MAIL_IN_PARIS	PARIS	QM1.PARIS (PARIS.MAIL_IN)
MAIL_IN_DALLAS	DALLAS	QM1.DALLAS (DALLAS.MAIL_IN)
MAIL_IN_SEATTLE	DALLAS	QM1.DALLAS (DALLAS.SEATTLE)

Figure 3-9. Distribution list WM100 / VM1001.0

Notes:

A *distribution list* allows an application to put a message to multiple destinations using a single MQPUT call.

The application first opens a distribution list using an MQOPEN call. Once a distribution list has been opened, the application can call MQPUT to put a message on the queues in the list. The queue manager responds by putting **one** copy of the message on each local queue, including the transmission queues. Thus, in the example depicted in the figure, only one copy of the message is put on the transmission queue DALLAS, even though the message is ultimately destined for two queues.

An important property of the implementation is that a message destined for multiple queues is only replicated at the last possible point at each stage of its journey. In this way, network traffic is minimized.

Checkpoint

Exercise — Unit 3 checkpoint

1. Messages are recovered when a queue manager is restarted if they are:
   a. Saved by the operator
   b. Persistent
   c. High-priority messages
   d. Retrieved using MsgId and CorrelId
2. A message group is made up of (select all that apply):
   a. One or more physical messages
   b. One or more logical messages
   c. Only one logical message but several physical messages
   d. All of the above

Unit summary

Having completed this unit, you should be able to:

- Describe some of the fields in the message descriptor and how they relate to the MQ environment
- Explain the various orders in which messages can be retrieved from a queue
- Describe message groups and segmentation and why they are used
- Discuss distribution lists

Figure 3-10. Unit summary WM100 / VM1001.0

Notes:

This unit offered a brief look at some of the fields in the message descriptor and how they relate to the IBM WebSphere MQ environment, as well as a method of sending the same data to multiple destinations using a distribution list.

Unit 4. Intercommunication

What this unit is about

This unit describes the function of interconnected systems using IBM WebSphere MQ.

What you should be able to do

After completing this unit, you should be able to:

- Describe message channels and message channel agents
- Explain the function of a transmission queue
- Define how a channel is triggered
- Describe queue manager clusters

How you will check your progress

Accountability:

- Instructor questions
- Classroom discussion

References

www.ibm.com/software/integration/wmq/library

IBM WebSphere MQ Library

SC34-6587 *WebSphere MQ Intercommunication*

SC34-6597 *WebSphere MQ Script (MQSC) Command Reference*

SC34-6589 *WebSphere MQ Queue Managers Clusters*

Unit objectives

After completing this unit, you should be able to:
- Describe message channels and message channel agents
- Explain the function of a transmission queue
- Define how a channel is triggered
- Describe queue manager clusters

Figure 4-1. Unit objectives WM100 / VM1001.0

Notes:

4.1. Intercommunication — distributed queuing

The message channel

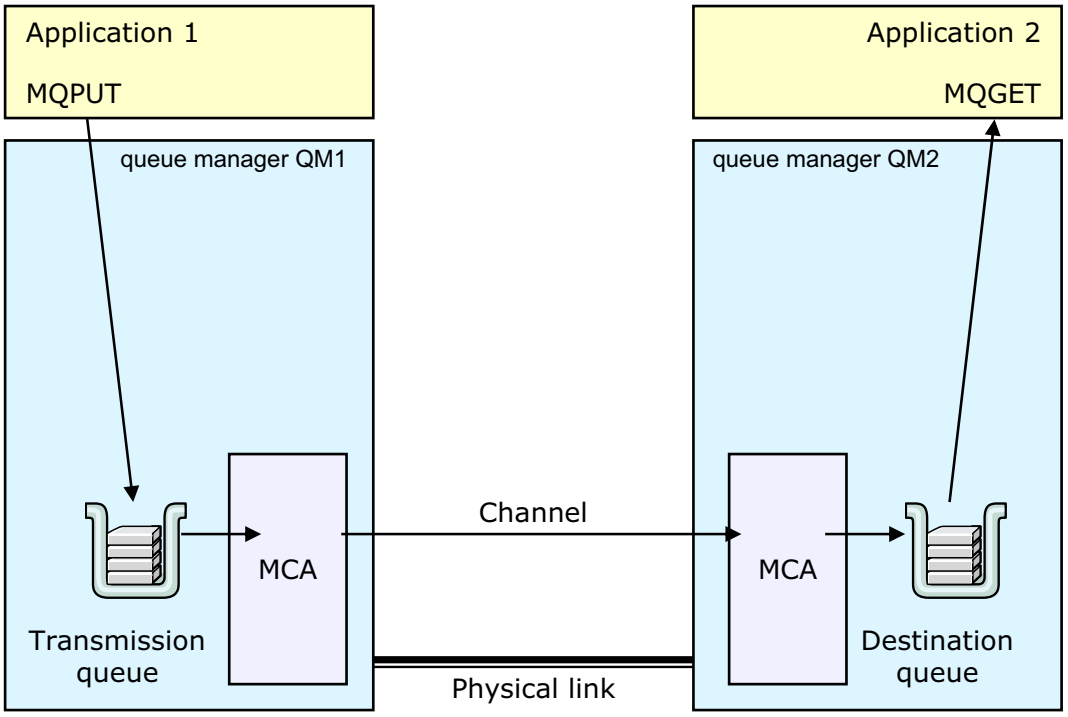


Figure 4-2. The message channel WM100 / VM1001.0

Notes:

This figure is a reminder of the structure of a simple network connection.

An application issues an MQPUT call. As a result, a message is placed on the transmission queue. A message channel agent (MCA) is started, which reads it from the queue, and sends it to a partner MCA. The partner MCA places it on the destination queue. At some point, the target application issues an MQGET call and reads the message from the queue.

To send a message, the following are needed:

- A delivery mechanism across the link
- A connection mechanism between the systems
- A protocol between the sending MCA and the receiving MCA

The physical link can be anything the system supports (token ring, ATM, and so forth). The connection mechanism can be SNA LU6.2, TCP/IP NetBIOS, DECnet, or SPX. Not all of these are supported on all platforms. The *WebSphere MQ Intercommunications* manual discusses the networking setup requirements of various MQ environments.

The message channel protocol between the message channel agents is the common thread that allows the MCAs on the various platforms to communicate.

A transmission queue is an ordinary local queue with the *Usage* attribute set to `MQUS_TRANSMISSION` instead of `MQUS_NORMAL`.

Only a queue manager (not an application) should put a message directly on a transmission queue, and only an MCA should get a message from a transmission queue. This is because a message on a transmission queue must have a special header called the *transmission queue header*. The queue manager provides this header when it puts the message on the transmission queue. The header also accompanies the message when it is sent across a message channel. The receiving MCA removes the header before putting the message on the destination queue. Normally, an application program need never be aware of this header.

To reach a destination queue on a remote queue manager, the local queue manager needs to know which transmission queue to put the message on. Each queue manager contains common software known as the *moving service* component, which allows a queue manager to communicate with other queue managers.

Types of channels

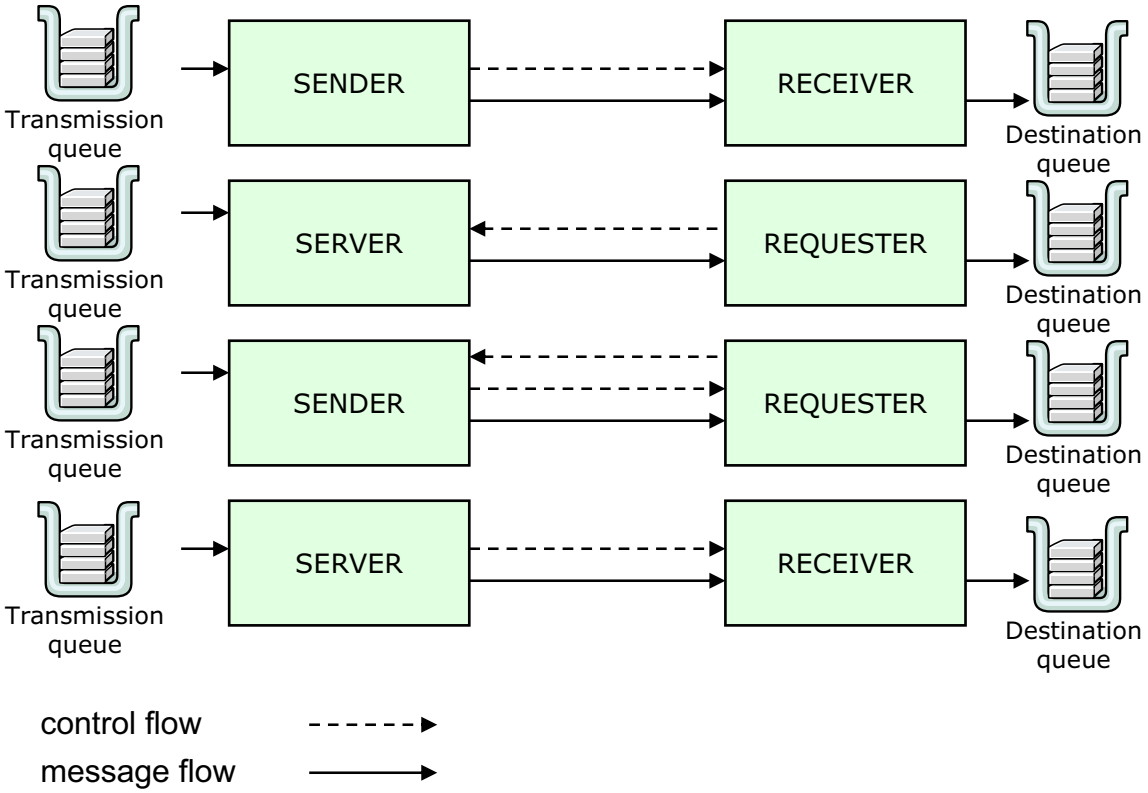


Figure 4-3. Types of channels WM100 / VM1001.0

Notes:

There is a pair of MCAs for each message channel, a sending MCA and a receiving MCA. A sending MCA is one that sends messages over a network connection to a receiving MCA at the other end.

When defining a channel at one end of the channel, a sending MCA may be defined as a SENDER or a SERVER, and a receiving MCA as a RECEIVER or a REQUESTER.

The last set of channels listed serves a very specific purpose. These channels support the connection between a WebSphere MQ server (SVRCONN) and the WebSphere MQ client (CLNTCONN).

In the figure, an arrow with a dotted line indicates a control flow at channel initiation. In particular, the first such arrow for each pair of MCAs represents the control flow that makes initial contact for the channel. The solid line arrows indicate the direction of flow of the messages once the channel has started.

Neither a RECEIVER nor a SVRCONN can start a channel; the other types may do so.

Generally, every channel must be defined at both ends of the channel, and the name of the channel must be the same at both ends. In the Version 5 and iSeries V4R2 queue managers, it is not required that a RECEIVER or a SVRCONN be defined; it can be done automatically.

When a channel is being defined, each end must be allocated a **channel type** as part of its definition. There are six channel types to choose from, namely:

- A **sender**
- A **receiver**
- A **requester**
- A **server**
- A **svrconn**
- A **clntconn**

The channel types at both ends of a channel must be compatible with one another, and the graphic depicts the compatible combinations. The compatible combinations of channel types are as follows:

Sender-receiver

This is the most frequently used combination.

Requester-server

This combination can be used when the end of the channel that sends the messages is not able to start the channel. It may be running unattended and thus unable to issue the appropriate command.

This combination could also be used when the physical network connection between two nodes is not permanent. A laptop computer, which can be connected to a server machine, might be one example. In this case, the most appropriate combination might be one that allows the laptop, when it is connected, to start a channel and then receive messages from the server machine.

Requester-sender (callback)

In this combination, the requester starts the channel but the sender closes it immediately. The sender then restarts the channel according to its channel definition. This feature is known as *callback*. It can provide additional security when, for example, the request to start a channel is coming from an untrusted environment.

Server (fully defined) — receiver or requester

A fully defined server is one whose channel definition has the attributes needed to start a channel — in particular, the LU name, the TCP host name, or the IP address. This combination is essentially the same as the *sender-receiver* combination in terms of function.

CIntconn-svrconn

(not shown in the diagram) This combination is reserved for use only in the MQI client environment. The client initiates the connection when the MQCONN is issued. Data about the MQI calls flow in both directions. The connection is terminated when the MQDISC is issued.

Starting a channel

- operator command: `START QM1_QM2`

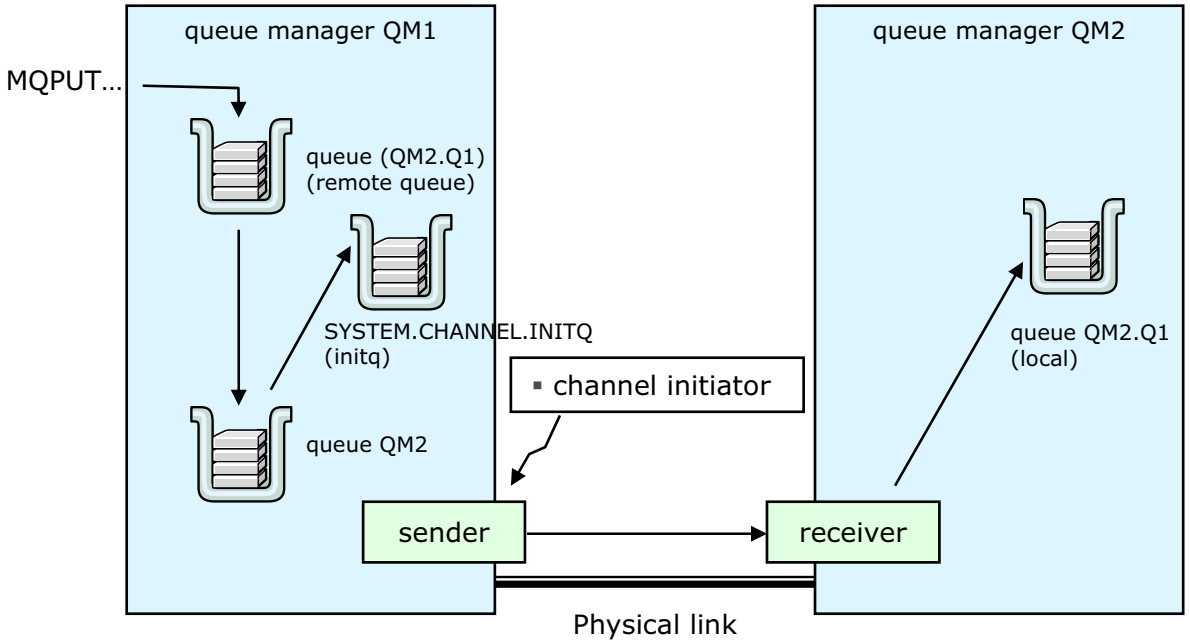


Figure 4-4. Starting a channel WM100 / VM1001.0

Notes:

Channels can be started in several ways on most queue managers:

- An operator can start most channels via a command line command or an interface.
- The transmission queue can be defined as a triggered queue, and the channel can be started when messages arrive that satisfy the triggering conditions.
- Certain channel types can be started remotely from the network.

In the figure, the *channel initiator* is a special version of a trigger monitor used just to start channels. The process used to start channels based on triggering is the same as starting other applications.

Starting and stopping a RECEIVER or a SVRCONN does not start or stop the flow of data; it merely enables or disables the channel to be started from the other side.

Stopping a channel

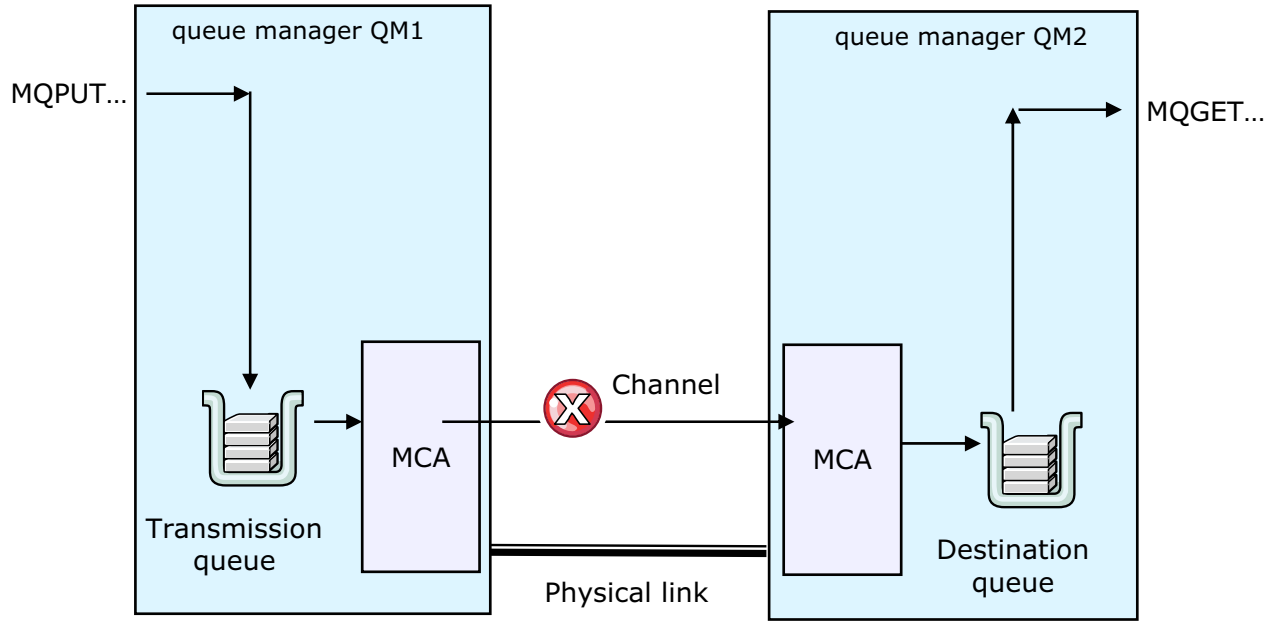


Figure 4-5. Stopping a channel WM100 / VM1001.0

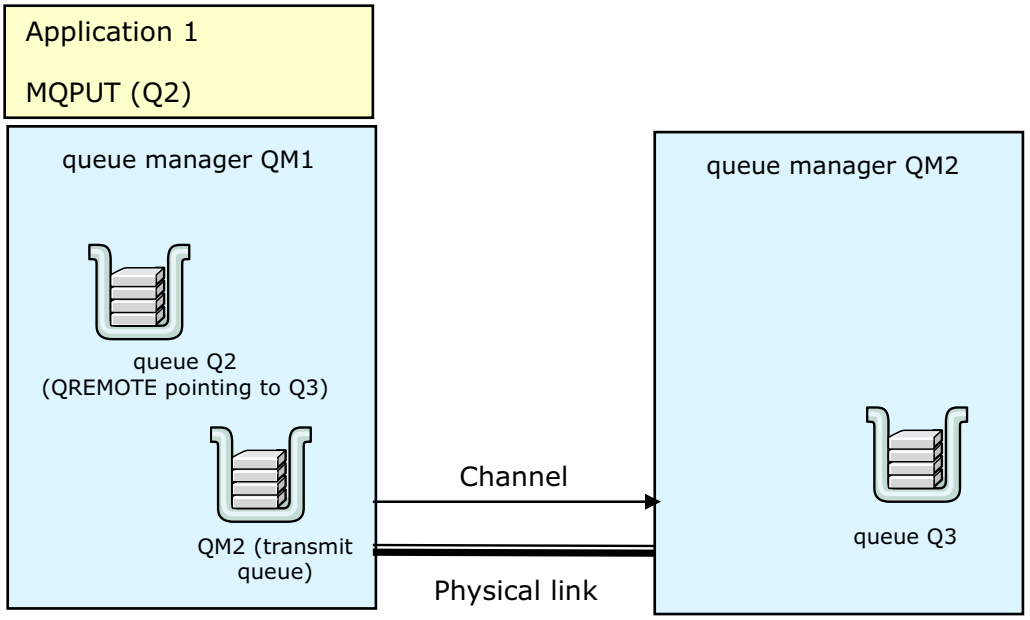
Notes:

Usually, channels between queue managers run for long periods. They are started and service a particular transmission queue. However, it is not always best to leave a channel running forever. There is a disconnect value associated with the channel. It allows for normal quiescing of a channel that is active but without work for a specified period.

It is possible to manually stop channels by command as well.

Channels can also stop during error conditions. Because IBM WebSphere MQ provides assured message delivery, messages you wish to keep will not be lost.

Remote queues



Application does not need to specify QMGR name to MQPUT, even if not connected to that QMGR

Figure 4-6. Remote queues WM100 / VM100q1.0

Notes:

Although it may be transparent to the sending application, the queue manager needs some help to know where to route messages. When a program wants to send a message to a queue on another system, the application can name the queue and the queue manager to be used. This allows the sending queue manager to easily complete the necessary addressing information in the MQ transmission queue header (MQXQH). The message is then placed on a transmission queue, either one with the same name as the remote queue manager or if defined as the local queue manager's default transmission queue. However, specifying the queue manager name at the application code level is generally not the best solution, except in the case of sending replies.

Usually, a program simply opens a queue without specifying a queue manager. This is an indication to the local queue manager that the queue named by the application is one that should be found among its own definitions (not necessarily a local queue). One of the types of definitions that the queue manager will check is the **QREMOTE** definition.

If the queue named by the application is a QREMOTE definition, the queue manager can use the information from that definition to complete the addressing information in the

MQXQH. The QREMOTE also allows for the specification of a transmission queue to be used. The queue manager can then place the message on a named transmission queue, allowing for greater flexibility. You explore the use of these alternatives as you proceed.

Note

An application may MQPUT to a remote queue, but it cannot issue an MQGET call to a remote queue.

For example, you can send a letter from any mailbox (queue manager). You have to be local to the mailbox (queue manager) to retrieve the letter. However, IBM WebSphere MQ zSeries has the cabability for multiple queue managers to share a single local queue from an MQGET perspective.

Message concentration

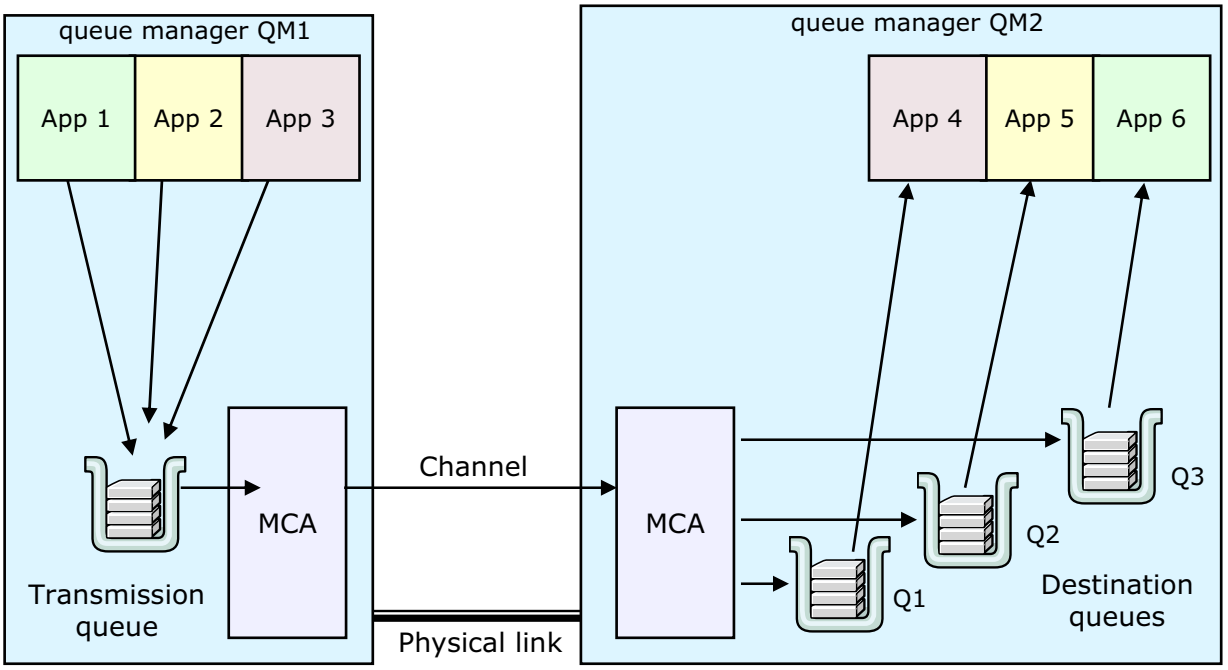


Figure 4-7. Message concentration WM100 / VM1001.0

Notes:

Given sufficient capacity, one channel can carry all the messages between a pair of queue managers. This is queue manager-to-queue manager communications. Using the MQI is less complex than direct application-to-application communication; that is, it is less complicated for applications to use the MQI than it is to write code to allow applications to talk directly with each other. This is one of the fundamental principles of IBM WebSphere MQ .

The sending queue manager builds transmission headers (MQXQH) for each message to be sent on a particular channel. It includes the destination queue manager name as part of that header. When the destination MCA receives the message, the MQXQH tells it where the message is to go. The MQXQH is stripped from the message prior to its delivery to the appropriate destination queue. In this scenario, there is no need for multiple paths between the two queue managers.

Message segregation

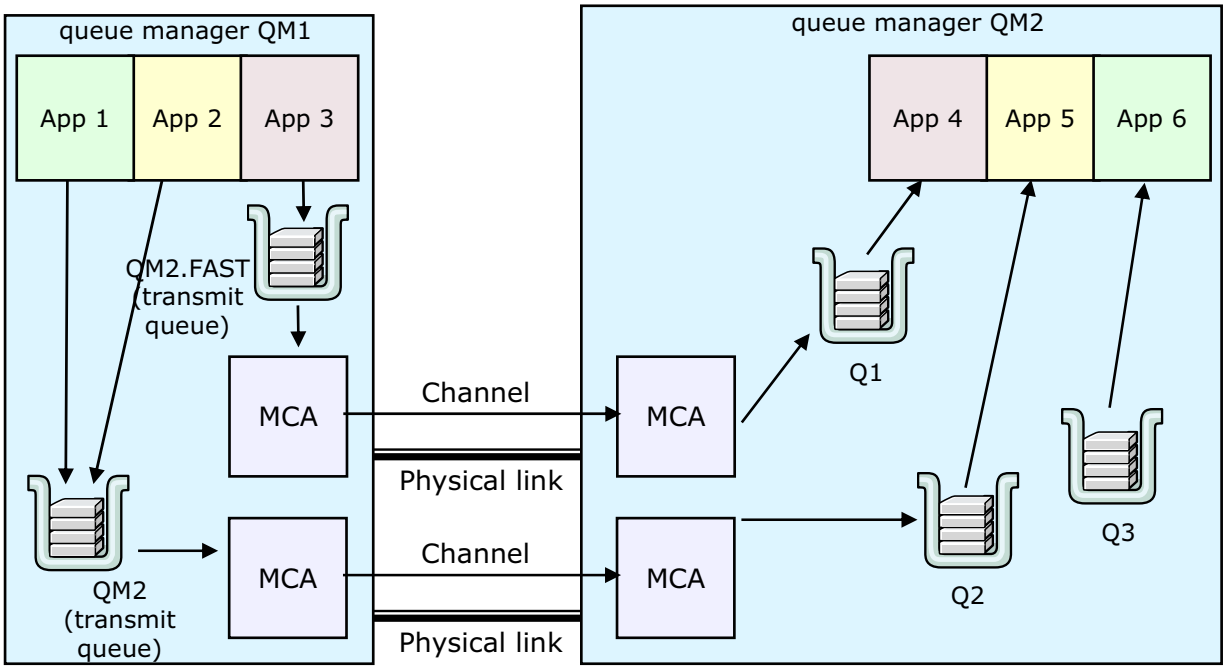


Figure 4-8. Message segregation WM100 / VM1001.0

Notes:

When there is a requirement to keep different forms of traffic apart, different queue definitions can point to different transmission queues and thus different channels.

Notice that each channel has its own MCA on each side ***and*** its own transmission queue. The figure shows a separate physical link, but it is also possible to use a single physical link for both channels.

In this scenario, the benefit of being able to explicitly name a transmission queue becomes apparent. Messages that need to go on the second channel can be directed there without work from the program.

Multiple hops

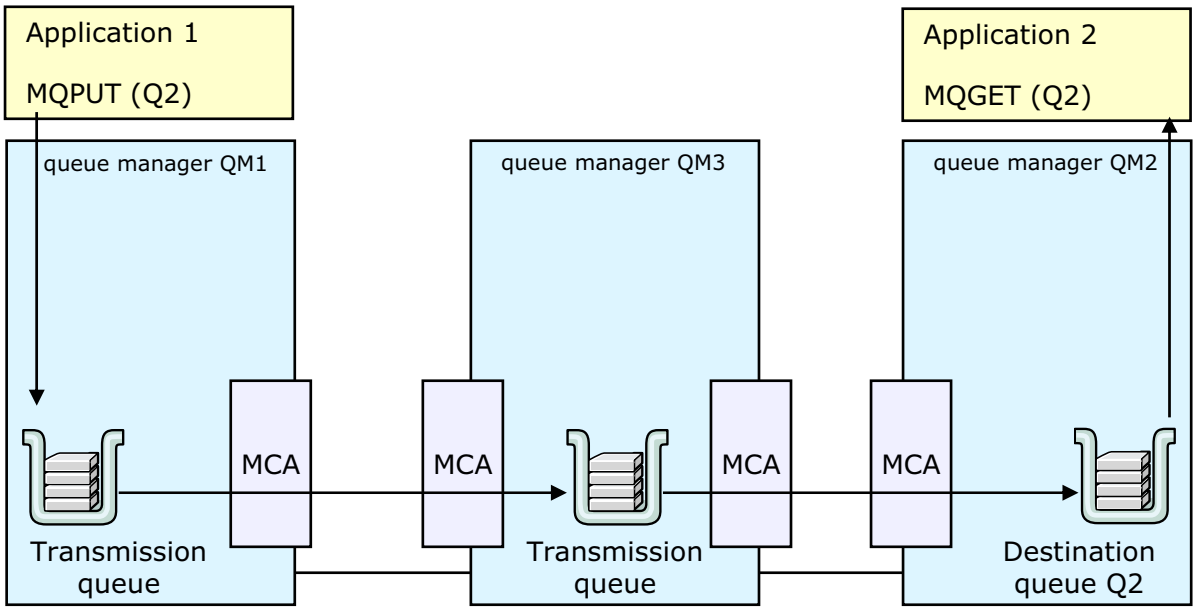


Figure 4-9. Multiple hops WM100 / VM1001.0

Notes:

So far, you have been looking at a very small portion of a network and considering two queue managers. Using proper definitions, an application can issue MQPUTs that result in messages travelling across intermediate queue managers to reach the final destination queue. In this case, the intermediate queue managers may be looked upon as a sort of gateway. The figure shows application 1 issuing an MQPUT call with a message destined for queue 2 on queue manager 2. The message travels via queue manager 3.

The definitions needed to accomplish this need coordination between the various queue managers, but the application program would not be required to handle the MQPUT any differently.

Channel exits

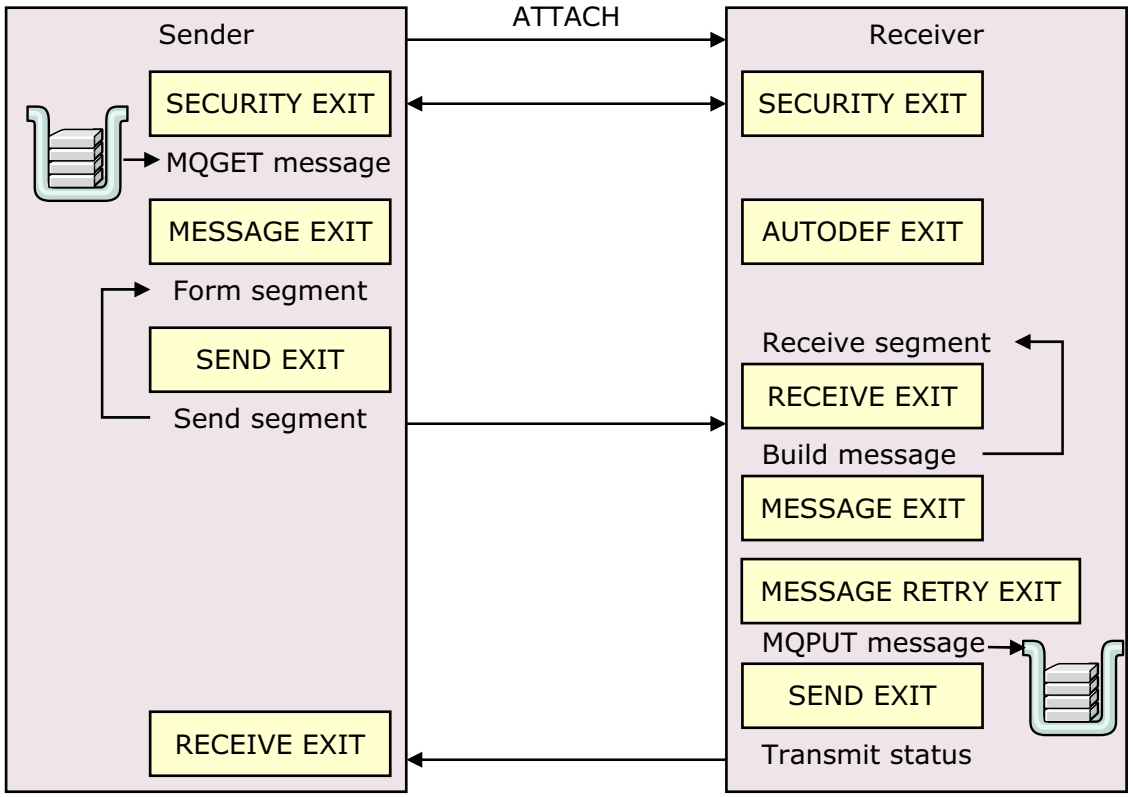


Figure 4-10. Channel exits WM100 / VM1001.0

Notes:

At both ends of the channel, exits may be called:

- At initialization time
- When a message is read from the transmission queue or placed on the target queue
- When a segment is transmitted or received

The security exit is used to establish trust between partners. Commonly it involves the exchange of encrypted data to ensure that a common encryption technique and key are being used.

The message (or GET/PUT) exit is commonly used to compress and encrypt messages, and to change values in the MQMD.

The segment (or SEND/RECEIVE) exit may also be used for encryption and compression.

The retry exit allows you to make more than one attempt to MQPUT a message to its destination queue.

Application data conversion

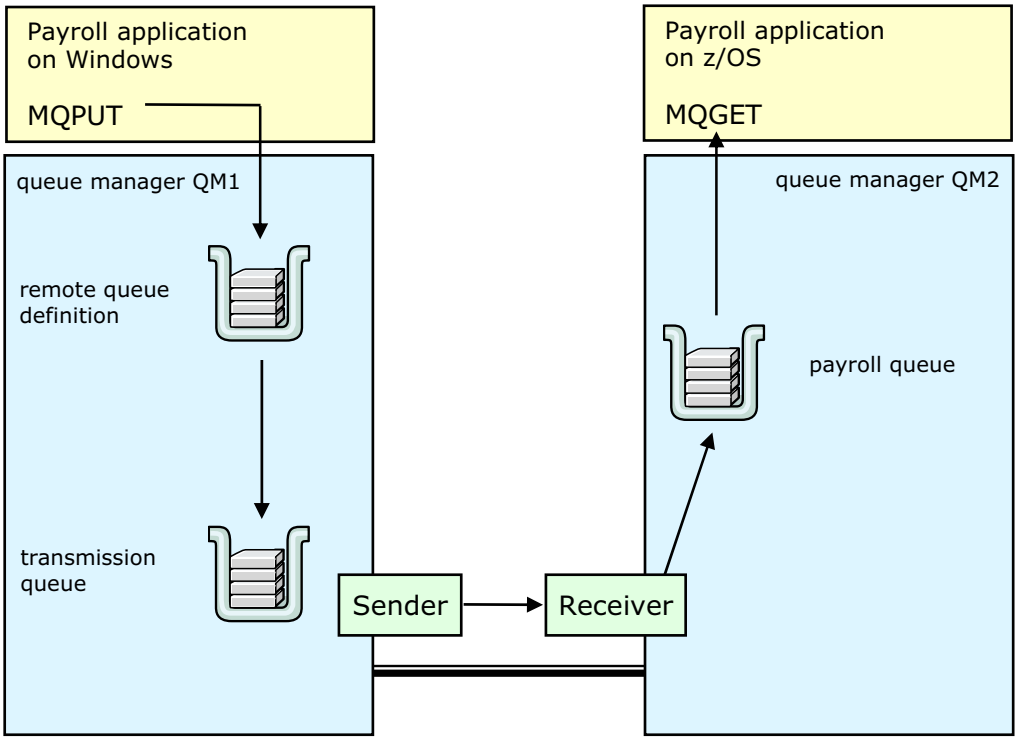


Figure 4-11. Application data conversion WM100 / VM1001.0

Notes:

In this example, you see payroll data flowing from a Windows environment to an z/OS environment. This is an everyday occurrence in many installations. However, data that flows between disparate platforms in a network introduces a special problem; it is not represented in an identical manner everywhere. Whether integer or character, ASCII or EBCDIC, US English or French, data conversion is an issue that must be considered.

IBM WebSphere MQ offers some assistance with this. In general, as long as the data is character, an application can identify it as such in the message descriptor (MQMD), and the receiving queue manager can handle the conversion, if requested. In the case where numeric and character data make up the application data, user-written exits will need to be provided. The queue manager invokes the format field identified exit during MQGET processing, if asked to do so.

As a result, if a message is sent in ASCII, it will be converted to EBCDIC in this example, and the receiver will be able to understand the information.

Channel attributes example

- At sender side:

```
DEFINE CHANNEL(ATLANTA_HURSLEY)  CHLTYPE(SDR) +
  TRPTYPE(TCP)  CONNAME(10.1.23.17)  XMITQ(HURSLEY) +
  DISCINT(6000)  HBINT(300)
```

- At receiver side:

```
DEFINE CHANNEL(ATLANTA_HURSLEY)  CHLTYPE(RCVR) +
  MSGEXIT(CHKUSER)
```

Figure 4-12. Channel attributes example WM100 / VM1001.0

Notes:

Only a few of the possible attributes are shown above to give a flavor for the types of things that can be controlled in the channel definitions created by an administrator.

- The names on both sides must be identical (including case).
- CHLTYPE is the channel type and is required.
- TRPTYPE is the transport type to be used and must match on both ends.
- CONNAME is the connection name of the partner.
- XMITQ tells the sending MCA what transmission queue to obtain its messages from.
- DISCINT is the time a sending MCA is to wait for a message on the transmission queue before becoming inactive.
- HBINT is used to ensure that both sides of a channel are still active.
- MSGEXIT contains the name of a user-written exit that can be used to interrogate or change contents of a message (including headers).

The *IBM WebSphere MQ Script (MQSC) Command Reference* lists all the attributes, and explains which are permitted on the various channel types as well as the various platforms. The use of the various attributes is further discussed in the *WebSphere MQ Intercommunication* manual.

Queue manager clusters

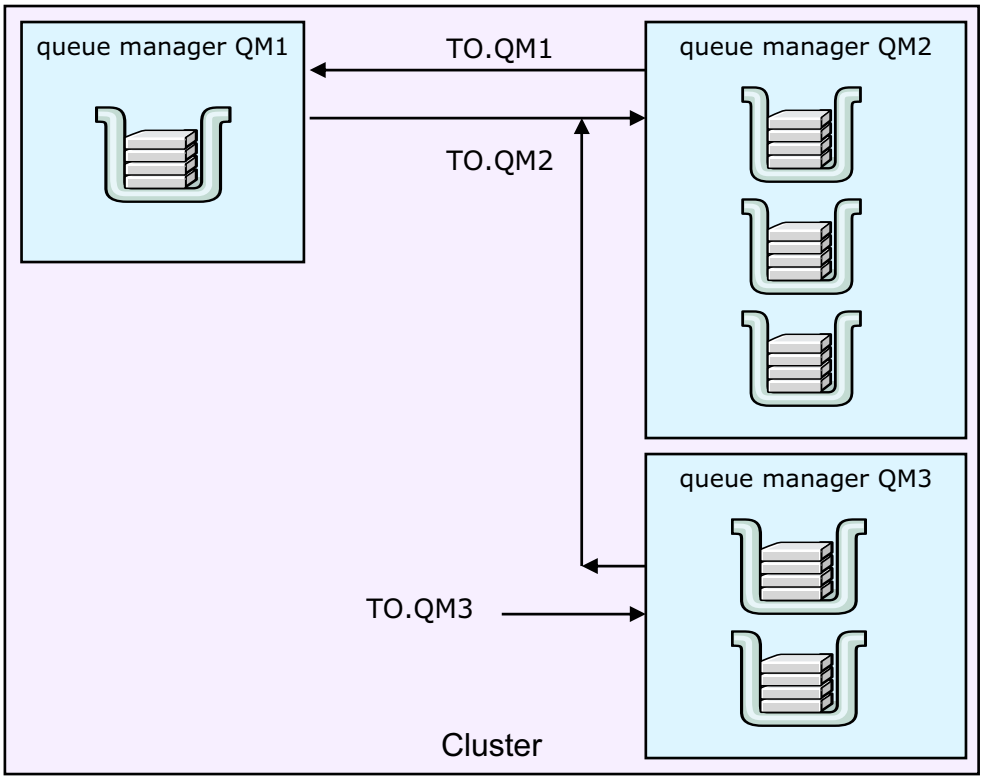


Figure 4-13. Queue manager clusters WM100 / VM1001.0

Notes:

A queue manager cluster is a network of queue managers that are logically associated in some way. The queue managers can be on the same system or across a network, even across enterprises.

Queue managers within the cluster may optionally advertise queues to other queue managers in the cluster. The other queue managers may then put messages to those queues by creating a sender channel dynamically from attributes (such as the network address) that they hold in a repository of cluster information.

With queue manager clusters, an MQ administrator does not need to define remote queue definitions and channel definitions between each of the queue managers. In addition, if a queue is defined on more than one queue manager in the cluster, the messages can be routed in the most efficient manner, enabling workload balancing as well as failover if one queue becomes unavailable — for example, because of a queue manager failure or network outage.

Cluster workload management

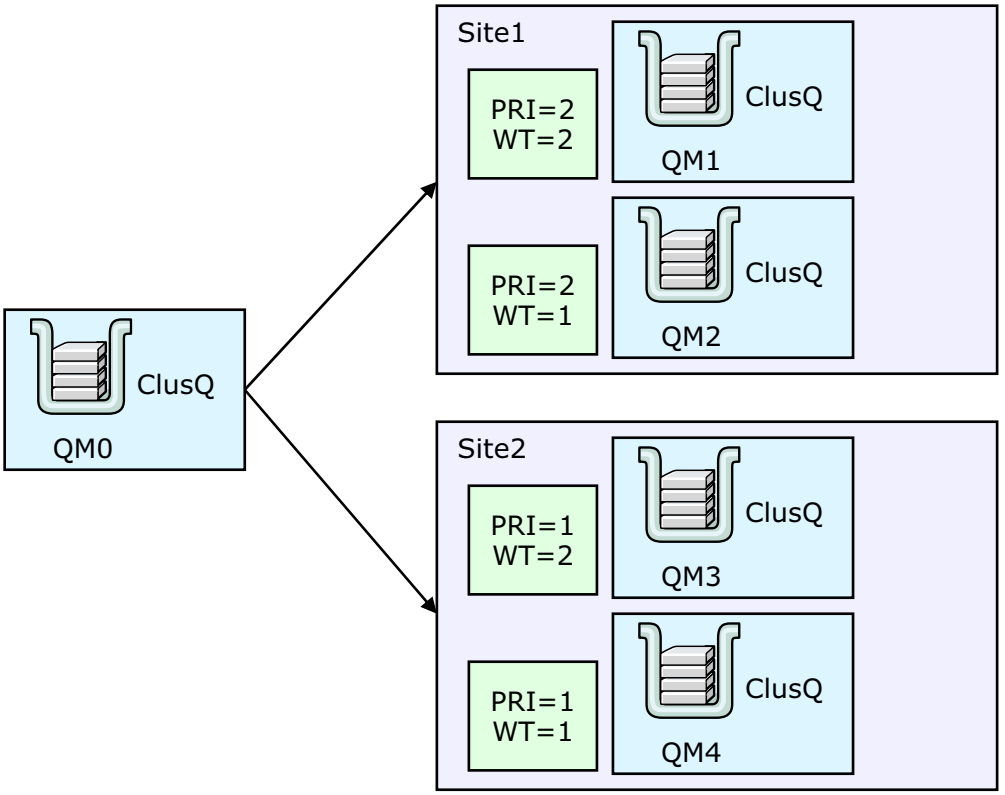


Figure 4-14. Cluster workload management WM100 / VM1001.0

Notes:

IBM WebSphere MQ clusters implement workload distribution by using various queue manager options. This figure shows how two of the options — weighting and priority — can be used. Note how the priority attribute can be used to send all messages to a primary data center, and then redirect to a backup site if the network to the prime location is down. The weighting parameter tells the cluster how to distribute the workload to the queues within the priority. In this example, two messages will be routed to QM1 for every one message routed to QM2.

This is a simplified example. There are a number of other factors that are used to determine message distribution. The *WebSphere MQ Queue Managers Clusters* manual describes the complete algorithm that is used for cluster workload management.

Shared queues Sysplex support

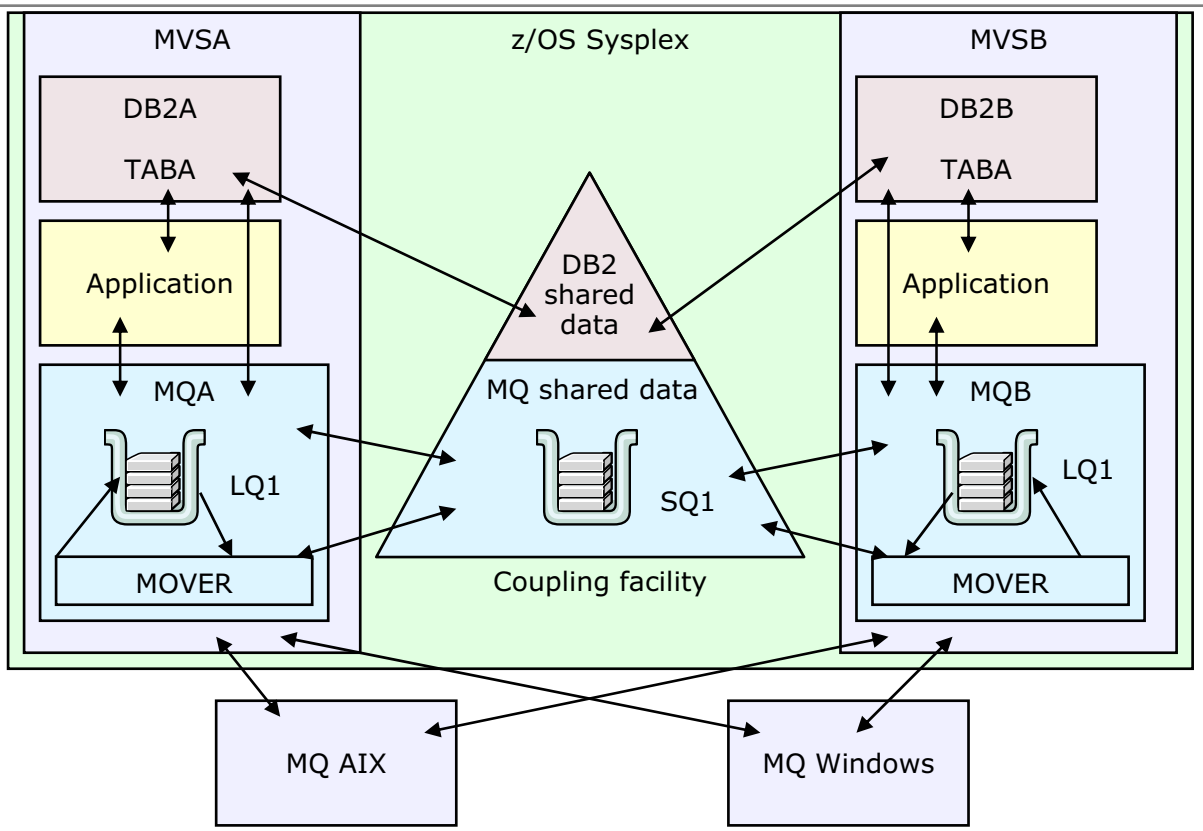


Figure 4-15. Shared queues Sysplex support WM100 / VM1001.0

Notes:

Shared queues help to balance workload and provide redundancy in case of queue manager failure.

Assume that MVSA and MVSB are cloned z/OS images running identical applications.

- With SQ1 being a shared queue, the application does not depend on the availability of a specific queue manager; any queue manager in the Sysplex can service the queue.
- If the queue manager on MVSA were to fail, the application and queue manager on MVSB can still MQGET the messages on SQ1, irrespective of whether these were originally MQPUT there by the mover on MVSA or MVSB.
  Furthermore, either mover can send the reply messages back to the remote queue manager.
- **Natural workload balancing:** If the application on each MVS image is processing a message that was retrieved from shared queue SQ1, the MVS image that finishes first will MQGET the next message from SQ1. Thus the least busy MVS image will process the most messages.

- You can add extra queue managers in extra MVS images to increase the processing power against a shared queue, up to the coupling facility capacity. This is especially useful for applications that do significant non-WebSphere MQ work.
- Intergroup communication allows messages to be delivered via the coupling facility for even faster throughput.

MQI client channels

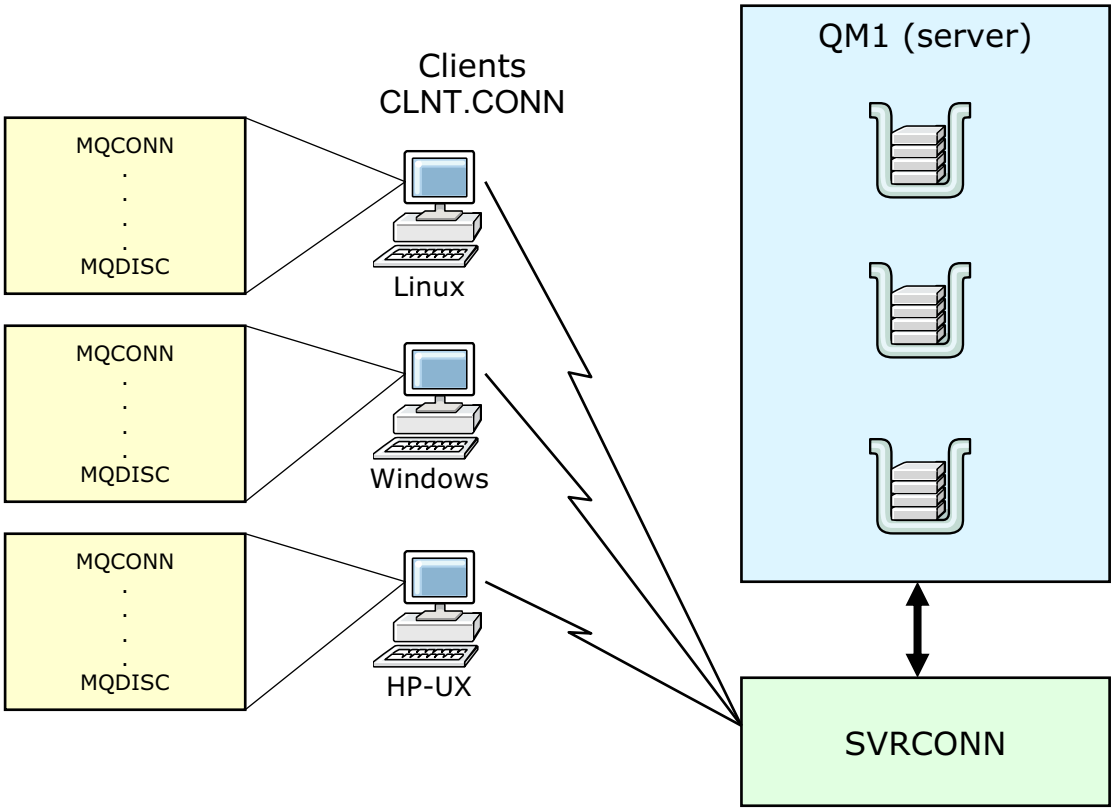


Figure 4-16. MQI client channels WM100 / VM1001.0

Notes:

WebSphere MQ clients use special channels to enable connections to the server. Many clients may connect using a single server. It is possible that one SVRCONN may support several clients. It is also possible for a client to connect to various servers.

In this picture, you see that the clients are using standard WebSphere MQ calls. However, the queue manager that the calls are directed to is on the server. When the MQCONN call is issued, the actual connection is activated. When the calls are issued, the client stub is responsible for sending the calls to the server and returning to the applications any messages coming from the server.

Although channel definitions are used, these MQI channels are very different from queue manager-to-queue manager channels. There are no transmission queues, so any assured delivery based on the MCA doing unit-of-work processing is not included. Applications therefore cannot rely on assured message delivery being provided by the MCA.

The connection data flow is bidirectional. One set of definitions will enable two-way communication. In fact, it is possible to have just a SVRCONN definition and to use environment variables for the client connection information.

Checkpoint

Exercise — Unit 4 checkpoint

1. Which of the following is ***not*** required for remote queuing between two queue managers?

   a. A delivery mechanism across a link
   b. A protocol between two message channel agents
   c. A remote queue definition
   d. A transmission queue

2. **T**/**F** WebSphere MQ message channel agents must always come in pairs.

3. **T**/**F** The following command is a valid definition for a transmission queue:
   `DEFINE QLOCAL(XMITQ1) MAXDEPTH(1000)`

4. Which of the following are valid channel types (message channel agents) that can send data held in a transmission queue?

   a. Sender
   b. Receiver
   c. Requester
   d. Server

Unit summary

Having completed this unit, you should be able to:
- Describe a message channel and message channel agent
- Explain the function of transmission queue
- Define the method of triggering a channel
- Describe queue manager clusters

Figure 4-17. Unit summary WM100 / VM1001.0

Notes:

The method of communicating between queue managers described in this unit is one of the fundamental strengths of IBM WebSphere MQ . Allowing for connectionless communications where the queue managers and message channel agents handle the delivery of messages across the network reduces the need for day-to-day administration and, in some cases, alleviates special programming requirements.

Unit 5. System administration

What this unit is about

This unit describes the administration of IBM WebSphere MQ and its queue managers.

What you should be able to do

After completing this unit, you should be able to:

- List the system administration interfaces for WebSphere MQ
- Describe the tasks for which a WebSphere MQ administrator is responsible
- Explain the concepts of logging and recovery
- Describe some of the common administrative features of WebSphere MQ

How you will check your progress

Accountability:

- Instructor questions
- Checkpoint questions

References

www.ibm.com/software/integration/wmq/library

IBM WebSphere MQ Library

SC34-6584 ***WebSphere MQ System Administration Guide***

SC34-6585 ***WebSphere MQ for z/OS V6.0 System Administration Guide***

SC34-6586 ***WebSphere MQ for iSeries V6.0 System Administration Guide***

SC34-6597 ***WebSphere MQ Script (MQSC) Command Reference***

SC34-6598 ***WebSphere MQ Programmable Command Formats*** *and Administration Interface*

Unit objectives

After completing this unit, you should be able to:

- List the system administration interfaces for IBM WebSphere MQ
- Discuss some of the tasks that a WebSphere MQ administrator is responsible for
- Explain the concepts of logging and recovery
- Describe some of the common administrative features of IBM WebSphere MQ

Figure 5-1. Unit objectives WM100 / VM1001.0

Notes:

After a brief look at the system management associated with IBM WebSphere MQ, you should realize that familiarity with a particular operating system makes working with IBM WebSphere MQ on that system a straightforward task.

IBM WebSphere MQ, in general, takes advantage of the facilities available in the operating system to accomplish its system management tasks.

5.1. System administration

Introduction

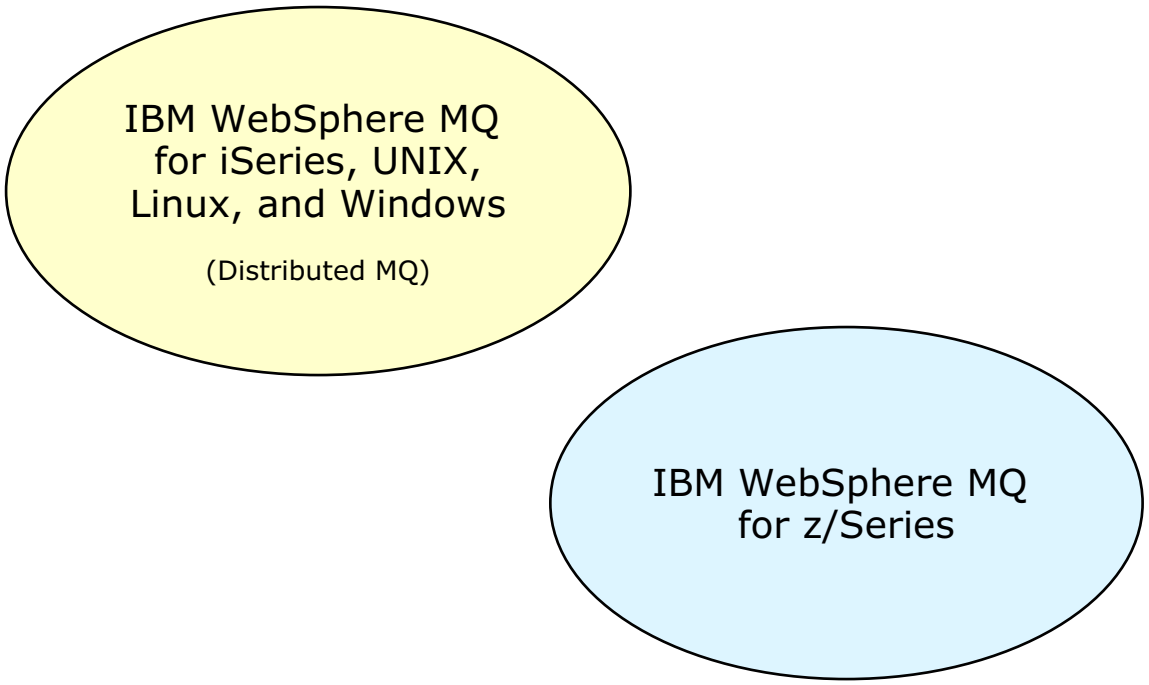


Figure 5-2. Introduction WM100 / VM1001.0

Notes:

There are family resemblances in the platforms built on each of the different code bases. This means that, in general, much of what is known about administering IBM WebSphere MQ can be applied to all platforms. Some of the unique properties of the various code bases are:

- IBM WebSphere MQ for z/OS has its own code base, with several administrative interfaces.
- Queue managers for the other WebSphere MQ implementations are all managed by using the same set of WebSphere MQ commands. Additionally, WebSphere MQ for Windows and for Linux have an Eclipse-based GUI which can be used to administer any queue managers on any MQ platform (except z/OS prior to Version 6). iSeries has, in addition, a set of CL commands that map to the WebSphere MQ commands.
- All implementations (apart from z/OS prior to Version 6) support PCF (programmable command format) messages, which facilitate programmed administration.

- IBM WebSphere MQ for Windows, AIX, iSeries, Linus, HP-UX, and Solaris support the WebSphere MQ Administration Interface (MQAI).

Installation

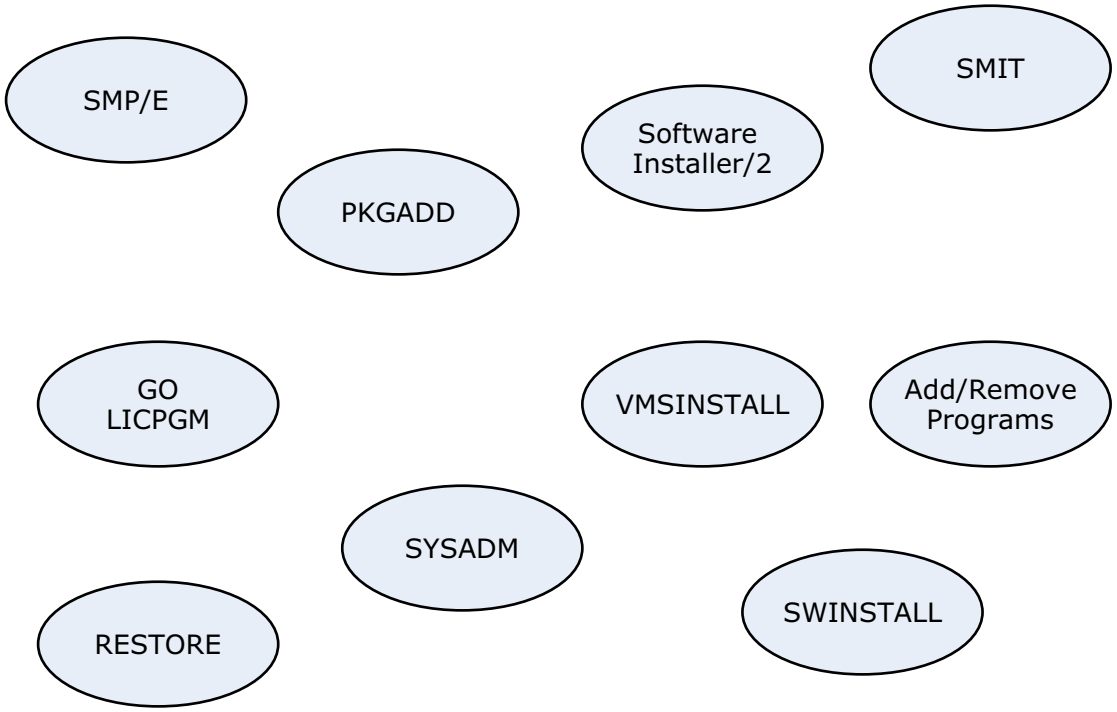


Figure 5-3. Installation WM100 / VM1001.0

Notes:

Installation on IBM WebSphere MQ uses the same mechanism used for installing most other software on the particular platform. This means that there is no major learning curve required to be able to work on a platform that you already know.

As with most software products, it is highly recommended that you apply any recommended maintenance as one of your installation steps.

Administrative tasks

- Install the IBM WebSphere MQ base code
- Create queue managers
- Operate the queue manager
- Manage the queue manager objects
- Manage logging and recovery including space requirements
- Manage adapters
- Manage channels
- Manage performance
- Manage security
- Manage problems

Figure 5-4. Administrative tasks WM100 / VM1001.0

Notes:

The above list of administration tasks can be divided into two sections. The first two items refer to the initial installation and setup, which is required only once. The remaining tasks are day-to-day operations that require a knowledgeable WebSphere MQ administrator. Many of the tasks listed would be done working with other groups: for instance, managing channels would require planning with the networking team; managing security would require planning with the security group.

WebSphere MQ administrative interfaces

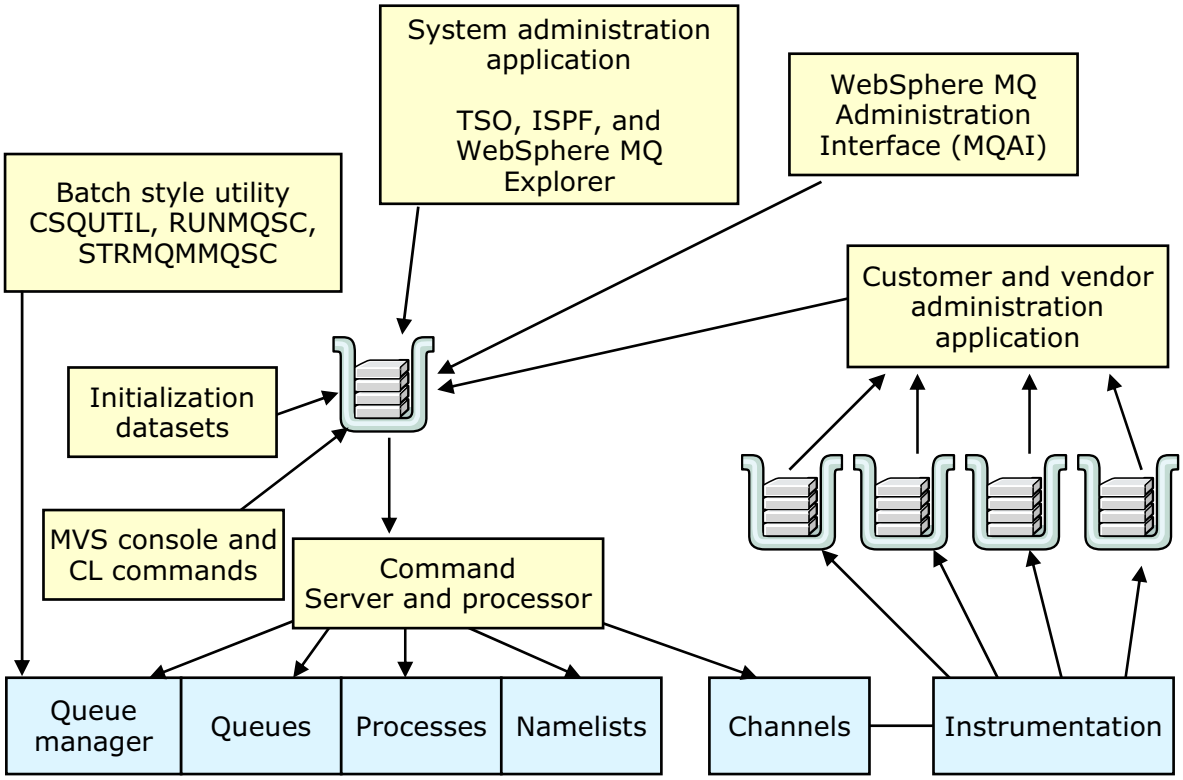


Figure 5-5. WebSphere MQ administrative interfaces WM100 / VM1001.0

Notes:

The figure depicts the administrative interfaces for IBM WebSphere MQ.

It is possible to manage IBM WebSphere MQ for a given system in several different ways, as you can see from the graphic. This offers great flexibility to the administrator. Notice that the commands are placed on a queue, and replies also use a queue. IBM WebSphere MQ uses its own facilities to do much of its work.

Messages containing WebSphere MQ commands can be put on the system command input queue and processed by the command server. Such messages may even originate on other systems. By sending control commands or programmable command format messages to another queue manager's command server, an administrator on the first system can look at or change queue manager objects on the second system.

The queues on the right side of the figure show something called *instrumentation* as input. These are events that may occur in the WebSphere MQ environment. One common type of event is that the starting or stopping of a channel occurred (a message is placed on the SYSTEM.ADMIN.CHANNEL.EVENT queue).

Other platforms may have additional events unique to them, such as:

- When a namelist is created on a z/OS queue manager, an event message is placed on the z/OS SYSTEM.ADMIN.CONFIG.EVENT queue.
- When a logging event occurs on a distributed queue manager, a message is placed SYSTEM.ADMIN.LOGGER.EVENT queue.

The use of that information (and even the removal of messages from these queues) is left to you. IBM WebSphere MQ does not provide programs to process these event messages.

WebSphere MQ Explorer

- Runs on Windows-XP and Linux platforms
- Integrated with Eclipse
- Maintains 'treeview + table' UI structure
- Many usability features, such as advanced filtering and a “compare with” function to identify differences between WebSphere MQ objects
- Merges MQ services into MQ Explorer
- Can remotely administer queue managers on all platforms, including z/OS

Figure 5-6. WebSphere MQ Explorer WM100 / VM1001.0

Notes:

IBM WebSphere MQ V7:

- Requires Windows XP, or Vista, or Windows 2003 x86-64
- IBM WebSphere MQ also requires Eclipse V3.3 which is included with IBM WebSphere MQ V7
- Linux based on Red Hat V4 or Suse Linux V9; AIX V5.3; z/OS 1.8

IBM WebSphere MQ V6:

- Requires Windows 2000, XP, or Vista, or Windows 2003
- IBM WebSphere MQ also requires Eclipse V3.0 which is included with IBM WebSphere MQ V6
- Linux based on Red Hat V3 or Suse Linux V8; AIX V5.3; z/OS 1.6

MQ Explorer — Queue Manager view

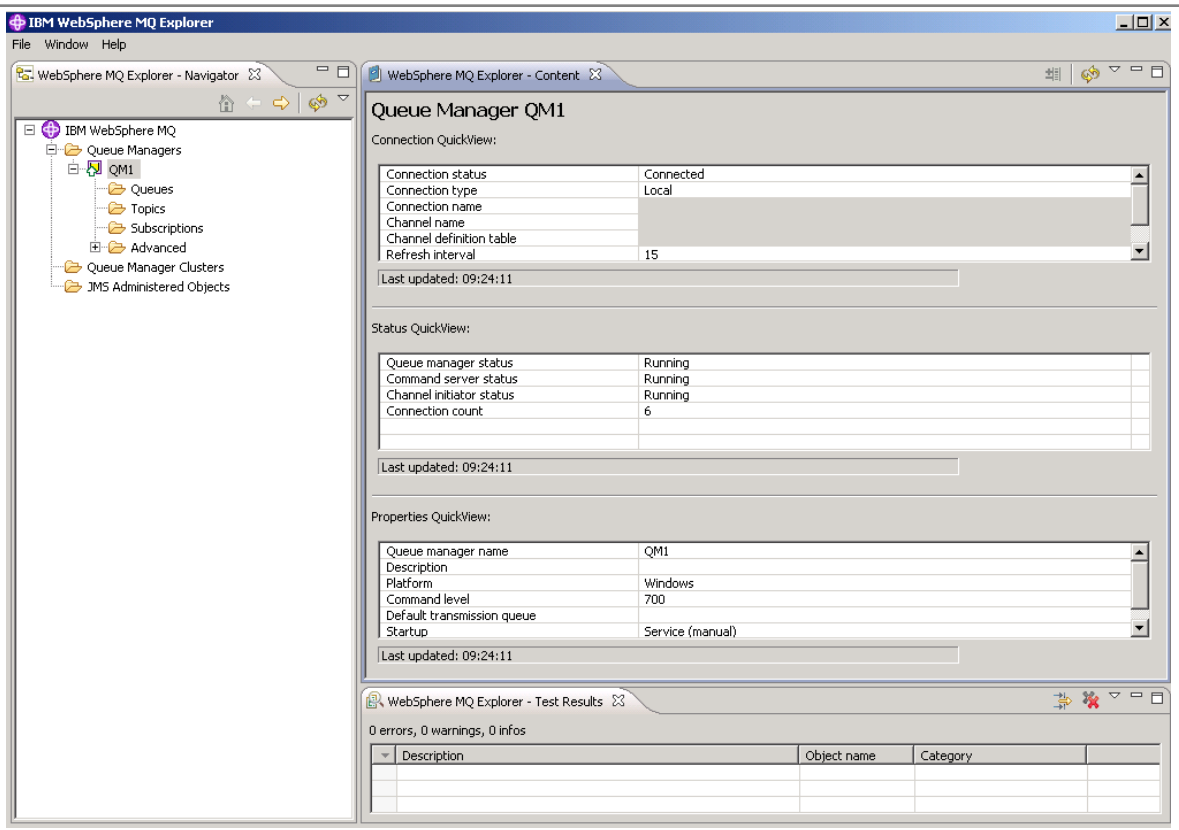


Figure 5-7. MQ Explorer — Queue Manager view WM100 / VM1001.0

Notes:

When a queue manager is selected in the Navigator view, the Content view displays three tables showing the status and properties of the queue manager.

WebSphere Explorer — Queues view

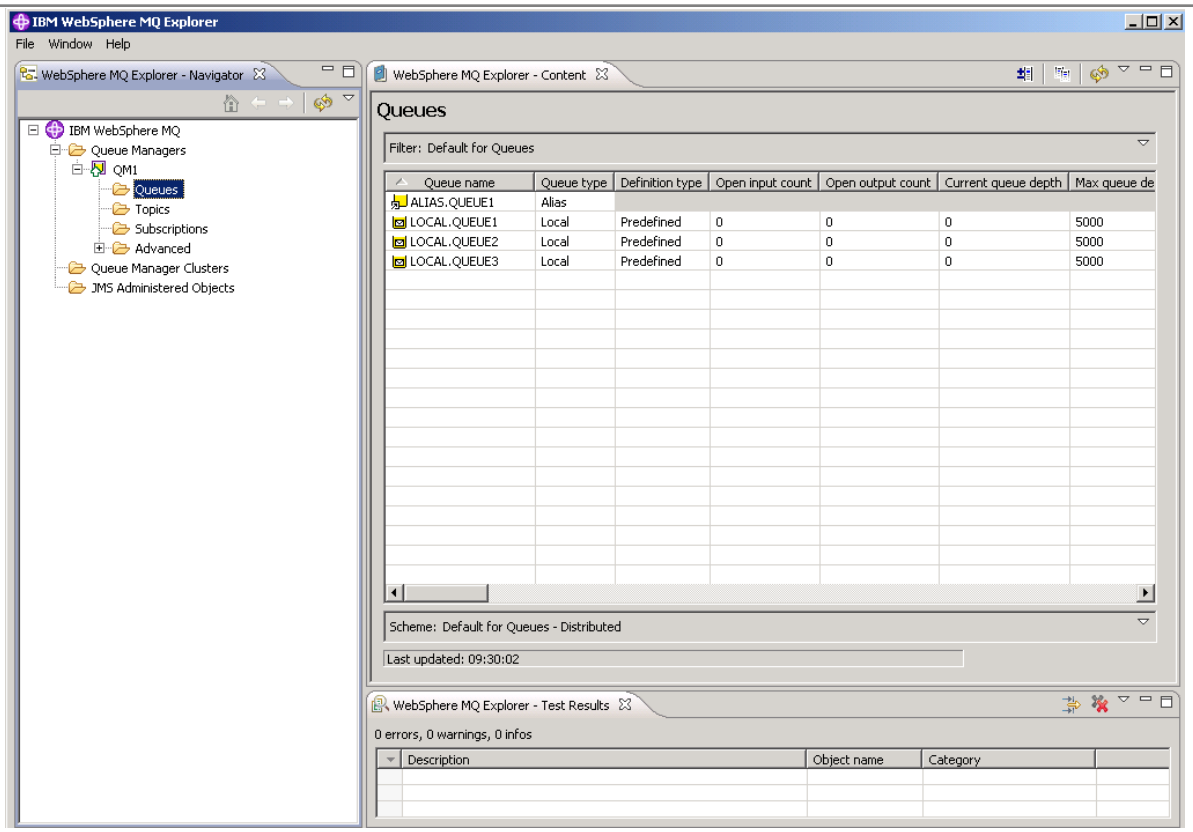


Figure 5-8. WebSphere Explorer — Queues view WM100 / VM1001.0

Notes:

When the Queues folder is selected (or the other object folders under a queue manager), the Content view shows one table listing those objects.

WebSphere MQ Explorer — queue selected

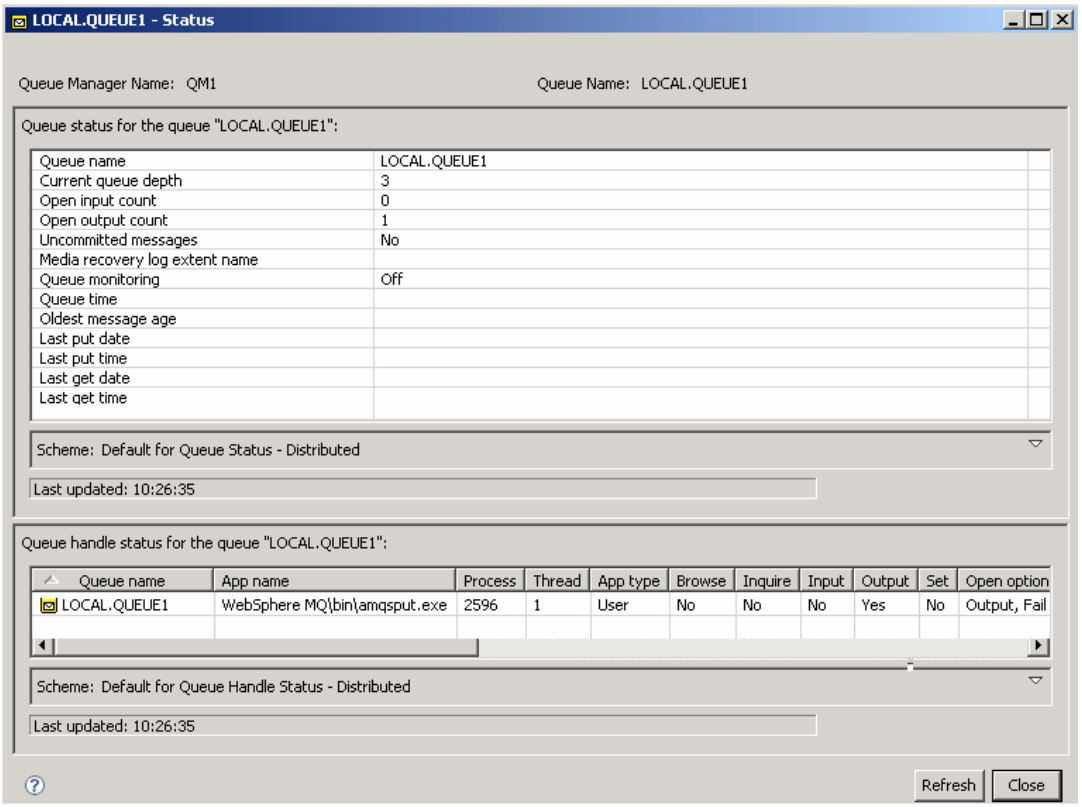


Figure 5-9. WebSphere MQ Explorer — queue selected WM100 / VM1001.0

Notes:

A context menu item on a queue in the table launches a dialog showing the status of that queue.

Status dialogs can also be shown for other objects:

- Queue manager
- Channel
- Service
- Listener
- Coupling facility

Explorer compare

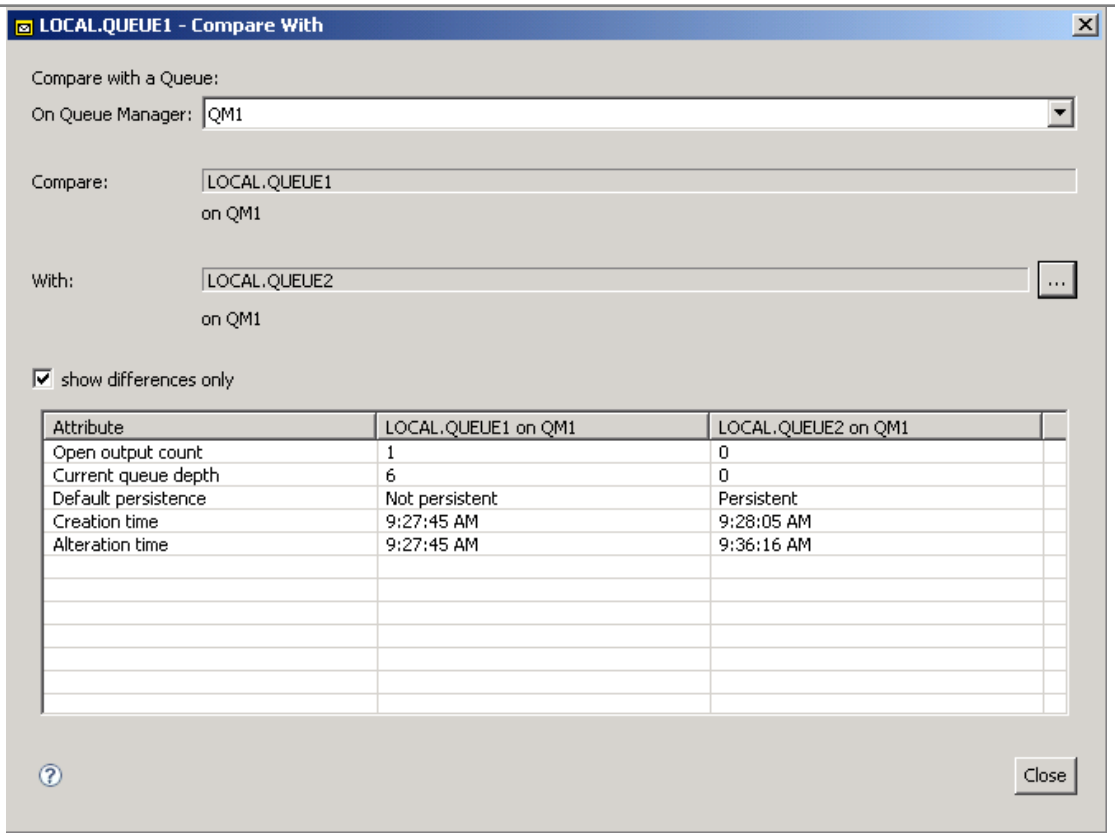


Figure 5-10. Explorer compare WM100 / VM1001.0

Notes:

Another context menu item on a queue in the table launches a dialog allowing the properties of the queue to be compared to those of another queue, which can be on the same or a different queue manager, even z/OS queues.

“Compare with” is available for most objects in Explorer.

MQ Explorer filtering

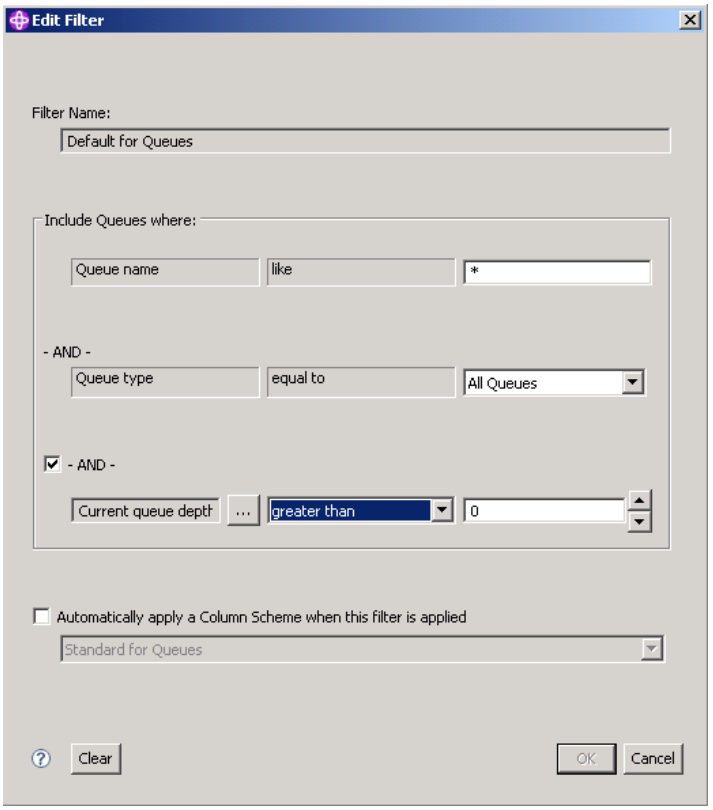


Figure 5-11. MQ Explorer filtering WM100 / VM1001.0

Notes:

The number of queues shown in the table can be reduced by using a programmable command format filter to select only those queues matching specified characteristics.

- The filtering is done at the remote queue manager to reduce network traffic.
- Explorer allows named filters to be created and persisted for each object type.
- Explorer also ships with a number of predefined filters.

WebSphere MQ Explorer filter results

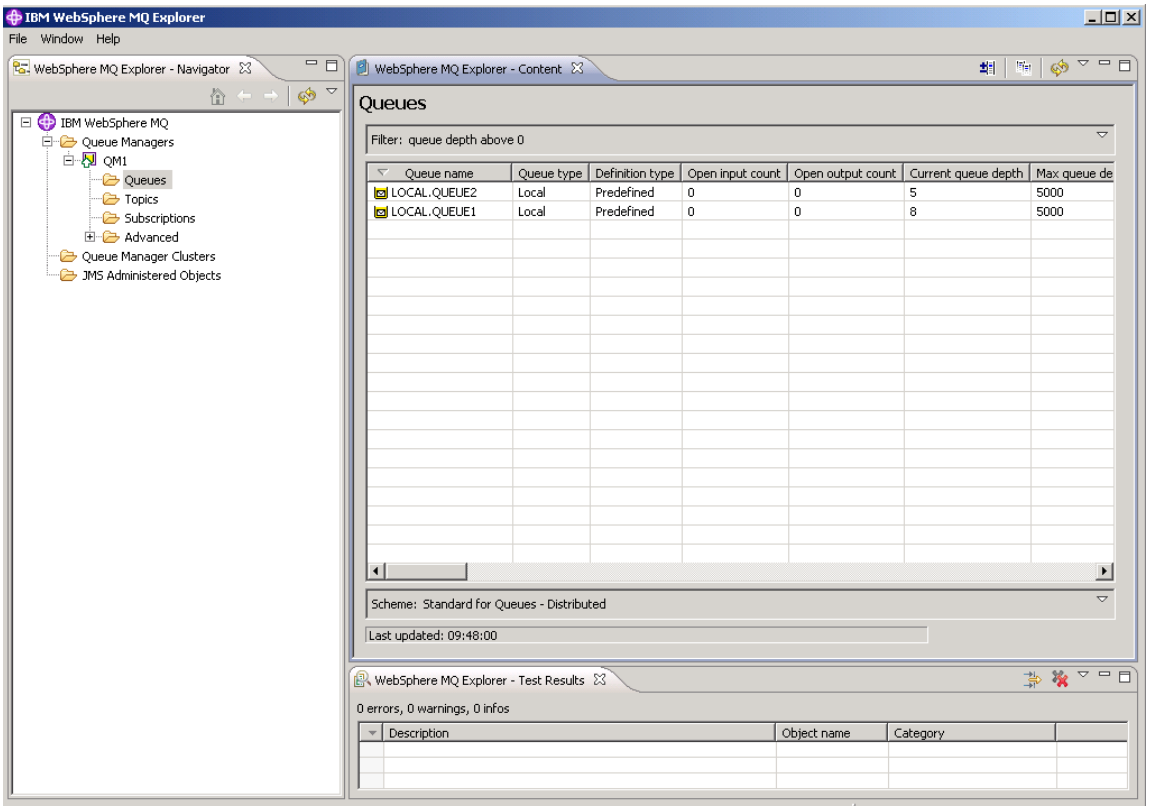


Figure 5-12. Filter result WM100 / VM1001.0

Notes:

Filtering allows the inclusion or exclusion of queue information. The sample filter was created to bypass the viewing of queues with a depth equal to zero.

5.2. WebSphere MQ on z/OS

Queue storage management

- Normally, four buffer pools in memory (max of 16)
  - By default, 1000 buffers of 4 KB each
- 100 page data sets
  - VSAM linear data sets
  - 4 KB pages
  - Up to 64 GB in size
- Storage classes
  - Associates local queue with page set
- Created by commands or utilities
- Objects stored in page set 00
- Page sets should be backed up regularly

Figure 5-13. Queue storage management WM100 / VM1001.0

Notes:

z/OS is a virtual storage operating system. It endeavors to keep the queues available in storage as far as possible, and uses *lazy write* to disk (asynchronously). The hardening function relies on the log.

You need to define buffer pools for the queues and align them with VSAM page data sets to which the messages are eventually put. You also need to back up the data sets regularly.

This hierarchy of queues to storage classes to page data sets to buffer pools is unique to IBM WebSphere MQ for z/OS system. So, some early planning regarding utilization of these resources is necessary.

Queue sharing

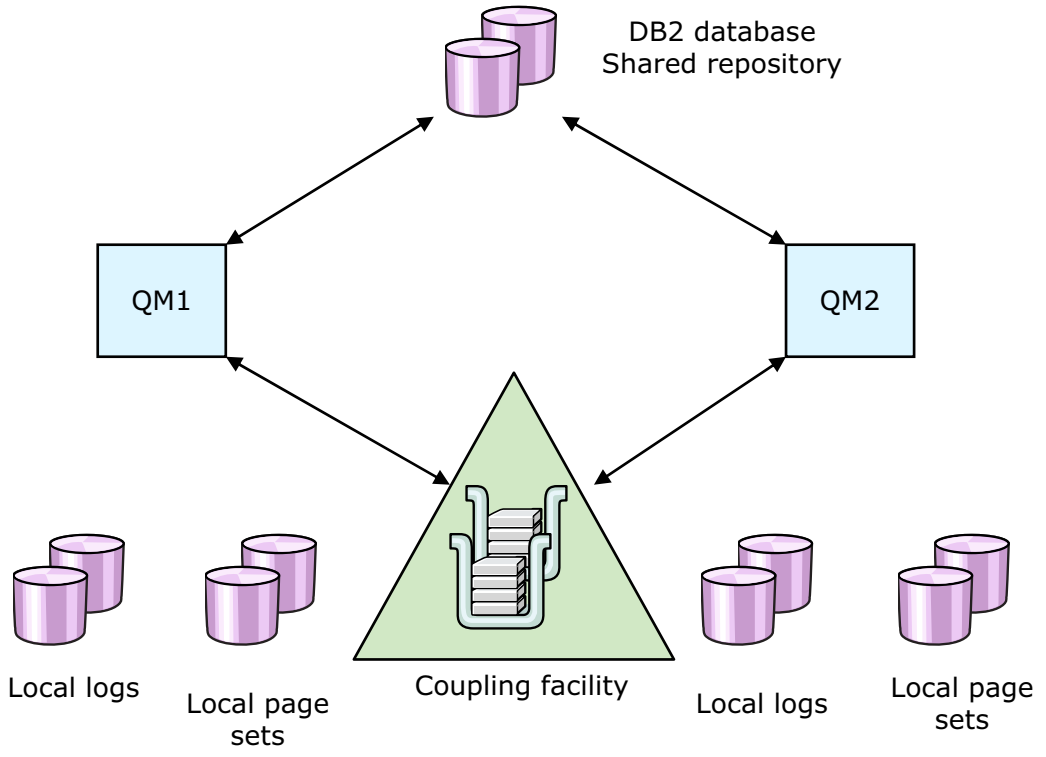


Figure 5-14. Queue sharing WM100 / VM1001.0

Notes:

IBM WebSphere MQ for z/OS can share queues between queue managers. This provides high availability of messages to any queue manager that can process the message.

This requires:

- A series of structures to be defined for the coupling facility
- A DB2 shared repository to be established to contain the definitions of the shared queues and messages greater then 63 KB in size

5.3. WebSphere MQ logging

The log and bootstrap data sets (z/OS)

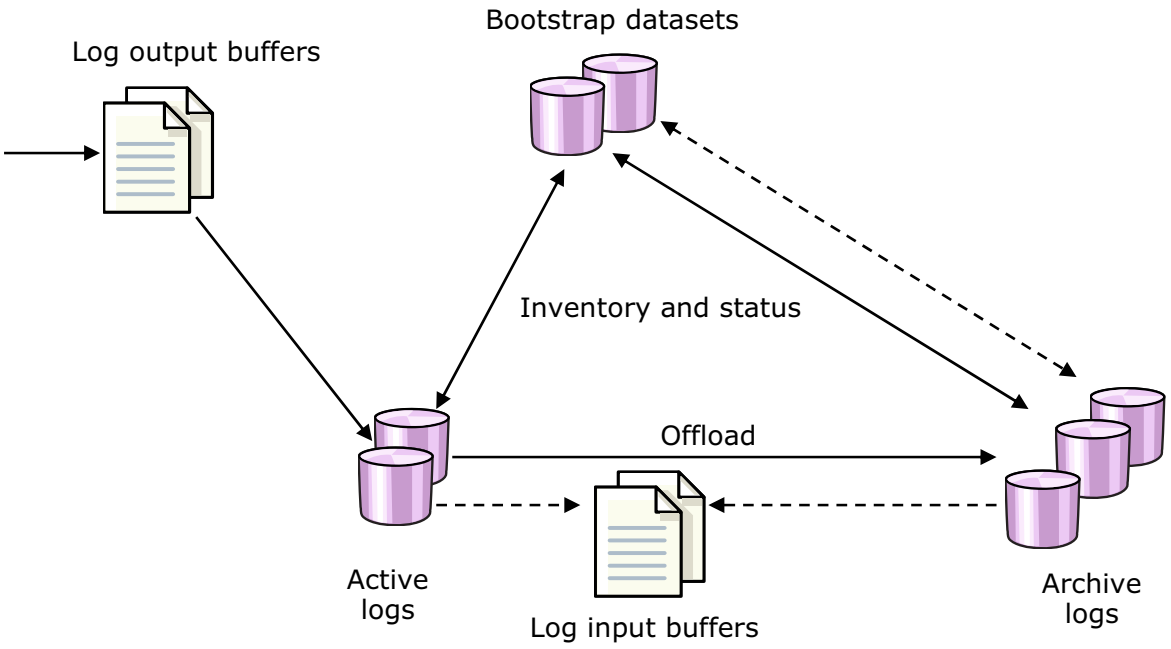


Figure 5-15. The log and bootstrap data sets (z/OS) WM100 / VM1001.0

Notes:

The log is a high-performance file accessed sequentially in journal mode to contain persistent messages and other updates that must survive failure. This may include media images of resources that need to be recovered after DASD failures.

The bootstrap data set records the current status of the log, including the presence of media images, checkpoint records, and the oldest unused message in the queues.

In a production environment, you should use dual logs and dual bootstrap data sets.

Journaling and recovery (iSeries)

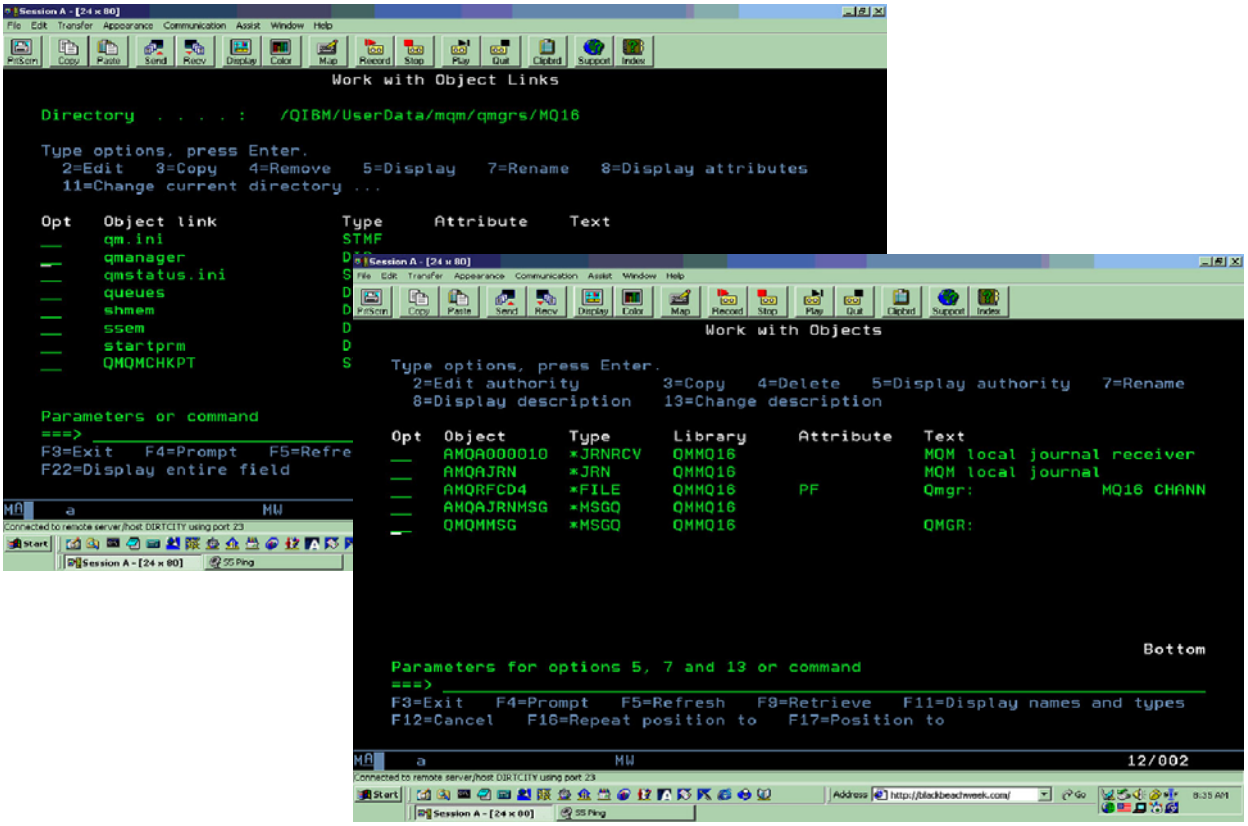


Figure 5-16. Journaling and recovery (iSeries) WM100 / VM1001.0

Notes:

IBM WebSphere MQ for iSeries uses OS/400 journaling support to help its backup and recovery strategy.

IBM WebSphere MQ for iSeries holds its data in an individual library for each queue manager, and in stream files in the integrated file system (IFS).

The queue manager specific libraries contain journals, journal receivers, and objects required to control the work management of a queue manager.

The IFS directories contain IBM WebSphere MQ configuration files, the descriptions of WebSphere MQ objects, and the data they contain.

For message queues, all persistent messages are recorded in the journals, while non-persistent messages are not.

Logging and recovery — UNIX and Windows

- Circular log
  - Crash recovery
  - Restart recovery
- Linear log
  - Crash recovery
  - Restart recovery
  - Media recovery
- Based on DB2 logging

Figure 5-17. Logging and recovery — UNIX and Windows WM100 / VM1001.0

Notes:

IBM WebSphere MQ for Windows and UNIX systems use logging to harden messages on a queue. By default, the log is circular. This means that it simply overwrites as it fills (unless uncommitted messages block it). This conserves space and reduces administration. However, if you desire logging where it is possible to offload to archives and recover from media failures, linear logging is an option. This decision must be made when a queue manager is created. It is not possible to alter the log strategy without re-creating the queue manager.

Checkpoint

Exercise — Unit 5 checkpoint

1. **T/F** IBM WebSphere MQ installation uses system-specific procedures for installing.
2. **T/F** The first queue manager created on a system will be the default.
3. Logging is used in which of the following queue managers? Select all that apply.
   a. IBM WebSphere MQ for z/OS
   b. IBM WebSphere MQ for HP-UX
   c. IBM WebSphere MQ for iSeries, UNIX, and Windows
   d. IBM WebSphere MQ for AIX
4. On IBM WebSphere MQ, which of the following interfaces are available for administration? Select all that apply.
   a. ISPF panels
   b. WebSphere MQ Explorer
   c. Management from a user or vendor supplied MQI client
   d. STRMQMMQSC

Unit summary

Having completed this unit, you should be able to:

- List the system administration interfaces for WebSphere MQ
- Discuss some of the tasks that a WebSphere MQ administrator is responsible for
- Explain the concepts of logging and recovery
- Describe some of the common administrative features of WebSphere MQ

Figure 5-18. Unit summary WM100 / VM1001.0

Notes:

By now, you should see that IBM WebSphere MQ uses facilities that probably are already familiar to you on your operating system.

While the products share a great deal of function across the various platforms, the appearance of IBM WebSphere MQ on each platform is designed to feel familiar to users of that platform.

Unit 6. Transactional support

What this unit is about

This unit describes the transactional support within IBM WebSphere MQ.

What you should be able to do

After completing this unit, you should be able to:

- Describe how WebSphere MQ acts as a resource manager
- Explain how and when WebSphere MQ acts as a transaction manager
- Describe the programming implications of unit of work processing

How you will check your progress

Accountability:

- Instructor questions
- Checkpoint questions

References

www.ibm.com/software/integration/wmq/library

IBM WebSphere MQ Library

SC34-6584 *WebSphere MQ System Administration Guide*

Unit objectives

After completing this unit, you should be able to:

- Describe how IBM WebSphere MQ acts as a resource manager
- Explain how and when IBM WebSphere MQ acts as a transaction manager
- Discuss programming implications of unit of work processing

Figure 6-1. Unit objectives WM100 / VM1001.0

Notes:

In this unit, you first examine the term *unit of work* and see what it means in IBM WebSphere MQ. IBM WebSphere MQ can coordinate its own resources and, in some cases, the resources of others. You look at these cases as well as consider how IBM WebSphere MQ resources can be coordinated with other resources using an external transaction manager.

6.1. Transactional support

Unit of work

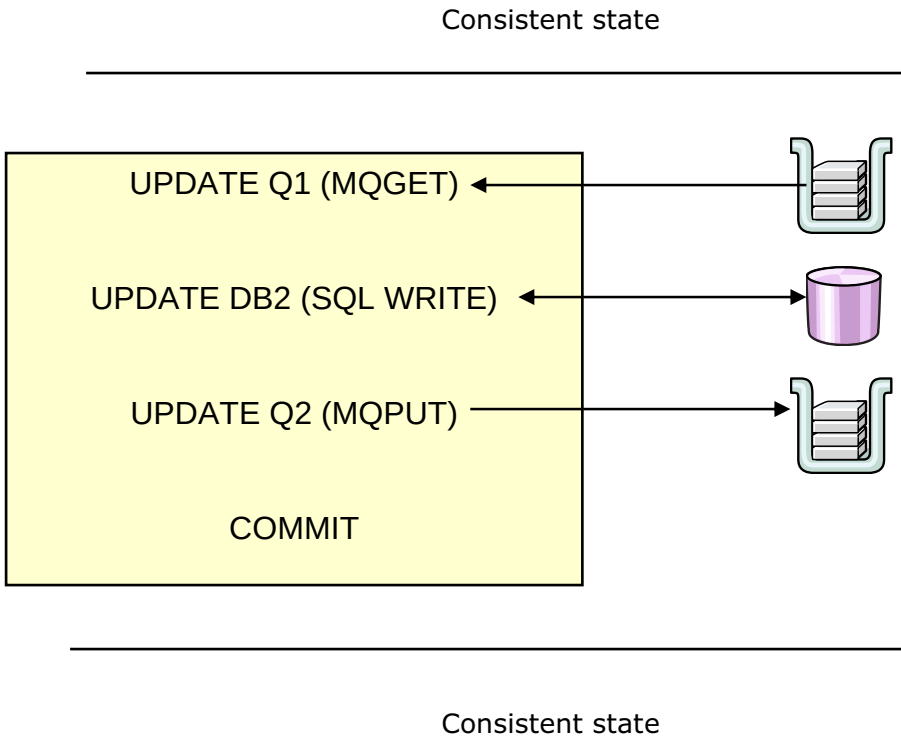


Figure 6-2. Unit of work WM100 / VM1001.0

Notes:

Within a *unit of work*, changes to resources are atomic. That is, either all of them take place and are *committed*, or none of them take place. There are no in-between states.

In the event of failure during a unit of work, or if the application determines that it cannot complete the unit of work for any reason, changes to resources that have already been made are *backed out*, or *rolled back*.

The point at which changes to the resources within a unit of work are committed or backed out is known as a *point of synchronization*, or more simply a *syncpoint*. At a syncpoint, the data within the resources is in a consistent state from the point of view of the business and its applications.

The visual shows changes to queues and a database, but other resources such as files and working storage might also be affected.

Resource manager

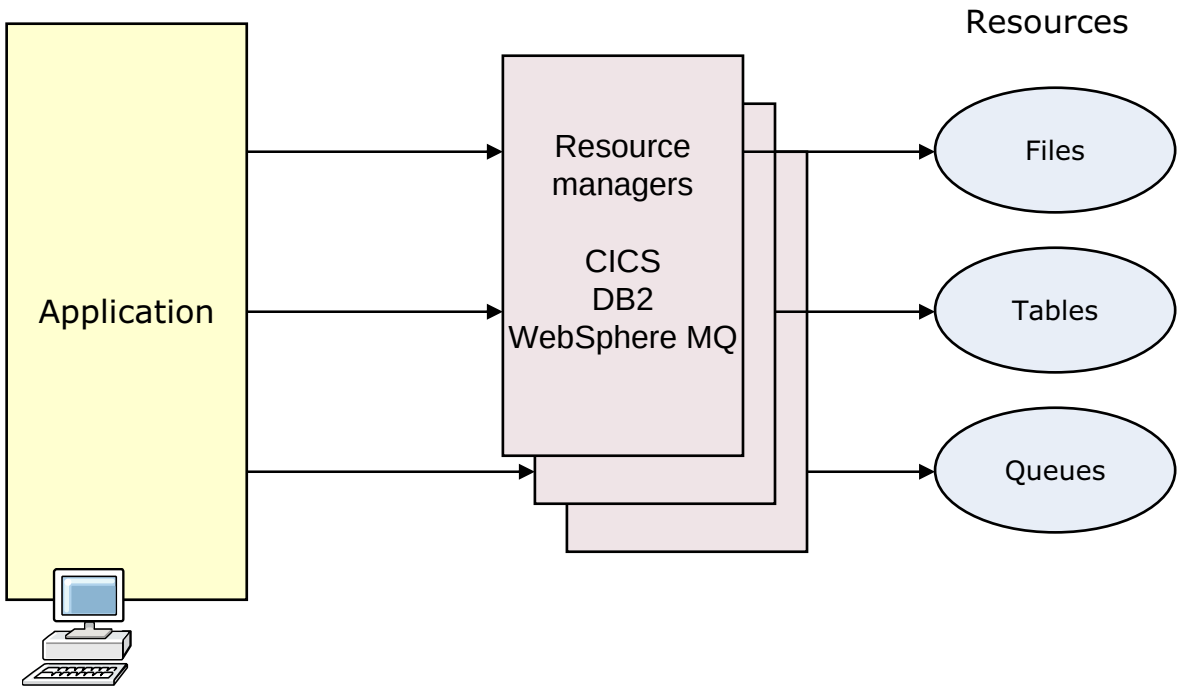


Figure 6-3. Resource manager WM100 / VM1001.0

Notes:

A *resource manager* is a generic term for a component of software that owns and manages resources such as files, databases, and queues. Typically, a resource manager provides:

- An API to allow an application access its resources
- Function to support the backup and recovery of its resources
- Function to support the creation and maintenance of its resources
- Function to support the commitment of changes to its resources within a unit of work, or the backing out of such changes
- Other functions as appropriate

Examples of resource managers include a CICS system managing a set of files under CICS file control, a DB2 database manager managing a set of tables, and a WebSphere MQ queue manager managing a set of queues.

An application may update resources belonging to more than one resource manager, as depicted in the figure. In this case, it would need to issue a separate commit request to each resource manager. Unless some application action is taken, this would destroy the atomicity of the unit of work because there is a window of opportunity for the application or

the system to fail between any two commit requests. The larger the number of resource managers, the more complex the application action would need to be.

Transaction manager

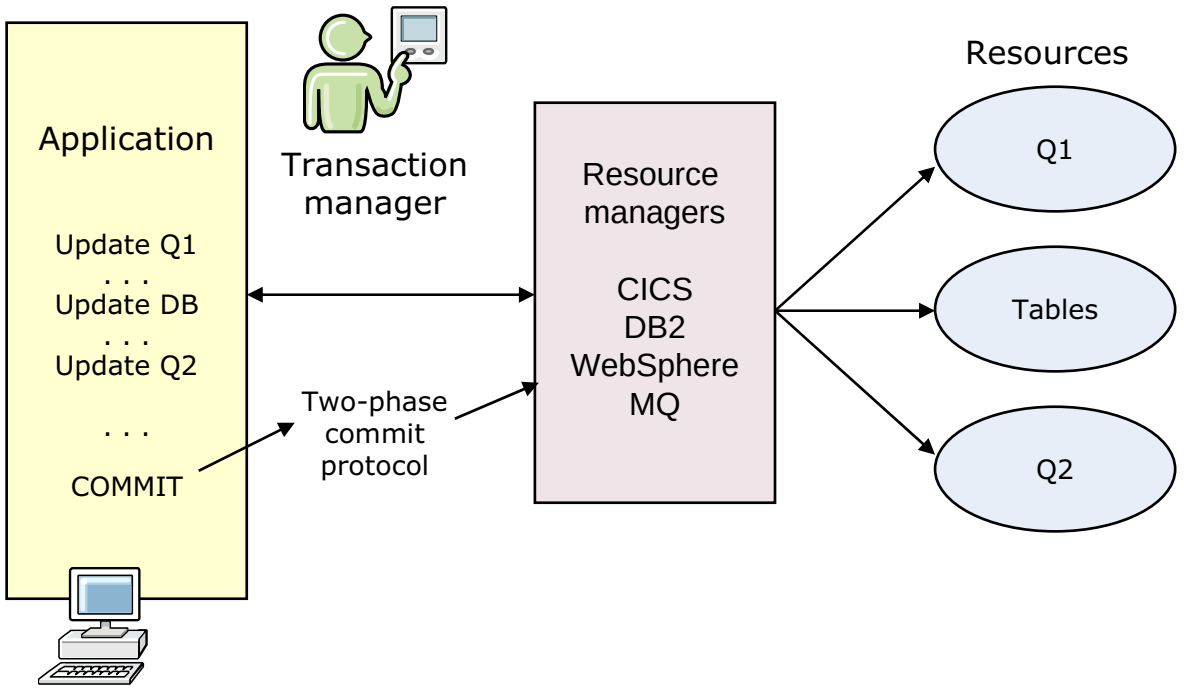


Figure 6-4. Transaction manager WM100 / VM1001.0

Notes:

A *transaction manager*, or *syncpoint coordinator*, is a component of software which can coordinate changes to the resources of multiple resource managers within a *single* unit of work as perceived by the application. The implication of this is that an application need only issue one request to commit the changes to all the resources even though multiple resource managers may be involved. In order to perform this function, a transaction manager uses a *two-phase commit protocol* in its interaction with the individual resource managers.

When an application issues a request to commit the changes it has made to the resources of multiple resource managers, the transaction manager asks each resource manager in turn to ensure that the changes to its resources are in a recoverable state, that is, they have been logged. When each resource manager has confirmed that this has been done, the first phase, known as the *prepare phase*, is complete.

When the prepare phase has completed successfully, the transaction manager makes the decision to commit the unit of work and the second phase, known as the *commit phase*, begins. This is now the point of no return. The transaction manager asks each resource

manager in turn to commit the changes to its resources. Even if there is a system failure during this phase, because the changes to the resources of each resource manager are in a recoverable state, commitment processing can recommence when the transaction manager and the resource managers are restarted.

Examples of transaction managers are CICS, IMS, Tuxedo, and Microsoft Transaction Manager.

The interface between a transaction manager and a resource manager needs to be standardized and documented if the aim is to make it easy to incorporate new resource managers as they become available. One such widely used interface is the X/Open XA interface, which uses a two-phase commit protocol. Transaction managers and resource managers that use this interface are said to be *XA compliant*.

MQGET within syncpoint control

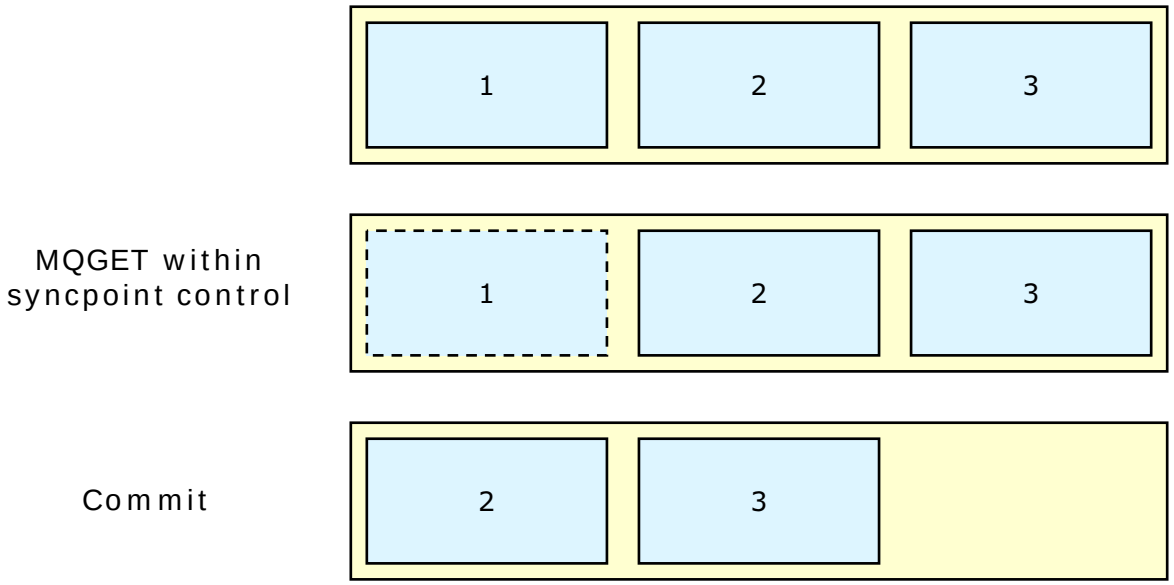


Figure 6-5. MQGET within syncpoint control WM100 / VM1001.0

Notes:

In a previous topic, the get message options structure was introduced. One of the fields in this structure, *Options,* allows an application to specify one of more options that control the action of the MQGET call. Two of these options are **within syncpoint control** and **outside of syncpoint control**.

When an application issues MQGET outside of syncpoint control, the message is removed from the queue immediately and the application is then wholly responsible for it. The message cannot be made available again by backing out a unit of work.

However, if MQGET is issued within syncpoint control, the message is not removed from the queue immediately, but it is made unavailable or invisible to other applications. Only when the unit of work is committed is the message actually removed from the queue.

MQPUT within syncpoint control

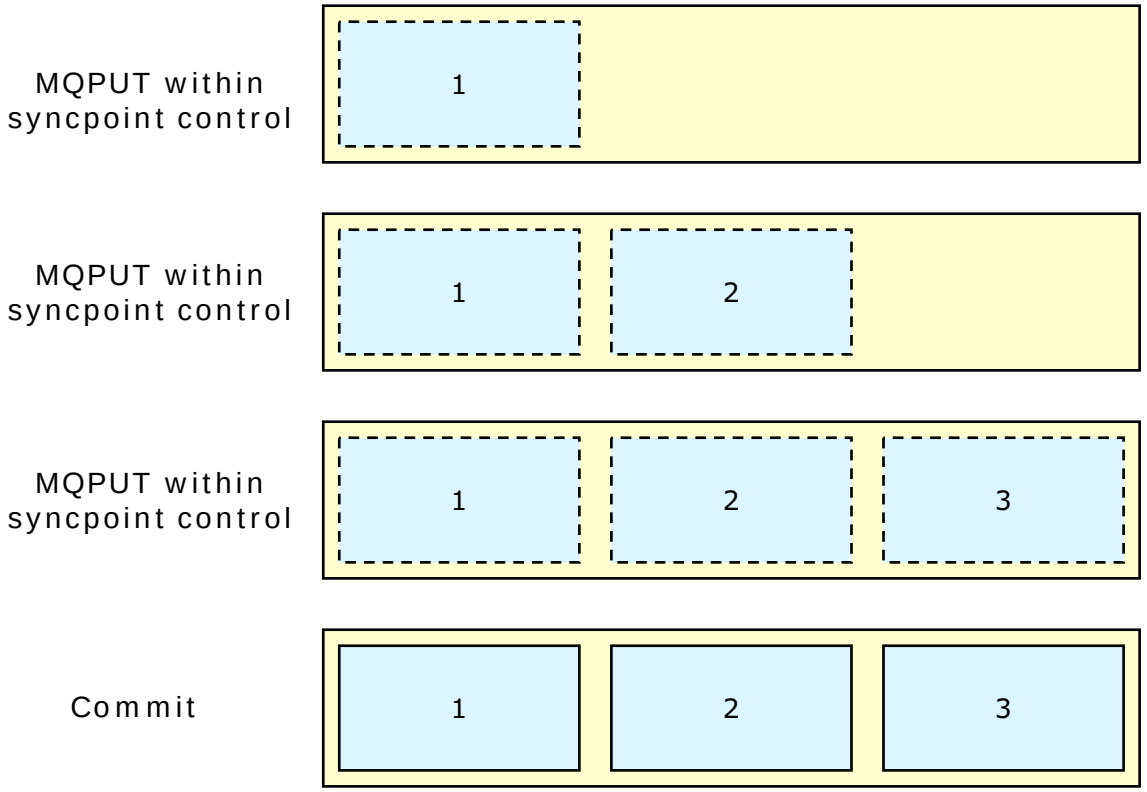


Figure 6-6. MQPUT within syncpoint control WM100 / VM1001.0

Notes:

Like the MQGET call, one of the parameters of the MQPUT call is the put message options structure. This structure also has a field called *Options* to allow an application to specify one or more options that control the action of MQPUT. Two of these options are **within syncpoint control** and **outside of syncpoint control.**

When an application issues an MQPUT call outside of syncpoint control, the message becomes available to other applications immediately and cannot be deleted by backing out a unit of work.

However, if MQPUT is issued within syncpoint control, the message is put on the queue, but remains unavailable or invisible to other applications until the unit of work is committed.

Coordinating local units of work

- A local unit of work is one in which the only resources being updated are those of the queue manager

```
MQGET  message from server queue
MQPUT  extra requests
MQPUT  reply message
  .
  .
  .
if error …
      MQBACK
if OK …
      MQCMIT
```


Figure 6-7. Coordinating local units of work WM100 / VM1001.0

Notes:

A *local* unit of work is one in which the only resources being updated are those of the queue manager to which the application is connected.

To support the coordination of local units of work, IBM WebSphere MQ provides two MQI calls, MQCMIT and MQBACK. The MQCMIT call commits changes to WebSphere MQ resources that have been made since the last syncpoint. The MQBACK call rolls back changes to IBM WebSphere MQ resources that have been made since the last syncpoint. Only changes to resources made under syncpoint control are affected by the MQCMIT and MQBACK calls.

The visual shows an example of a server program that gets a request message from a queue and puts a reply message on a reply-to queue within a local unit of work. This guards against the possibility of losing the request message if there is a system failure while the server program is processing it.

A WebSphere MQ client application may also use the MQCMIT and MQBACK calls.

Internal coordination of global units of work

- A global unit of work is one in which the resources of other resource managers are also being updated

```
MQBEGIN
MQGET  message from server queue
EXEC SQL INSERT data base record
MQPUT  reply message
  .
  .
  .
if error …
      MQBACK
if OK …
      MQCMIT
```


Figure 6-8. Internal coordination of global units of work WM100 / VM1001.0

Notes:

A *global* unit of work is one in which the resources of other resource managers are being updated as well as those of a queue manager.

The coordination of global units of work may be *internal* or *external* to the queue manager. You will look at the external coordination of global units of work a little later. On this visual, you focus on internal coordination.

Using the X/Open XA interface, a queue manager is able to coordinate changes to its own resources and to those of other resource managers within a unit of work. An external syncpoint coordinator is not required under these circumstances.

The visual depicts an example of a global unit of work in which changes are made to WebSphere MQ resources and to those of a relational database within a unit of work. The MQCMIT and MQBACK calls are used to commit and roll back changes, as with local units of work. However, when coordinating global units of work, the MQBEGIN call is needed in order to start a unit of work.

Database coordination

- The platforms where the following database manager exist are supported:
  - DB2 Universal Database
  - Informix Dynamic Server
  - Oracle
  - Sybase
- Restrictions
  - A WebSphere MQ client cannot participate in a global unit of work, unless it is the extended client version
  - Only one queue manager may participate in a global unit of work
  - Normally, updates to WebSphere MQ and database resources must be made on the same system
  - However, a database server may be on a different system, provided it can supply an XA-compliant client feature

Figure 6-9. Database coordination WM100 / VM1001.0

Notes:

The visual depicts the XA-compliant database managers that are supported by IBM WebSphere MQ. These database managers may participate in a global unit of work coordinated by a WebSphere MQ queue manager.

The visual also lists some restrictions regarding the internal coordination of global units of work.

- A WebSphere MQ client application cannot participate in a global unit of work and cannot therefore issue the MQBEGIN call.
- Although a queue manager may be XA-compliant, both as a syncpoint coordinator and as a resource manager, it is not possible to configure two or more queue managers as participants in a global unit of work. This is because an application may only be connected to one queue manager at a time.
- Normally, updates to WebSphere MQ resources and to those of a database manager must be made on the same system. IBM WebSphere MQ does not have the capability to coordinate a distributed unit of work.

- However, a database manager may reside on a different system than the queue manager, provided it can supply an XA-compliant client feature that resides on the same system as the queue manager.

WebSphere MQ for z/OS RRS support

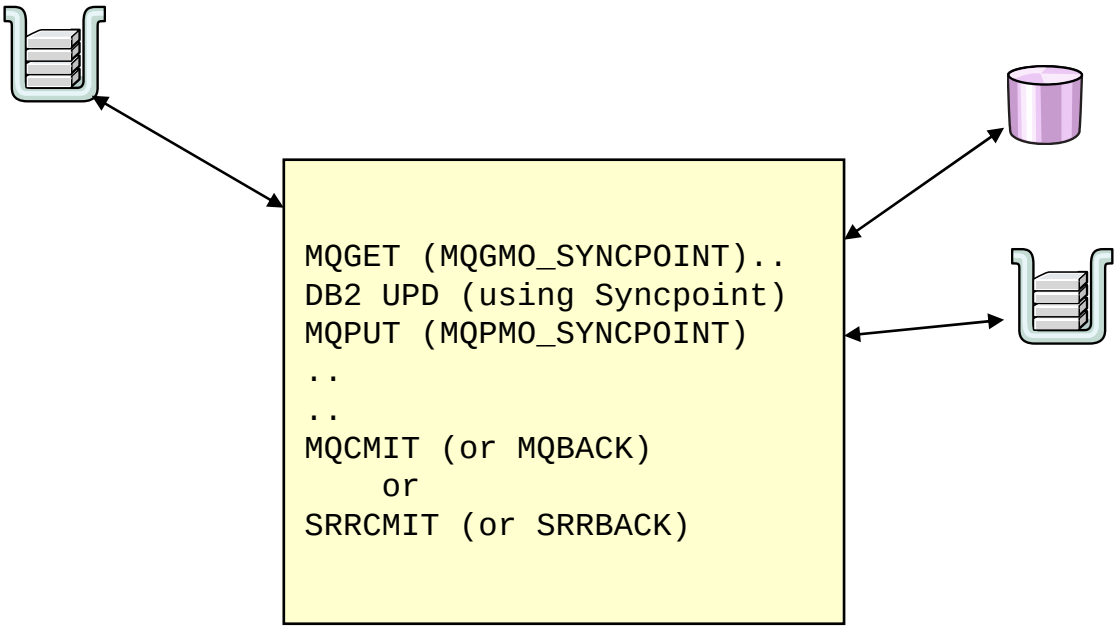


Figure 6-10. WebSphere MQ for z/OS RRS support WM100 / VM1001.0

Notes:

In addition to the two-phase commit capabilities available in the CICS and IMS environments, with the announcement of WebSphere MQ for z/OS V2R1 and higher, WebSphere MQ batch programs can participate in units of recovery managed by z/OS Recoverable Resource Manager Services (RRS).

MQPUTs and MQGETs would still be used to specify the syncpoint option desired. It is even possible to use the MQCMIT and MQBACK verbs. The alternative is SRRCMIT and SRRBACK. Linking with a different stub enables this choice. Single-phase (non-RRS) processing is also still available.

Checkpoint

Exercise — Unit 6 checkpoint

1. **T/F** Within a *unit of work*, changes are atomic.
2. An MQPUT within syncpoint control means:
   a. Messages are not placed on a queue until the commit.
   b. Messages are placed on the queue and the “getting” program needs to check the syncpoint flag.
   c. Messages are placed on the queue but are not available to the getting program until after the commit.
3. **T/F** Standard WebSphere MQ clients can use global unit of work processing.
4. Global units of work are supported for Oracle on which of the following? Select all that apply.
   a. WebSphere MQ for HP-UX
   b. WebSphere MQ for Windows
   c. WebSphere MQ for AIX
   d. WebSphere MQ for Linux(x86)
   e. WebSphere MQ for Solaris

Unit objectives

After completing this unit, you should be able to:

- Describe how IBM WebSphere MQ acts as a resource manager
- Explain how and when IBM WebSphere MQ acts as a transaction manager
- Discuss programming implications of unit of work processing

Figure 6-11. Unit summary WM100 / VM1001.0

Notes:

Unit of work processing allows control over delivery of messages until a time when the application is satisfied that related processing has been successful. In addition, it allows for backing out of any updates to queues in case of errors.

WebSphere MQ, in addition to using MQCMIT and MQBACK to coordinate its own resources, can take part in a unit of work that is coordinated among several resource managers under control of a transaction manager. A queue manager can also, sometimes, be a transaction manager of sorts.

Unit of work processing should be an important consideration in the application design process.

Unit 7. Security

What this unit is about

This unit describes the security aspects of the IBM WebSphere MQ platforms.

What you should be able to do

After completing this unit, you should be able to:

- Describe security features of WebSphere MQ platforms
- Determine which security features are appropriate in a given environment
- Explain the role of WebSphere MQ security within and across enterprises

How you will check your progress

Accountability:

- Instructor questions
- Checkpoint questions

References

www.ibm.com/software/integration/wmq/library

IBM WebSphere MQ Library

SC34-6588 *WebSphere MQ Security*

Unit objectives

After completing this unit, you should be able to:

- Describe security features of IBM WebSphere MQ platforms
- Consider which features are appropriate in a given environment
- Explain the role of IBM WebSphere MQ regarding security within and across enterprises

Figure 7-1. Unit objectives WM100 / VM1001.0

Notes:

IBM WebSphere MQ is a resource manager but does not claim to be a security manager. However, security is important in an enterprise and IBM WebSphere MQ certainly participates. It will rely on IBM WebSphere MQ skills.

The purpose of this unit is to identify the various aspects of the IBM WebSphere MQ product that may participate in the overall security management of your installation.

7.1. Security

IBM security architecture

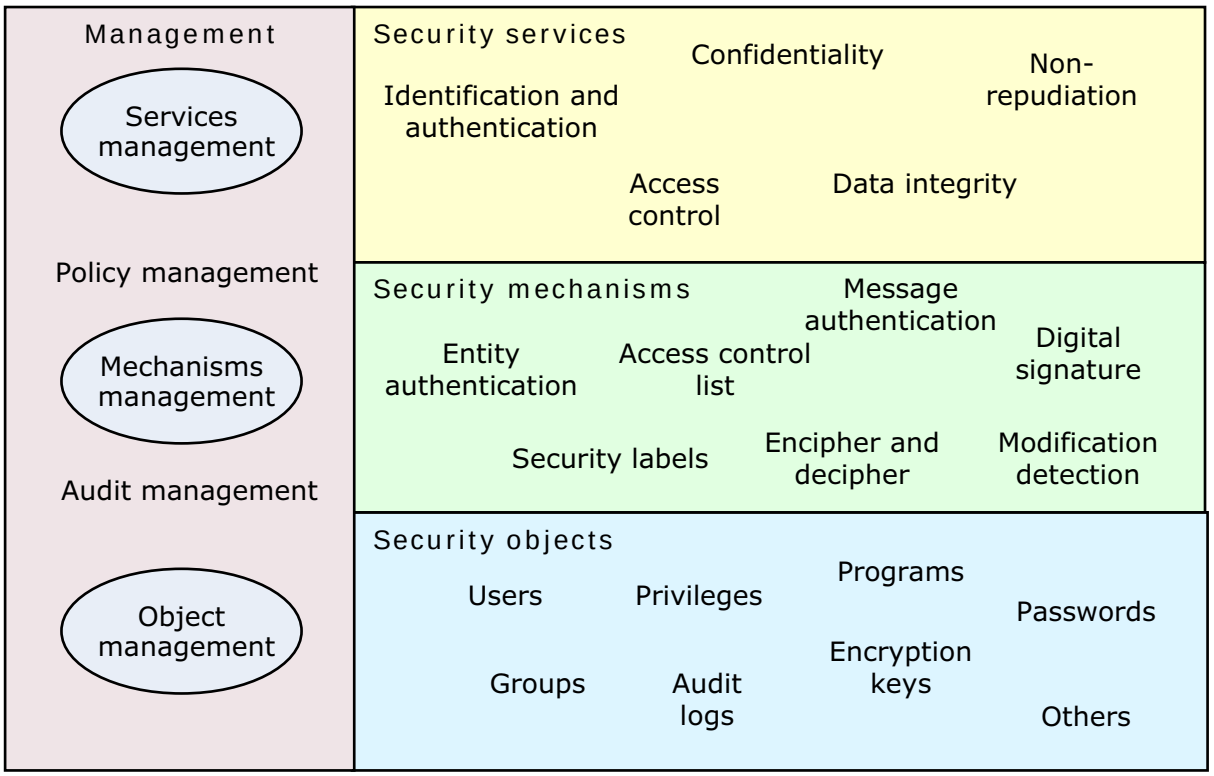


Figure 7-2. IBM security architecture WM100 / VM1001.0

Notes:

Security services are those functions that, when provided in a systems environment, serve to ensure the protection of resources by enforcing the defined security policies of the organization. Security mechanisms are technical tools and techniques used to implement the security services. Mechanisms may operate individually, or in concert with others, to provide a particular service. The security objects consist of two components: managed objects and managing objects. Managed objects describe what is managed and how it behaves. The definition of managed objects includes specification of their attributes and their behavior, which provides a concrete description of what is manageable. The how of management is defined by managing objects consisting of applications and data, which support the management and use of the rest of the system.

Security management is the administration, control, and review of an enterprise security policy. Security managers make use of procedures and system security services to implement policies consistent with the organization objectives. System auditability can

provide checks and balances on system users and administrators to ensure that security management policies are enforced.

Security implementation

- Mechanisms required at various levels
  - Application code and exits
  - Product code
  - Environment code
  - Network code

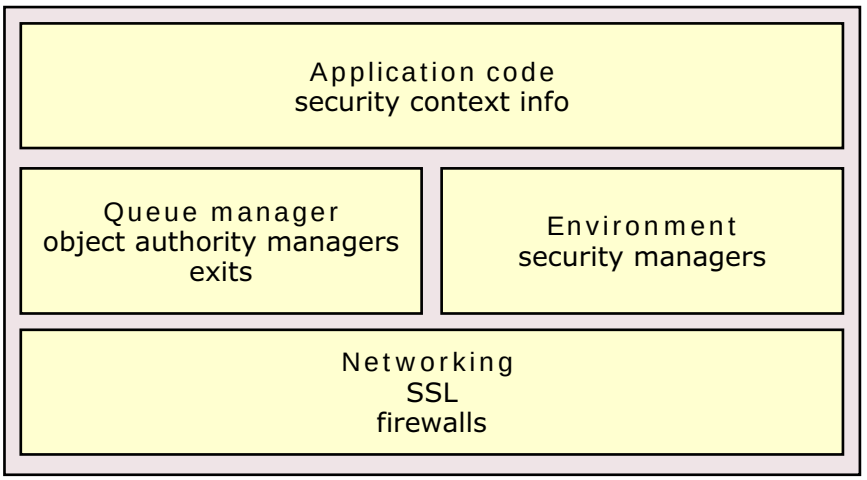


Figure 7-3. Security implementation WM100 / VM1001.0

Notes:

Although there are mechanisms in the surrounding environment, in the operating systems, networks, and transaction managers, you are most interested in the facilities provided by the queue managers and in applications that they support.

The queue managers allow for complementary security services but rely on the environment in which they are running to provide the mechanisms required. For example, on z/OS, WebSphere MQ queue managers use the Resource Access Control Facility (RACF) or equivalent product to provide access control for queue manager resources (queues, commands, and so forth).

Security in WebSphere MQ

- Access control
  - Resources
  - Commands
- Message context
  - Carried with message
- Exit facilities
  - Message channel agent
    - Security
    - Message
    - Send and receive
  - Object authority managers
    - Internal
    - External
- API
- SSL

Figure 7-4. Security in WebSphere MQ WM100 / VM1001.0

Notes:

IBM WebSphere MQ provides:

- **Access control** — check when accessing queue manager resources and commands against the user ID under which the application program is running. z/OS uses an external security manager as RACF, iSeries system security for OS/400, or OAM for Windows. Access control can include the following:
  - Checking the user ID when the MQCONN, MQOPEN, MQPUT1, and MQCLOSE calls are issued
  - Checking access to commands as they are issued to determine if the issuing user has appropriate permissions
  - Additional checks when commands are issued to determine whether the issuing user has appropriate permissions for the actual resource the command is being directed to

- **Message context** — Contextual information contained within the MQ message descriptor of each message, which describes who generated the original request and where this specific message came from.
- **MCA exits** — Exit programs can be attached to the message channel agents. This exit facility has been designed to allow security-related functions to be implemented in the exit routines associated with links between systems. Today, some exit programs are supplied with some IBM WebSphere MQ products.
- **Secure Sockets Layer (SSL) —** Secure Sockets Layer provides channel security, to the user, as is. It is an industry standard protocol for transmitting data in a secure manner over an insecure network. SSL is widely deployed in both Internet and intranet applications. IBM WebSphere MQ support for SSL enables you to specify, on the channel definition, that a particular channel uses SSL security.
- **API-crossing exit** — Invoked before and after every MQI call. Available only for CICS applications under z/OS. Using the API-crossing exit can degrade WebSphere MQ performance, so its use needs to be planned carefully.
- **Security management and audit** — Facilities that allow system administrators to set up and manage the various security operations, and to verify that the security facilities are working in the expected manner.

Resource access security

- Queue security
  - User IDs
  - Options
- Process security
- Namelist security
- Context security
  - MQOPEN, MQPUT1
- Alternate user security

Figure 7-5. Resource access security WM100 / VM1001.0

Notes:

Security involves the protection of the queue manager resources from unauthorized access. In addition to checking that the user is able to connect to the queue manager, checks can be done to ensure that the user is only requesting options that are authorized.

For instance, an application might issue an MQOPEN for output and the user is permitted to do that. However, if the option is changed to open the queue for output and to set context fields (those containing user ID information), the call may fail.

In addition to wanting to change the contents of the context fields in an individual message, a program may wish to make use of user ID context information from an incoming message. Both of these are protected.

The namelist and process objects can be opened by an application for inquiry. This can also be checked to determine whether the user is permitted.

Command security

- Authorization checked when command is issued
- Based on user ID
  - Command security
    - May this user issue this command?
      - for example, **`DISPLAY QUEUE`**
  - Command resource security
    - May this user change this resource?
      - for example, **`DELETE QLOCAL('Payroll')`**

Figure 7-6. Command security WM100 / VM1001.0

Notes:

As mentioned previously, checks can be performed to determine whether a user is permitted to issue certain commands. In addition, the check can also be whether the user is authorized to issue the command against a specific resource.

Message context

- Travels with the message
  - In message descriptor
- Two parts
  - Identity context
    - User ID
    - Accounting token
    - Application identity data
  - Origin context
    - Application name
    - Type
    - Date and time
    - Application origin data
- Allows system authorization by ID
- Provides accounting structure
- Allows application authorization

Figure 7-7. Message context WM100 / VM1001.0

Notes:

The fields shown in the figure are fairly self-explanatory. The first three fields are treated as a group called *identity context*, while the remaining fields comprise another group known as *origin context*.

This allows not only a user to be authenticated and authorized, but the application that actually performed the MQPUT can be checked as well.

Unless specifically permitted and opened with the options to update them, an application will not be able to update these fields. The queue manager fills in the information when the MQPUT is executed.

User IDs

- Operating system user ID
- Address-space identifier
- Console ID
- Access violations logged to
  - Environment log
  - QMgr log
  - QMgr event queue

Figure 7-8. User IDs WM100 / VM1001.0

Notes:

Many systems operate under user IDs, often associated with passwords. Those that do not may have some comparable identifier, such as an ID associated with an address space in z/OS or a Windows or UNIX process. These may be used to check for authorized access.

It is possible that an attempt to access a resource may result in a message being delivered to the security manager log. It is also possible that such a message is placed in the queue manager logs. If enabled, the unauthorized access may also be recorded as a message on a system queue.

Passing context information

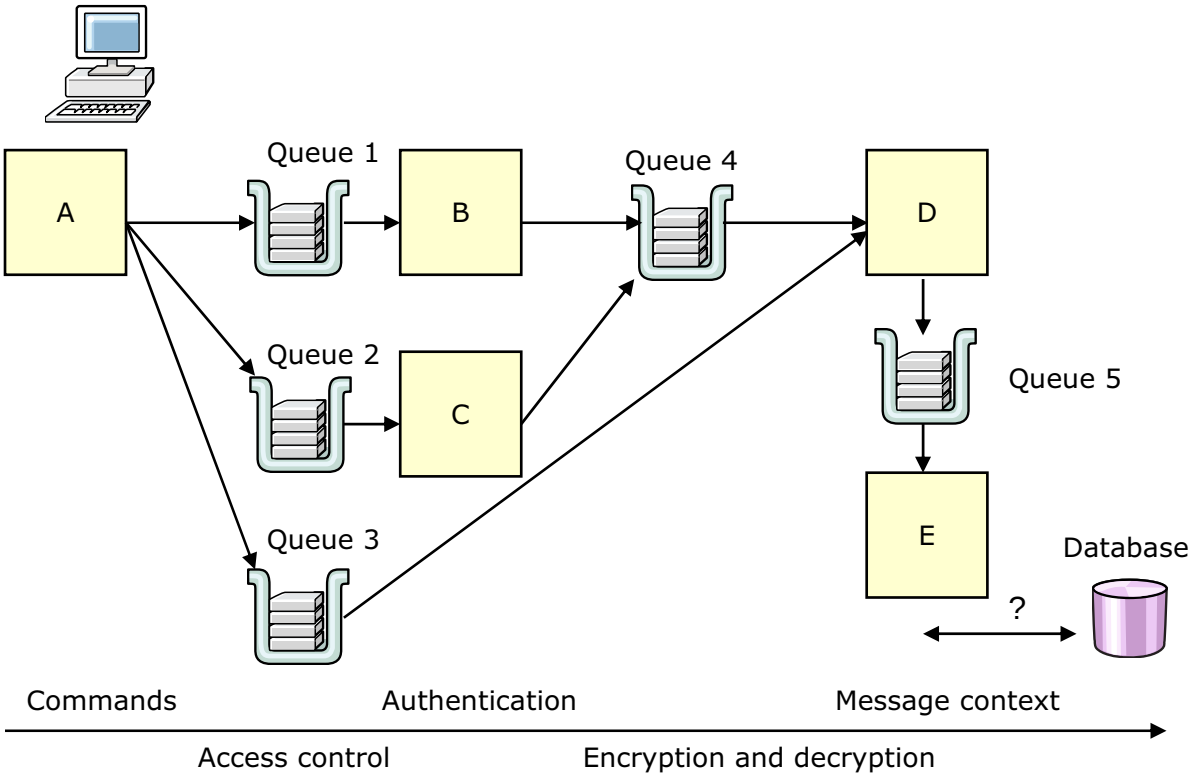


Figure 7-9. Passing context information WM100 / VM1001.0

Notes:

In this example, messages have split, been processed by different systems, and then rejoined for a request to be sent to the database server.

Under whose authority should the final database calls be made?

It seems that the user ID that starts application D is the one checked for authority to open the **database update queue** for output. However, it is possible to pass the initial user ID all the way from the user that started application A and issued MQPUTs to the three output queues. This user can then be checked to determine if he or she is authorized to perform database updates.

Notice the continuum across the bottom. Commands are controlled at a local level, as is access control in programs (MQOPEN, for instance). When the data starts to flow across networks, the use of security exits can ensure that the MCAs on each side of a channel establish trust that they are connecting to the proper **partner**. As data flows, additional exits can be used for encryption and decryption to protect messages that contain sensitive information. Finally, the context information can aid in assuring that only authorized users obtain access to critical resources.

WebSphere MQ channel security

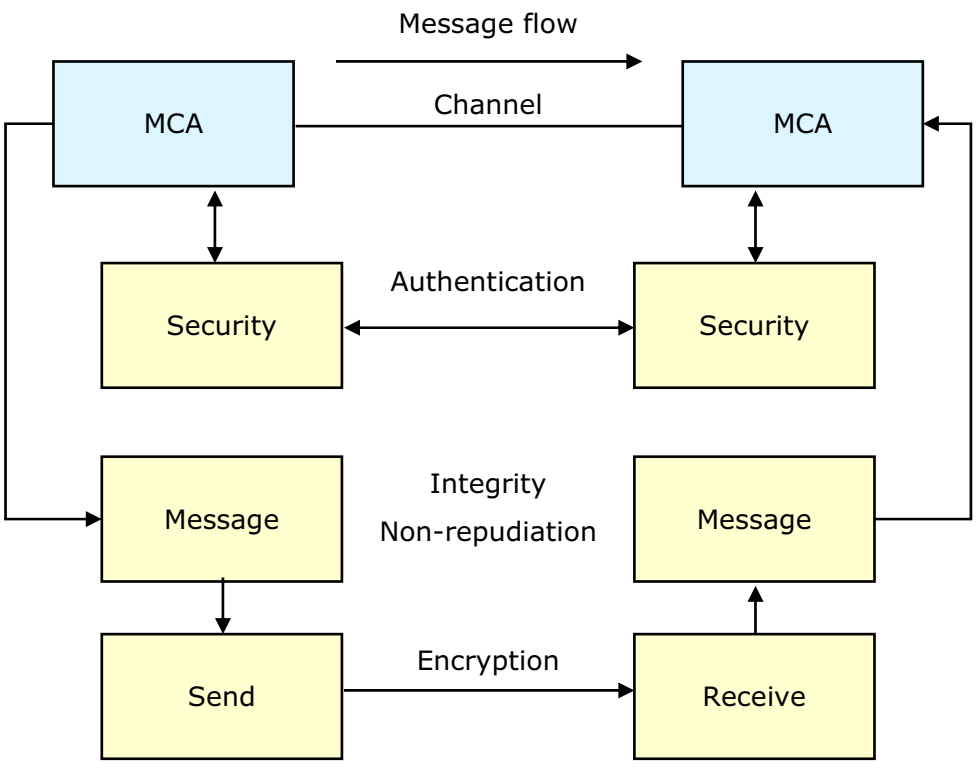


Figure 7-10. WebSphere MQ channel security WM100 / VM1001.0

Notes:

The uses of message channel agent exits are:

Security exit — is primarily used by the MCA at each end of a message channel to authenticate its partner.

Send and receive exits — can be used for purposes such as data compression and decompression, or data encryption and decryption.

Message exit — can be used for any purpose that makes sense at the message level. The following are some examples:

- Application data conversion
- Encryption and decryption
- Additional security checks, such as validating an incoming user identifier
- Substitution of one user identifier for another as a message enters a new security domain

Secure Sockets Layer (SSL)

- Protocol to allow transmission of secure data over an insecure network
  - Encryption, integrity, authorization
  - Protection
    - Client/server
    - Qmgr and QMgr channels
- To combat security problems
  - Eavesdropping
  - Tampering
  - Impersonation

Figure 7-11. Secure Socket Layer (SSL) WM100 / VM1001.0

Notes:

Secure Sockets Layer (SSL) is common across all the platforms: z/OS, UNIX, Windows, and iSeries.

SSL is an industry-standard protocol for secure communications, involving encryption, authentication, and integrity of data. SSL is supported in both client/server and queue manager-to-queue manager channels (including queue manager clusters). There are many flexible capabilities built in, including the ability to select who they are prepared to accept communications from, based on their fully authenticated identity. This removes, for many people, the need to set up channel exits, where they were used for security purposes.

SSL is widely accepted in the Internet community, and has been subjected to significant testing by the hacker community, such as eavesdropping, impersonation, and tampering.

Management and audit tasks

- Management actions
  - Set up and change access control lists
  - Enable or disable security checks
  - Create and monitor user ID timeouts
  - User ID re-verification
  - Password monitoring
- Audit
  - Violation messages on job log
  - Violation queue monitoring
  - Access list

Figure 7-12. Management and audit tasks WM100 / VM1001.0

Notes:

The types of tasks listed above may be a shared job between the WebSphere MQ administrator and the security team, depending on your organization. It is doubtful that some of the items listed will be handled outside of the security group. However, the WebSphere MQ administrator needs to have accurate information to ensure that WebSphere MQ security is properly set up.

Using user ID context and alternate user ID

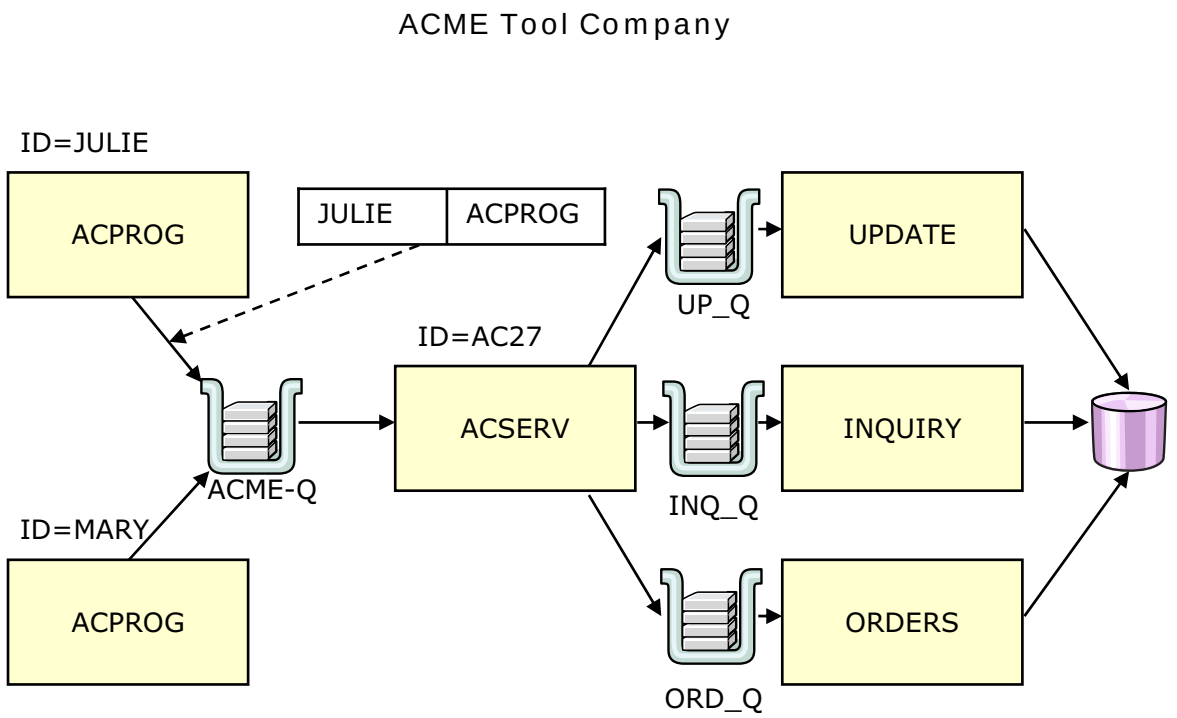


Figure 7-13. Using user ID context and alternate user ID WM100 / VM1001.0

Notes:

As you look at the example, consider that Julie and Mary both work in a department that enters requests for information, orders, and account updates. However, based on experience, Julie is authorized to do all three while Mary is only permitted to enter orders and inquire. How might the use of IBM WebSphere MQ facilities help ensure that only Julie is permitted to place messages on the update queue?

Using channel message exit

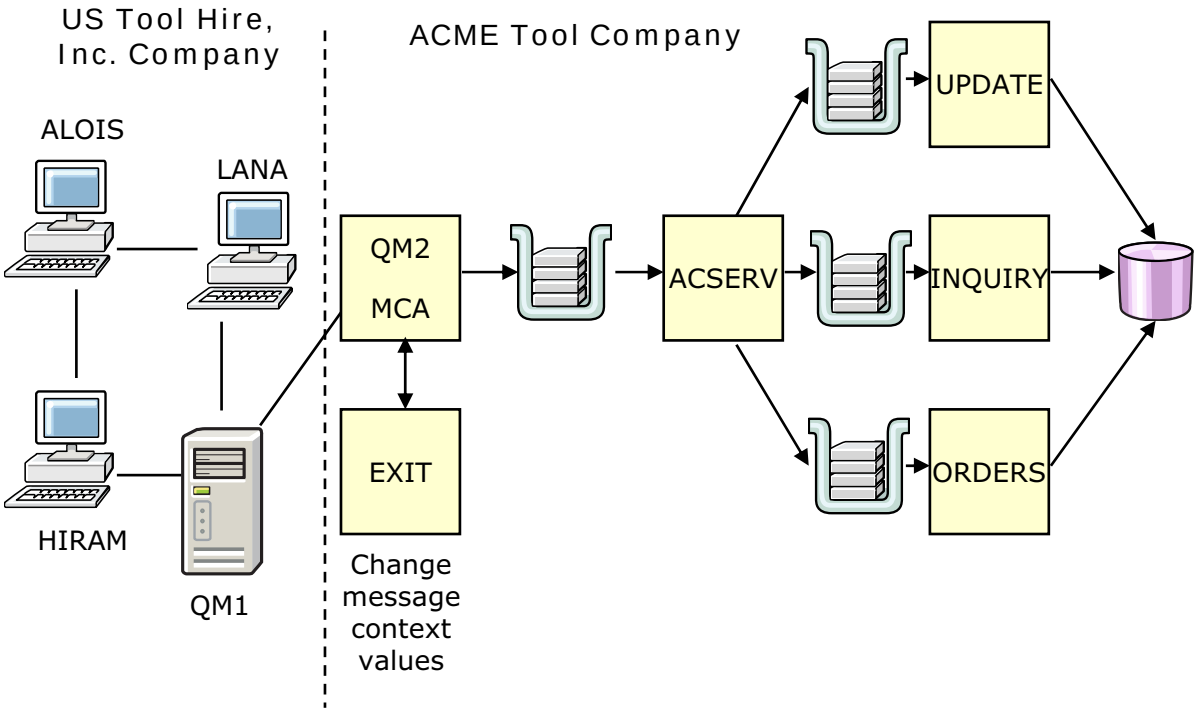


Figure 7-14. Using channel message exit WM100 / VM1001.0

Notes:

Now ACME wants to allow employees of US Tool Hire, Inc. to have limited access to the order application. They do not need to know anything about US Tool Hire's employees. How can ACME handle this using IBM WebSphere MQ capabilities?

Consider the use of message exit and MQMD context information.

Checkpoint

Exercise — Unit 7 checkpoint

1. There are various services involved in a security implementation; which one is WebSphere MQ most concerned with?

   a. Authentication
   b. Authorization
   c. Access control

2. **T**/**F** Usually, a site security administrator sets up all the security for WebSphere MQ objects.

3. Context security is used to:

   a. Check that users can access WebSphere MQ programs
   b. Tell the MCA where to deliver the message
   c. Allow programs to set up their authorization
   d. Enable access to queues to be controlled on a message-by-message basis

4. **T**/**F** Once an application puts a message on a queue and receives a successful completion code, there is no way to tell who created the message.

Unit objectives

After completing this unit, you should be able to:

- Describe security features of IBM WebSphere MQ platforms
- Consider which features are appropriate in a given environment
- Explain the role of IBM WebSphere MQ regarding security within and across enterprises

Figure 7-15. Unit summary WM100 / VM1001.0

Notes:

This unit has introduced the topic of security and its role with WebSphere MQ. The implementation of security is an important consideration. It should be part of early planning, involving the security management administration staff to ensure there is a clear understanding of the IBM WebSphere MQ requirements as well as the standards of the organization.

Unit 8. Linking, bridging, and the WebSphere MQ family

What this unit is about

This unit includes a brief overview of various linking and bridging mechanisms available to further expand the scope of messaging and queuing in an enterprise.

What you should be able to do

After completing this unit, you should be able to:

- Describe the various links and bridges used with WebSphere MQ
- List other functions and products that make up the WebSphere MQ family
- Describe the publish/subscribe functionality provided by IBM WebSphere MQ

How you will check your progress

Accountability:

- Classroom discussion
- Checkpoint questions

References

www.ibm.com/software/integration/wmq/library

IBM WebSphere MQ Library

SC34-6584 *WebSphere MQ System Administration Guide*

SC34-6606 *WebSphere MQ Publish/Subscribe User's Guide*

Unit objectives

After completing this unit, you should be able to:

- List various links and bridges used with IBM WebSphere MQ
- List other functions and products that make up the IBM WebSphere MQ family
- Describe the publish/subscribe functionality

Figure 8-1. Unit objectives WM100 / VM1001.0

Notes:

Over the past few years, IBM WebSphere MQ has become an essential part of many organizations' IT infrastructure. Because of its strengths, it has been used in several implementations that allow assured delivery of information between many different environments. In many cases, the underlying messaging and queuing mechanism is transparent, not only to the user, but to the programmer as well.

You take a brief look at the various links, bridges, functions, and products that make up the WebSphere MQ family.

8.1. Linking, bridging, and the WebSphere MQ family

The IMS bridge

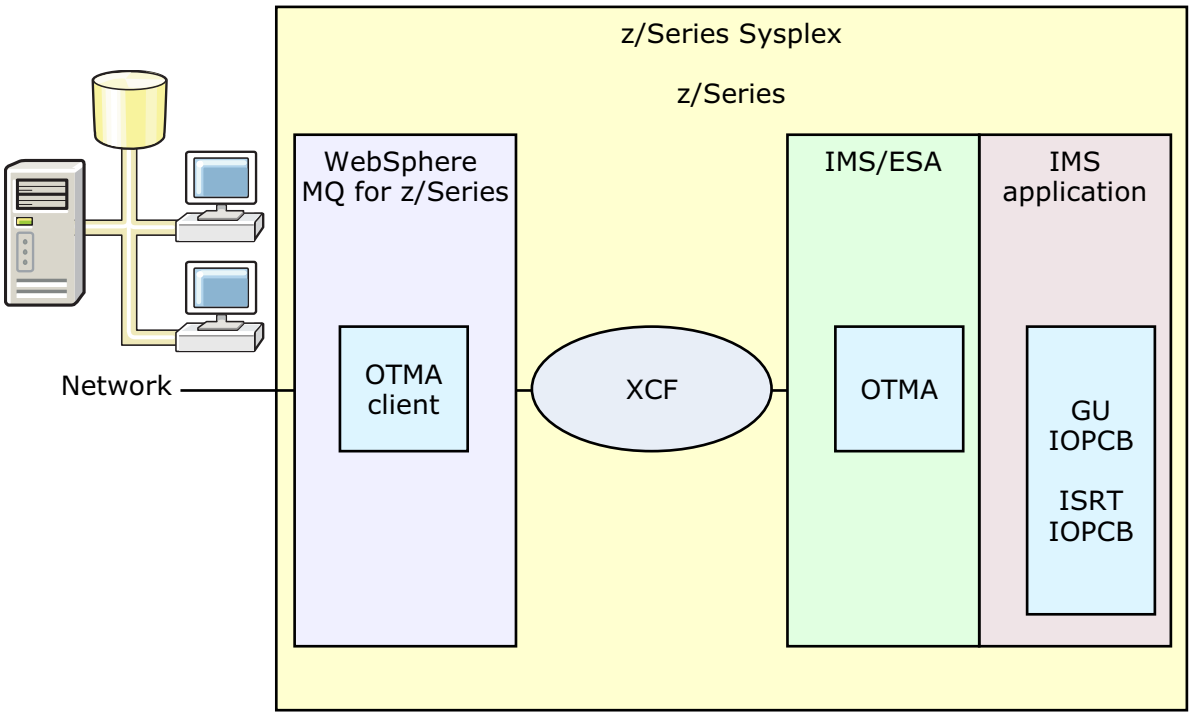


Figure 8-2. The IMS bridge WM100 / VM1001.0

Notes:

The WebSphere MQ-IMS bridge allows direct access to IMS applications from WebSphere MQ applications. This means that the IMS applications can be included in a IBM WebSphere MQ solution without any rewriting, recompiling, or relinking.

No MQI calls are required in the IMS applications. The sending application makes use of a special header as part of the messages intended for the IMS applications.

The IMS bridge is an IMS Open Transaction Manager Access (OTMA) client. OTMA is a connectionless client/server protocol that is transaction-based and functions as an interface for accessing IMS TM applications through the MVS Cross System Coupling Facility (XCF).

The IMS bridge is supplied as part of IBM WebSphere MQ for z/OS.

The CICS DPL bridge

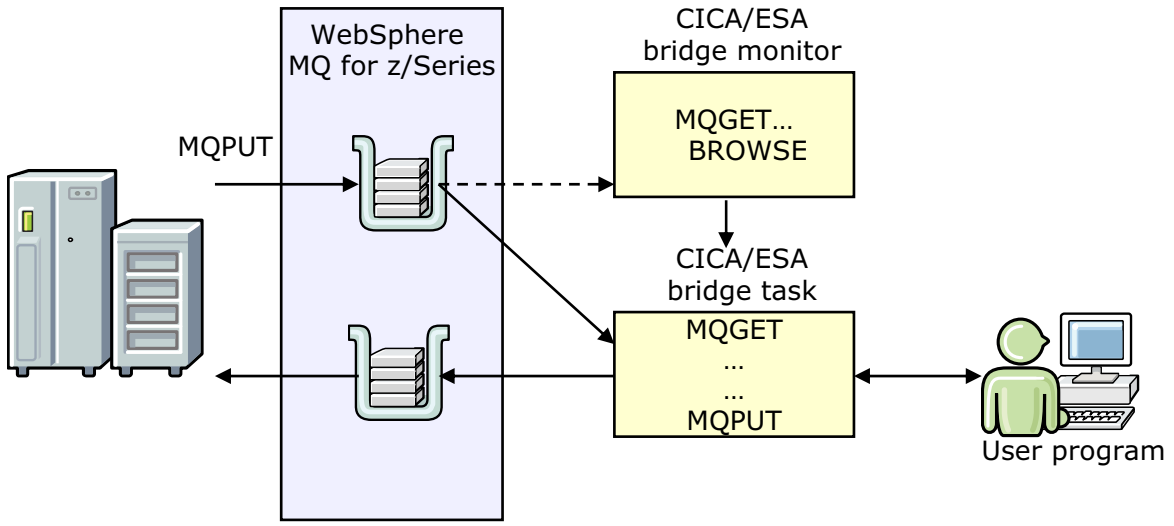


Figure 8-3. The CICS DPL bridge WM100 / VM1001.0

Notes:

The WebSphere MQ-CICS DPL (distributed program link) bridge allows a CICS program to be invoked using the EXEC CICS LINK command. The program must conform to the DPL subset of the CICS API. This allows non-CICS applications to communicate with CICS DPL programs without needing to change the CICS program to include WebSphere MQ API calls.

Like the IMS bridge, the CICS bridge is delivered as part of IBM WebSphere MQ for z/OS.

The CICS 3270 bridge

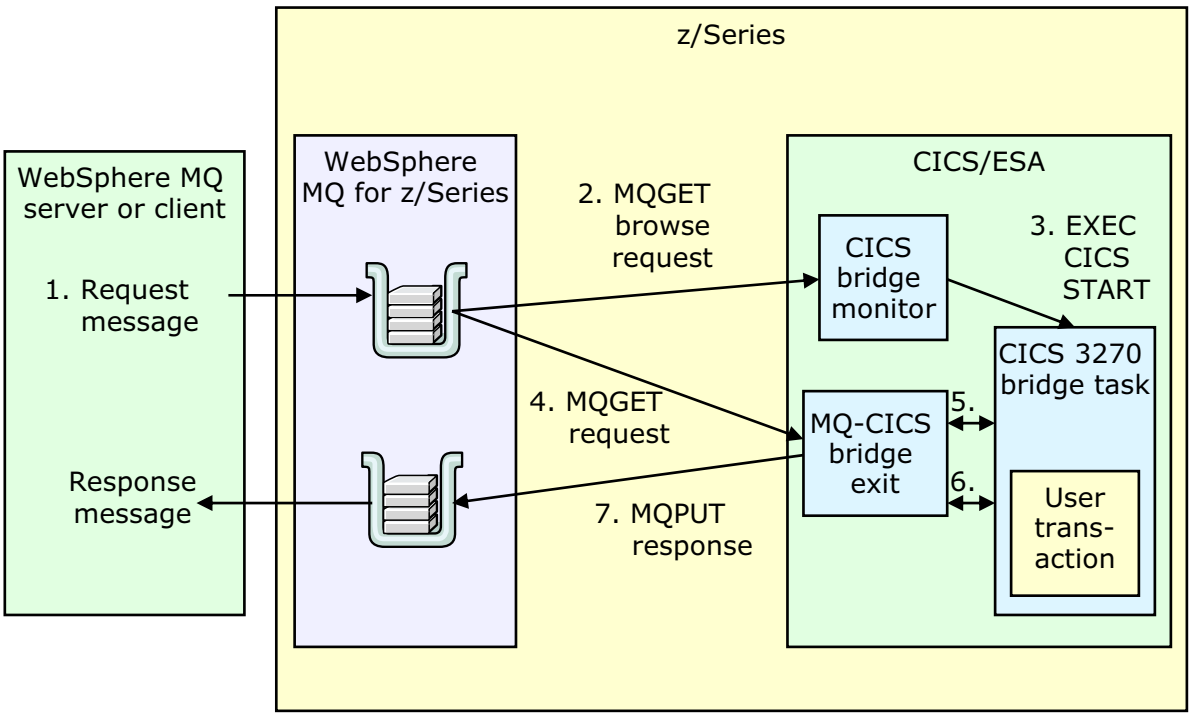


Figure 8-4. The CICS 3270 bridge WM100 / VM1001.0

Notes:

The WebSphere MQ-CICS 3270 bridge allows an application to run a 3270-based transaction without knowledge of the 3270 data stream. Data necessary to run the transaction is MQPUT on a queue by a non-CICS program. From the CICS transaction perspective, it runs as though it has a real 3270 terminal. In reality, WebSphere MQ messages are being used to communicate.

It is important to note that, unlike traditional emulators, the bridge does ***not*** replace VTAM flows with WebSphere MQ messages. The program is dealing with the transaction rather than via an emulator.

The link for SAP R/3

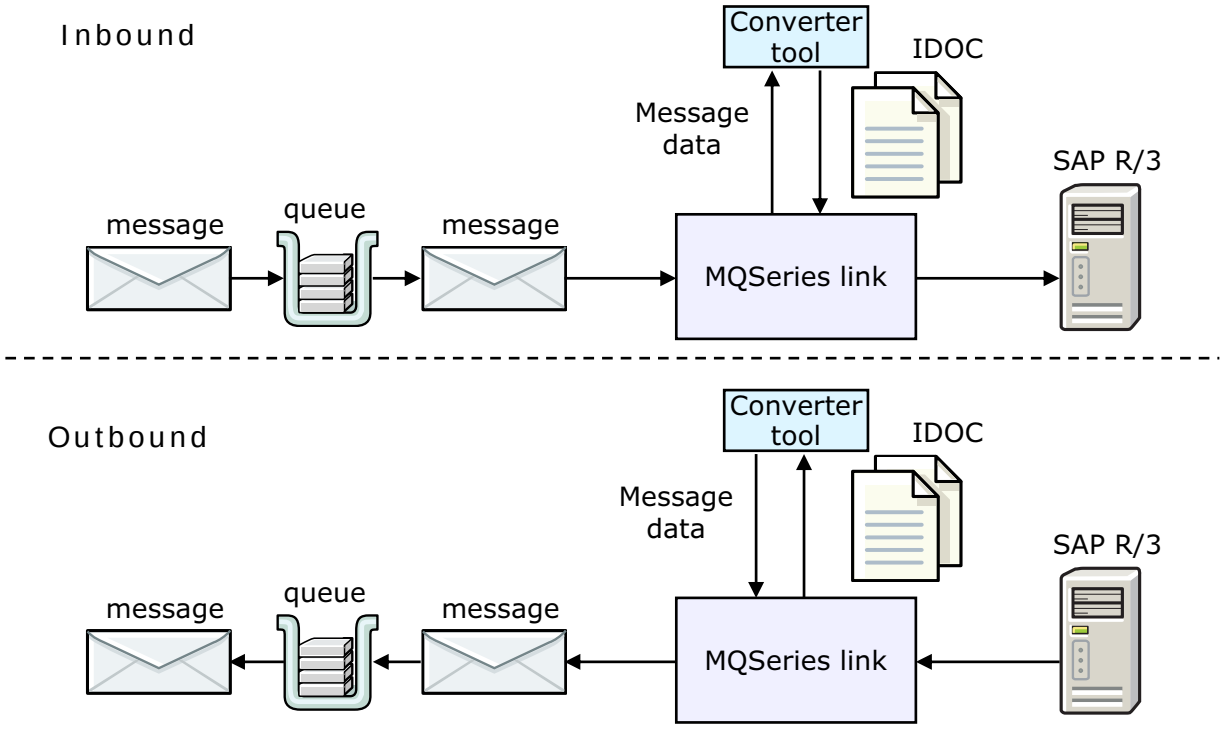


Figure 8-5. The link for SAP R/3 WM100 / VM1001.0

Notes:

The IBM WebSphere MQ link for SAP R/3 connects existing R/3 business applications with other applications, using IBM WebSphere MQ facilities. This allows programs to be easily adapted and more flexible because all networking is transparent to the application.

IBM WebSphere MQ acts as a transport mechanism to allow access to R/3 IDOCs from non-SAP R/3 environments. In addition, IBM WebSphere MQ can be used to connect two or more R/3 systems so they can take advantage of the messaging and queuing benefits.

WebSphere MQ Everyplace

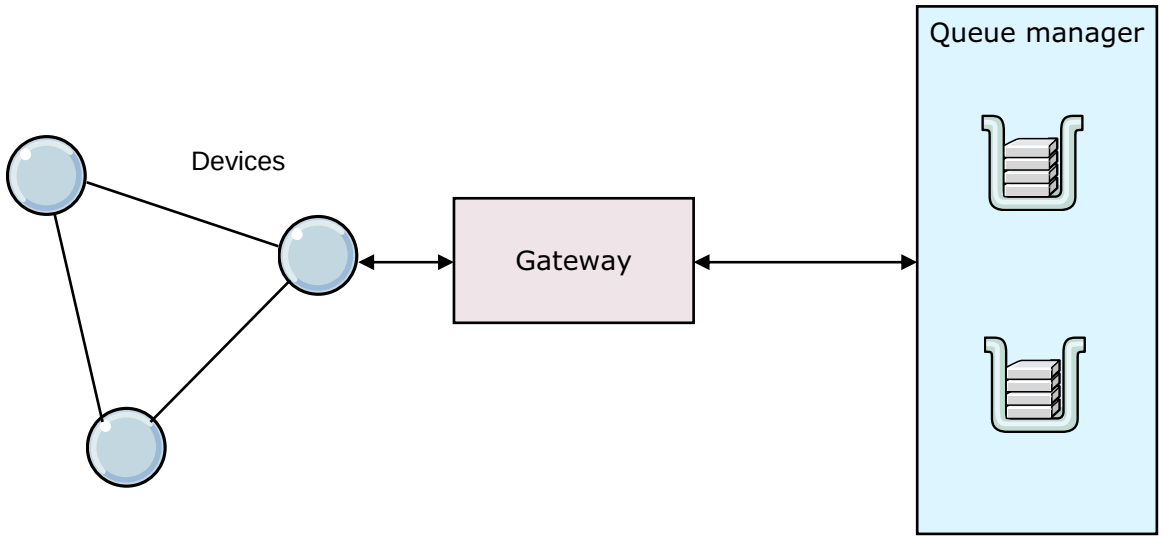


Figure 8-6. WebSphere MQ Everyplace WM100 / VM1001.0

Notes:

IBM WebSphere MQ Everyplace is a toolkit that enables users to write IBM WebSphere MQ Everyplace applications and to create an environment in which to run them.

The WebSphere MQ device is a computer running IBM WebSphere MQ Everyplace code *without a channel manager*. This means that a device is restricted to communication with only one other device or gateway at a time. IBM WebSphere MQ Everyplace devices can range from the very small (such as a sensor), through larger devices (such as a telephone, personal data assistant, or laptop computer), up to desktop machines and workstations.

A *gateway* is a computer running the IBM WebSphere MQ Everyplace code with a *WebSphere MQ Everyplace channel manager* or *WebSphere MQ Everyplace channel listener* configured. This offers all the capabilities of the device code plus the ability to communicate with multiple device gateways concurrently. Gateways also provide the mechanism for exchanging messages between a WebSphere MQ Everyplace network and a WebSphere MQ network.

Publish/subscribe examples

- Magazine publishing
- Airline departure notification
  - Notification boards might display (subscribe to)
  - all departures, departures at this terminal
  - Departures by this airline
  - Automated delay and cancellation notification
    - Email notification
    - Automated outbound telephone call message notification
- Weekly sale item valued customer notification

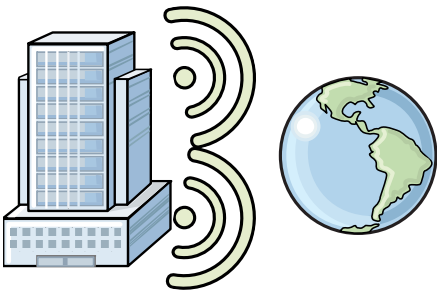


Figure 8-7. Publish/subscribe examples WM100 / VM1001.0

Notes:

Publish/subscribe is an asynchronous messaging paradigm where senders (publishers) of messages are not programmed to send their messages to specific receivers (subscribers). Rather, published messages are characterized into classes, without knowledge of what (if any) subscribers there may be.

Publishers are loosely coupled to subscribers, and needn't even know of their existence. With the topic being the focus, publishers and subscribers are allowed to remain ignorant of system topology. Each can continue to operate normally regardless of the other.

WebSphere MQ publish/subscribe

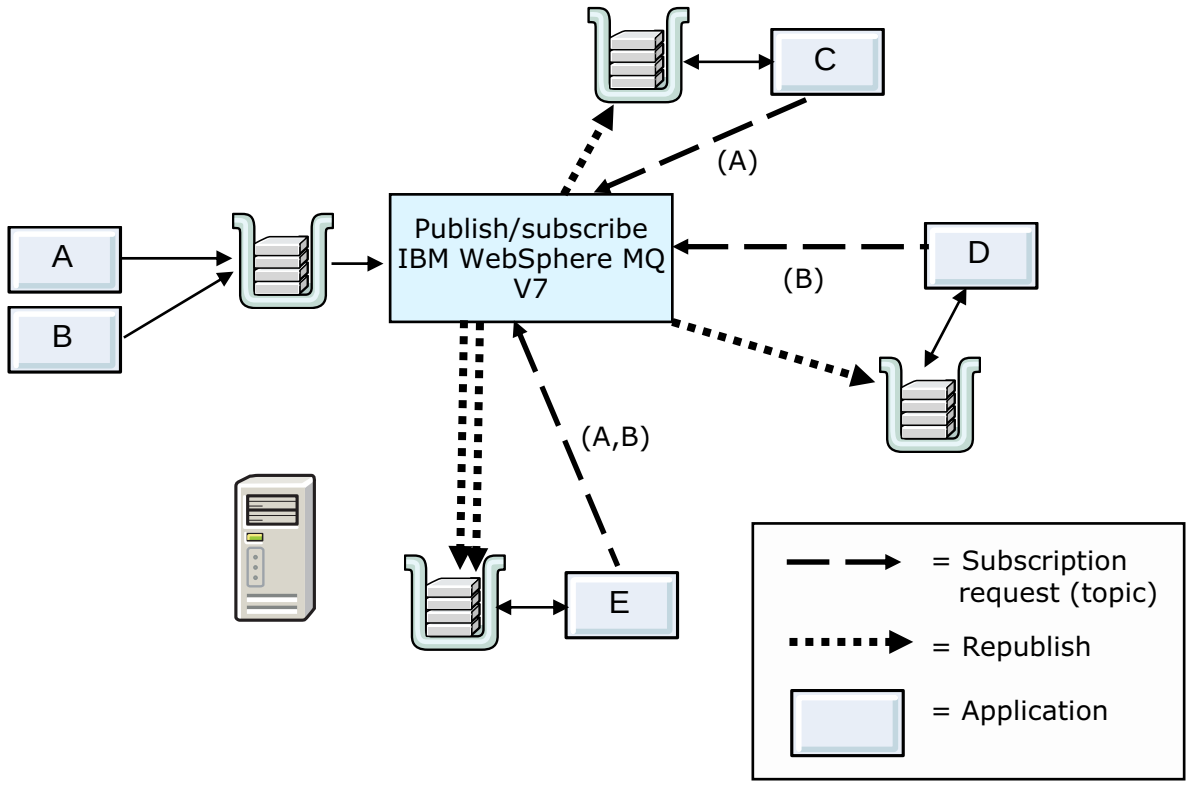


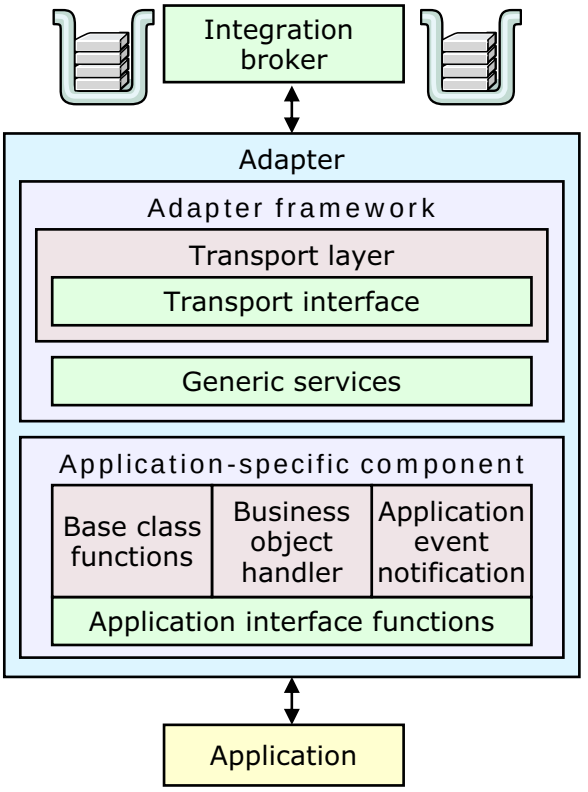
Figure 8-8. WebSphere MQ publish/subscribe WM100 / VM1001.0

Notes:

Publish/subscribe is a messaging interaction model. It defines an interaction where each message can be delivered to multiple receivers. The actual model defines zero or more recipients, all receiving the same published message. Publish/subscribe works by the use of topics (also known as subjects). Potential receivers subscribe to a set of topics that they are interested in. These potential receivers are known as subscribers. Senders, or publishers, publish messages on a particular topic. When messages are published, a publish/subscribe broker delivered with IBM WebSphere MQ V7 scans the subscription lists to see which subscribers are interested in the topic and republishes the message to each subscriber.

The model described above can be implemented on many different transports, providing different classes of service for the messages. There is an implementation available within IBM WebSphere MQ providing publish/subscribe on any type of supported messaging. IBM WebSphere MQ V7 also supports extended capabilities such as subscribing based on message content as well as topic.

WebSphere Business Integration Adapters



- Transport
  - JMS-based message queueing
  - Other transports available
- Adapter framework
  - Common runtime infrastructure
  - Assures uniformity in administration and behavior across the adapter family
- Application-specific component
  - Bidirectional connectivity with application API or technology interface

Figure 8-9. WebSphere Business Integration Adapters WM100 / VM1001.0

Notes:

Because not all application environments support messaging, it is necessary to provide an alternative mechanism to allow non-messaging applications to interact with a messaging environment. This alternate mechanism is WebSphere Business Integration Adapters. As shown on the diagram, there are three main parts to the adapter:

1. The transport is the target message exchange mechanism for the adapter. This is generally messaging, but the architecture of the adapters allows this to be any appropriate transport.
2. The adapter framework is the heart of the adapter. It is common to all of the adapters available from IBM, listed on the next page, and provides all of the support required to process information associated with the target application environment. As the description suggests, this component provides functions that are common to all of the adapters, regardless of the target application.
3. The application-specific component is the part of the adapter that understand the application environment and how to interact with it. As the name suggests, this component is specific to each application environment.

There are many different WebSphere Business Integration Adapters. Each provides interaction with a specific target environment. There is a set of application adapters, designed to interact with particular, named applications. There is also a set of technology adapters. These interact with specific technologies such as files, databases, Web services, and so forth.

For an current list of WebSphere Business Integration Adapters, see:
`www.ibm.com/software/integration/wbiadapters`

Application adapters

- ACCORD
- Ariba
- Centricity
- Clarify CRM
- eMatrix
- ESRI
- i2
- i2 Active Data Warehouse
- Indus Connect
- JD Edwards OneWorld
- Manugistics
- Maximo MEA
- MetaSolv Applications
- mySAP.com
- Oracle Applications
- PeopleSoft
- Portal Intranet
- QAD
- Siebel e-business Applications
- WebSphere Commerce
- NightFire Applications
- Spirent Applications
- Telcordia Applications
- Retek

Technology adapters

- Adapter for e-mail
- XML
- WebSphere MQ
- WebSphere MQ Integrator
- WebSphere MQ Workflow
- JText
- JDBC (Java Database Connectivity API)
- SWIFT
- FIX protocol
- COM
- CORBA
- HTTP
- Microsoft Exchange
- Healthcare Data Protocols
- Lotus Domino
- iSeries
- SAP Exchange Interface

e-business adapters

- Trading Partner Interchange Trading Networks
- Trading Partner Interchange Sol
- Trading Partner Interchange On-Ramp
- Web Services
- J2CA or EJB Connection to InterChange Server
- iSoft Peer-to-Peer Agent

Mainframe adapters

- CICS
- IMS Transaction Manager
- IMS database manager
- Adapter for VSAM
- DB2 databases
- ADABAS
- Natural
- IDMS database

WebSphere Message Broker

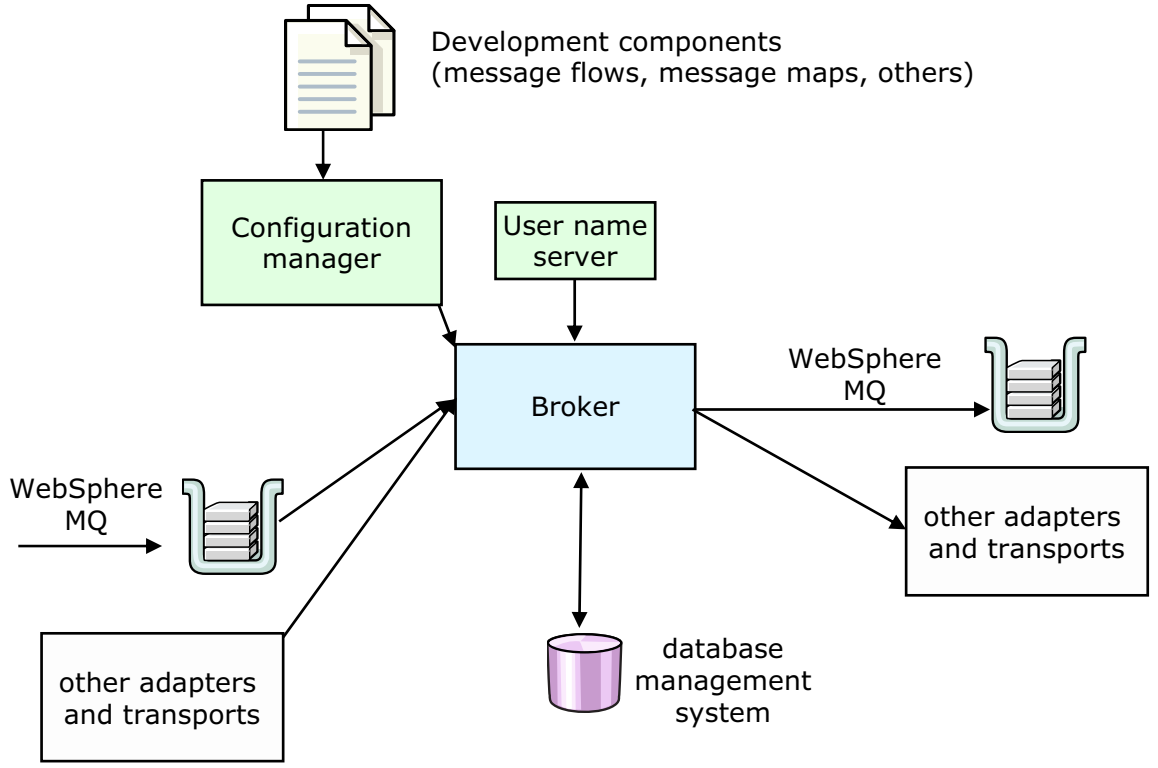


Figure 8-10. WebSphere Message Broker WM100 / VM1001.0

Notes:

IBM WebSphere MQ is designed to reliably transport messages between applications. In doing so, the queue managers do not pay any attention to the content of the messages that are being transported. This works well if the sending and receiving applications are aware of each other's requirements, particularly in terms of how the message data should be represented. If this not the case, some other component needs to broker the interaction between applications, providing content based routing, message transformation, and complex data handling capabilities. These capabilities are provided by the WebSphere Message Broker. Message Broker provides functions to enable messages to be broken down into their constituent parts (the message fields) and for messages to be routed and transformed.

WebSphere Message Broker supports the notion of message flows to control the processing of messages through the broker. A visual, declarative programming environment is provided to enable the construction, deployment, and debugging of these message flows.

As mentioned earlier, WebSphere Message Broker also supports sophisticated publish/subscribe messaging. It provides a superset of the capabilities offered by IBM WebSphere MQ.

WebSphere Message Broker supports message queue-based interactions, but also supports other transports, including HTTP (for Web services support), Java Message Service support, and direct IP connections, along with many others. WebSphere Message Broker also provides native support for message mapping between source and target applications. It also provides the capability of interacting directly with other vendor application environments.

Checkpoint

Exercise — Unit 8 checkpoint

1. The IMS bridge is:
   a. A Java client
   b. An OTMA client
   c. An MQI client
2. **T/F** The CICS DPL bridge enables a transaction to run as if it has a real 3270 terminal.
3. **T/F** WebSphere MQ is a major transport mechanism of the Enterprise Server Bus.
4. **T/F** Publish applications need to be aware of all subscribers interested in receiving published information

Unit summary

Having completed this unit, you should be able to:

- List various links and bridges used with IBM WebSphere MQ
- List other functions and products that make up the IBM WebSphere MQ family
- Describe the publish/subscribe functionality

Figure 8-11. Unit summary WM100 / VM1001.0

Notes:

The unit you have just seen should provide some interesting points to consider if you plan to implement IBM WebSphere MQ. Enabling the assured once-and-once-only delivery provided by IBM WebSphere MQ to aid in delivery of data throughout the enterprise becomes easier with the use of such links, bridges, and complementary products as you have seen.

Appendix A. Checkpoint solutions

Unit 1

1. b

   The Message Queue Interface (MQI) is introduced in this unit.

2. False

3. a, d (although it is possible for a message to contain only a message descriptor and no application data)

   Security information called context is part of the message descriptor.

4. c

   The MQMD (message descriptor) always accompanies WebSphere MQ messages.

5. False

   The trigger monitor starts the triggered application.

Unit 2

1. c

   Connection options allow the program to specify normal or fast path bindings. It is not used with MQCONN. It also allows certain clients to specify a particular connection to use at run time.

2. False

   MQSET can be used only to modify certain attributes of queues, no other MQ objects.

3. a, b, d

   MQPUT1 never requires an MQOPEN.

Unit 3

1. b

2. a, b

Unit 4

1. c

   Although a remote queue definition can be used, it is not required.

2. True

3. False

   The DEFINE command needs to include a USAGE clause that defines the queue as a transmission queue:
   `DEFINE QLOCAL(XMITQ1) MAXDEPTH(1000) USAGE(XMITQ)`

4. a, d

   The requester can start the channel but does not send data held in a transmission queue.

Unit 5

1. True

2. False

3. All of the above

4. All of the above

Unit 6

1. True

2. c

3. False

4. All do.

Unit 7

1. c

   Actually, WebSphere MQ leaves authentication and authorization to the security manager.

2. False
   Typically a WebSphere MQ administrator performs this task, in consultation with a security administrator.

3. d

4. False

Unit 8

1. b
2. False
3. True
4. False

Appendix B. Bibliography

All WebSphere MQ documents and reference materials are available online at the WebSphere MQ product family home page:

```
http://www.ibm.com/software/integration/wmq/library
```

On the Web site you will find all of the reference manuals that are listed on the first page of each unit of this course. Also on the site are links to white papers, webcasts, IBM Redbooks, and other reference material. The WebSphere MQ Information Center is also available for online use, as well as in downloadable form. It is a searchable reference tool that contains all the product documentation for specific releases of IBM WebSphere MQ. It is frequently updated as service packs and product updates are released.

Glossary of abbreviations and acronyms

A

ACL access control list

ACORD Agent-Company Organization for Research and Development

AIX Advanced IBM UNIX.

API application programming interface

ASCII American Standard Code for Information Interchange

B

B2B business-to-business

C

CA-AMS the new SAP Cross-Application – ALE Message Handling Systems

CD ROM Compact Disk Read Only Memory

CDRSC cross-domain resource

CHTML Compact HTML

CIA CICS Interdependency Analyzer

CICS Customer Information Control System

COBOL Common Business Oriented Language

COMMAREA communication area

CORBA Common Object Request Broker Architecture

CPU central processing unit

CR carriage return

CRL certificate revocation list

CRM customer relationship management

D

DASD direct access storage device

DB database

DPL distributed program link

DREP data repository

DRS data replication service

DSA dynamic storage area

DSA data set name

DSS Digital Signature Standard

DTD document type definition

E

EJB Enterprise JavaBean

ESA extended storage area

ESB enterprise service bus

F

FIFO first-in first-out

G

GUI graphical user interface

H

HTTP Hypertext Transfer Protocol

I

IBM International Business Machines Corporation

IMS Information Management System

IP Internet Protocol

ISC Integrated Solutions Console

ISC intersystem communication

ISPF Interactive System Productivity Facility

IT information technology

J

J2EE Java 2 Enterprise Edition

JAAS Java Authentication and Authorization Service

JAXP Java API for XML Parsing

JAXP Java API for XML Processing

JMS Java Message Service

JSP JavaServer Page

K

KB kilobyte

L

LAN local area network

M

MAC message authentication code

MQ Message Queue

MQI Message Queue Interface

MVS Multiple Virtual Systems, or Multiple Virtual Storage

N

O

OS operating system

OTMA Open Transaction Management Access

P

PDF Portable Document Format

Q

R

RACF Resource Access Control Facility

S

SAF Security Access Facility

SAS Secure Association Service

SMP Software Maintenance Program

SMP/E Software Maintenance Program Extended

SNA Systems Network Architecture

SOA service-oriented architecture

SSL Secure Sockets Layer

SWIFT Society for Worldwide Interbank Financial Telecommunication

SYSLINK system link

T

TCP Transmission Control Protocol

TCP/IP Transmission Control Protocol/Internet Protocol

TSO Time Sharing Option

U

UNIX Uniplexed Information and Computing System

V

VSAM Virtual Storage Access Method

VTAM Virtual Telecommunication Access Method

W

X

XCF cross-system coupling facility

XML Extensible Markup Language

Y

Z

z/OS zSeries operating system