WebSphere MQ Everyplace V2.0.2

Contents

Welcome to MQe

The full name of this product is WebSphere® MQ Everyplace® Version 2 Release 0 Modification 1, more usually expressed as WebSphere MQ Everyplace V2.0.1.

In this documentation, the product is generally referred to simply as *MQe*.

MQe is used for secure messaging on lightweight devices such as sensors, phones, personal digital assistants (PDAs), and laptop and desktop computers.

Programming Interfaces

The application programming interface to MQe is referred to in this documentation as the *MQe API*. The MQe API supports the Java™ and C programming languages as follows:

Java Provides access to all MQe function at Version 2.

There are three versions of the C support:

The Native C code base
Provides access to a subset of MQe function, the main restriction being that only a device queue manager can be used, so it can only send messages, not receive them.

The C Bindings
Are supplied for use until the Native C code base provides full functionality. Your application calls the C API in the bindings, and the call is routed to the Java classes for MQe to carry out the function. The bindings were written for MQe Version 1, but still provide access to nearly all of the Java function in MQe Version 2.

For more details on the code base see "Code base" on page 3.

See also: "Trademarks" on page 30.

MQe in a nutshell

What is MQe for?

- Secure messaging on lightweight devices such as sensors, phones, personal digital assistants (PDAs), and laptop and desktop computers

What is messaging?

- Software (as contained in MQ and MQe) that performs for you the work of sending and receiving data between your applications, and over networks. Message delivery is *assured*, *decoupled* from the application, and your application programmers do not need to have detailed communications programming knowledge.
- When an application wants to transfer data to another application, it puts the data into messages, and then puts the messages onto a *queue*.
- The *queue* is owned and run by a *queue manager*.
- A further application (or another part of the same one) can be configured to do one of the following:
  - The further application can retrieve those messages from the same queue.
  - The queue manager can be configured to send the messages on the queue through a *connection* over the network to a queue on a *remote* queue manager on another computer, where another application retrieves them.

  - The destination application can pull the messages across the network when it needs them.
- You can have many queues on one queue manager.
- On MQe, you can only have one queue manager per JVM or process.

What is MQe?
- A toolkit, supported on a range of platforms:
- The API is available in Java or C.
- The product function is delivered as Java classes and C .DLLs.

How does it work?
- A queue manager is created as a set of information that describes the queue manager's original basic configuration.
- This information is held in the MQe *registry* (Note: on Windows® systems this is not the Windows registry).
- On your device, an application can now start that queue manager, and it runs as long as the application runs. When the application stops, the queue manager stops.
- When a queue manager is running it can be extensively configured and reconfigured by sending it MQe *administration messages*, (which also update the information in the registry). Typically you generate these messages with the administrative tools MQe_Script and MQe_Explorer.
- On your server, you can write an application that runs the queue manager. This application can then run constantly, allowing client queue managers to send messages as required. On a Windows system, it is possible to run a queue manager application as a *service* so that it runs every time the computer is started.

How do you use it?
- You must write your own application to use MQe.
- You create your MQe *device queue manager* configuration, and develop your MQe application, on a PC.
- You download only the files and components that you need (for both your application, and MQe) to your device, and then run it there.
- You run an MQe *server queue manager*, on a computer other than a device, for the devices to connect to and send their messages to.
- If you want to communicate with MQ, you run an MQe *gateway queue manager* (on a computer other than a device) which acts as an intermediary - it can receive MQe messages from your devices (or your server), *transform* them into MQ-compatible messages, and then exchange them with MQ.
- You can create queue managers and remotely manage your MQe network using two downloadable SupportPacs: MQe_Script (scriptable commands), or MQe_Explorer (GUI).
- You can experiment with MQe, without writing an application, using these same tools.

Are there any special features?
- You can exploit *adapters* to map MQe to device interfaces. For example:
  - *Channels* (used on each end of a connection) exploit *protocol adapters* to run over HTTP, native TCP/IP, UDP, and other protocols.
  - Queues exploit *field storage adapters* to interface to a storage subsystem such as memory or the file system.
- You can achieve security at three levels:

  Local security
  Protects message-related data at a local level.

  Message-level security
  Protects messages between the initiating and receiving MQe application.

Queue-based security
Protects messages between the initiating queue manager and the target queue.

- You can use *rules* to customize the behavior of some of the main MQe components.
- You can use tracing and event logging to help debug your application.

See also:

"Code base"

"MQe SupportPacs" on page 21

Code base

Overview

The MQe application programming interface (API) is the programming interface to MQe. The MQe API supports the Java and C programming languages as follows:

The Java version
Provides access to all MQe function at Version 2. The detailed classes, methods, and procedures are described in the Java Programming Reference. Examples of MQe programming are given throughout this information center.

There are three versions of the C support:

The Native C code base
Provides access to a subset of MQe function, the main restriction being that only device queue managers can be used. Other restrictions are as follows (see also the table below):

- Does not support store-and-forward queues or bridge queues
- Supports the HTTP adapter only
- Supports the RLE compressor only
- Supports the RC4 cryptor only
- Supports the *MAttribute* and local security features only

The detailed methods and procedures are described in the C Programming Reference. Examples of programming MQe for the C API are given throughout this information center.

The C Bindings
Supplied for use until the Native C code base provides full functionality. Your application calls the C API in the bindings, and the call is routed to the Java classes for MQe to carry out the function. The bindings were written for MQe Version 1, but still provide access to nearly all of the Java function in MQe Version 2. The detailed methods and procedures are described in the C Programming Reference. Examples of programming MQe for the C bindings are given in the C Bindings Programming Reference.

Types of queue manager

Throughout this documentation, and in the table below, the following queue manager descriptions are used, and it is important to distinguish between them:

Device queue manager
A queue manager with no listener component, and no bridge component. It therefore can only send messages, it cannot receive them.

Server queue manager
A queue manager that can have a listener added. With the listener it can receive messages as well as send them.

Gateway queue manager
A queue manager that can have a listener and a bridge added. With the listener it can receive messages as well as send them, and with the bridge it can communicate with MQ.

Table of options

Option	Java	Native C	C Bindings
Operating systems	Any with Java 2 (which began at Java Version 1.2)	PocketPC2002; PocketPC2003; Windows (from MQe V2.0.1.5 on);	Windows 32bit
Queue managers	Any	Device only	Any
Gateway to MQ (queue manager with bridge and listener)	Yes	No	Yes
Store-and-forward queues, bridge queues	Yes	No	Yes
Adapters	All	HTTP only	All
Compressors	All	RLE only	All
Cryptors	All	RC4	All
Security features	All	*MAttribute* and local only	All
Add messages to Trace	Yes	No	Yes
Event logging	Yes	No	Yes
Private registry and credentials	Yes	No	Yes
Attribute rules	Yes	No	Yes
Bridge rules	Yes	No	Yes
Classes for customizing	All	No	Some
Application loading	Yes	No	Yes

What's new in 2.0.2

Changes in the release of WebSphere MQ Everyplace Version 2.0.2

- All documentation has been removed from the toolkit install and placed on the Web.
- UTF-16 with surrogate character support has been added to the C Native code base.
- Support for WinCE .NET 4.2 in the C Native code base. It should be noted that support for the WebSphere MQ Everyplace C DLLs on WinCE .NET 4.2 is provided using the Platform Invocation Services provided by the .NET environment.
- A migration section called has been added. This section contains information about how to use the new classes shipped with MQe that allow users to send, publish and subscribe messages to the WebSphere Message Broker via a gateway queue manager.

Using this Information Center

Accessibility

The sections below on this page give specific tips on using Eclipse for the MQe documentation.

For more general information on accessibility in Eclipse, see Eclipse's own help on *Accessibility in the help system*.

Keyboard tips

- To move between panes use function key F6, or Ctrl-F6 if using a browser based on Mozilla 1.2 or newer.
- In any pane or toolbar, use the tab key to move between icons or objects.
- To move back to the previously viewed topic use the backspace key. To move forward and back use Alt-RightArrow and Alt-LeftArrow respectively.
- After following a link or a search to find a topic, locate the topic in the table of contents by clicking the Show in Table of Contents icon on the topic toolbar (from the keyboard use function key F6 to move between panes, or Ctrl-F6 if using a browser based on Mozilla 1.2 or newer), then on the toolbar use the Tab key to select the icon).
- To open context-menus use Shift-F10, or the special context-menu key if your keyboard has one.

The window panes

- The Navigation Pane is on the left. It has tabs for:
  - Contents
  - Search results
- The Topic Pane is the one you're reading these words in now, and it displays the current help topic. Click on highlighted links to jump straight to other topics (from the keyboard, use the Tab keys to move the focus between links, press Enter to actuate the focussed one). Click in the topic and press **Ctrl-F** to find words in the page (from the keyboard use function key F6 to move between panes, or Ctrl-F6 if using a browser based on Mozilla 1.2 or newer). From the keyboard use the UpArrow and DownArrow keys to scroll the page.
- The Toolbar across the top of this Topic Pane provides icons for: Go Back, Go Forward, Show in Table of Contents (useful after a search), and Print (from the keyboard use Alt-LeftArrow, Alt-RightArrow, the icon on the toolbar, and Ctrl-P respectively; use function key F6 to move between panes, or Ctrl-F6 if using a browser based on Mozilla 1.2 or newer, then on the toolbar use the Tab key to select the icon).
- The Toolbar above the Navigation Pane provides search facilities.

Useful features

- After using the Search toolbar to find a topic, locate the topic in the table of contents by clicking the Show in Table of Contents icon on the topic toolbar (from the keyboard use function key F6 to move between panes, or Ctrl-F6 if using a browser based on Mozilla 1.2 or newer), then on the toolbar use the Tab key to select the icon).
- Click in the Topic Pane (from the keyboard use function key F6 to move between panes, or Ctrl-F6 if using a browser based on Mozilla 1.2 or newer) and then press **Ctrl-F** to Find words in the page.
- Right-click on an entry in the Table of Contents and select **Properties** to open a dialog box where you can swipe and copy a URL for the topic that can be used for future reference, for example it can be added to your browser bookmarks list, or copied into an email.

Printing

- Click on the print icon on the toolbar, or click in the topic and press Ctrl-P (from the keyboard use function key F6 to move between panes, or Ctrl-F6 if using a browser based on Mozilla 1.2 or newer).

Font size

- Use your browser's own features to control the font size.

Search tips

Use the Search toolbar just above the Navigation pane to find topics in the whole library. Note the following:

- Use the Advanced Search to restrict the scope of your search within the library.

- When searching for a number, with some versions of the Eclipse help browser the search will fail unless you enclose the number in double-quotes, for example: ″**2.0.1**″.
- Use * for incomplete words, for example: **admin*** will find occurrences of *administration*, *administer*, and so on.
- Search for multiple words by separating them with spaces, for example: **queue manager** will find both topics with the word *queue*, and topics with the word *manager*.
- Search for a phrase by enclosing it in double-quotes, for example: ″**queue manager**″ will find only topics with the whole phrase *queue manager*.
- The found words will be highlighted when you view the topic.
- To find words in a displayed topic, click in the topic pane (from the keyboard use function key F6 to move between panes, or Ctrl-F6 if using a browser based on Mozilla 1.2 or newer) and press **Ctrl-F**.

What is MQe

Introduction to MQe

MQe is a member of the WebSphere MQ family of business messaging products, and also the WebSphere Everyplace family. You use it to write your own applications that exchange *messages* containing data, providing once and once-only assured delivery. MQe is designed to integrate well with other members of the WebSphere MQ family.

MQe is designed to satisfy the messaging needs of lightweight devices, such as sensors, phones, personal digital assistants (PDAs), andlaptop and desktop computers. It supports mobile environments and is suitable for use over public networks, supporting requirements that arise from the use of fragile communication networks. As many MQe applications run outside the protection of an Internet firewall, it also provides security capabilities.

To understand this product and the documentation, an understanding of the concepts of secure messaging is an advantage.

If you do not have this understanding, you might find it useful to read the WebSphere MQ book *An Introduction to Messaging and Queuing*, GC33-0805. This book is available in softcopy form from the Book section of the online WebSphere MQ library. This can be reached from the WebSphere MQ Web page: http://www.ibm.com/software/integration/websphere/library/.

MQe in the MQ family

Basic messaging

Messaging, irrespective of the particular product or product group, is based on *queues* and *queue managers*. Queue managers manage queues that can store messages. Applications communicate with a *local queue manager*, and *get* or *put* messages to queues. If a message is put to a remote queue (a queue owned by another queue manager), the message is transmitted over *connections* to the *remote queue manager*. In this way, messages can hop through one or more intermediate queue managers before reaching their destination. The essence of messaging is to uncouple the sending application from the receiving application, queuing messages at intermediate points, if necessary.

MQ and MQe supply MQ family messaging. Both are designed to support one or more hardware server platforms and most associated operating systems. Given the wide variety in platform capabilities, these individual products are organized into product groups, reflecting common function and design:

Distributed messaging
: WebSphere MQ for Windows NT®, Windows 2000, AIX®, iSeries™, HP-UX, Solaris, and other platforms

Host messaging
WebSphere MQ for z/OS®

Pervasive messaging
MQe for Windows, AIX, Solaris, Linux®, and HP-UX

For more details see "How MQe works" on page 8.

MQe

MQe supports a variety of network configurations. There is no concept of a client or a server as in the MQ host or distributed products. Instead, you can configure MQe *queue managers* to act as clients or servers, enabling them to perform application-defined tasks.

An example of tailored configuration is that you can give MQe the ability to exchange messages with MQ host queue managers. To do this, configure an MQe queue manager with *bridge* capabilities. Without the bridge, an MQe queue manager can communicate directly with other MQe queue managers only. However, it can communicate indirectly through other queue managers in the network that have bridge capabilities.

For more details see "How MQe works" on page 8.

How MQe extends the MQ family

MQe extends the messaging scope of the MQ family by:

- Supporting low-end devices, such as PDAs, telephones, and sensors. MQe also supports intermediate devices such as laptops, workstations, distributed, and host platforms. MQe offers once and once-only assured delivery of messages, and permits message exchange with other family members.
- Offering lightweight messaging facilities.
- Providing extensive security features to protect messages, queues, and related data, whether in storage or in transmission.
- Operating efficiently in hostile communications environments where networks are unstable, or where bandwidth is tightly constrained. MQe has an efficient wire protocol and automated recovery from communication link failures.
- Supporting the mobile user, allowing network connectivity points to change as devices roam. MQe also allows control of behavior in conditions where battery resources and networks are constrained.
- Operating through suitably configured firewalls.
- Minimizing administration tasks for the user. This makes MQe a suitable base on which to build utility-style applications.
- Being easily customized and extended, through the use of application-supplied *rules*.

MQe does not support all the functions of MQ. Apart from environmental, operating system and communication considerations, these are some of the more significant differences:

- No clustering support
- No distribution list support
- No grouped or segmented messages
- No load balancing or warm standby capabilities
- No reference message
- No report options
- No shared queue support
- No triggering
- No unit of work support, no XA-coordination
- Different scalability and performance characteristics

However, within MQe many application tasks can be achieved through alternative means using MQe features, or through the exploitation of subclassing, the replacement of the supplied classes, or the exploitation of the rules, interfaces, and other customization features built into the product.

What you might use MQe for

MQe supports mobility, and fragile communication networks. Because MQe is targeted at lightweight devices, it is frugal in its use of system resources. It offers tailored functions and interfaces and does not aim to provide exactly the same capabilities as other members of the MQ family. It also includes unique functions to support its particular classes of user, such as comprehensive security provision, messages, synchronous and asynchronous messaging, remote queue access, and message push and pull.

Scenarios and Applications

There are various possible types of MQe applications, many of which are expected to be custom applications developed for particular user groups. The following list gives some examples:

Retail applications

- Trickle feeding till transactions to host systems, such as message brokers

Consumer applications

- Supermarket shopping from home using a PDA
- Gathering traveller preferences on airlines
- Financial transactions from a mobile phone

Control applications

- Collection and integration of data from oil pipeline sensors transmitted via satellite
- Remote operation of equipment (such as valves) with security to guarantee the validity of the operator

Mobile workforce

- Visiting professionals, for example an insurance agent
- Rapid publication of proof of customer receipt for parcel delivery companies
- Information exchange between kitchen and waiting staff
- Golf tournament score keeping
- Secure mobile systems messaging for the police
- Job information for utility workers in situations where communication is frequently lost
- Domestic meter reading

Personal productivity

- Mail and calendar replication
- Database replication
- Downloading to laptops

How MQe works

The fundamental elements of the MQe programming model are messages, queues, and queue managers.

- MQe messages contain application-defined content. Messages are stored in a queue and can be moved across an MQe network. You can address messages to a target queue by specifying the target queue manager and queue name.
- Applications place messages on queues through a *put* operation and typically retrieve them through a *get* operation.
- Queues can either be *local* or *remote* and are managed by queue managers.
- The *registry* stores configuration data.

Read the rest of the topics in this section to learn more.

Messages

A message is a collection of data sent by one application and intended for another application. MQe messages differ from those supported by MQ messaging:

- In MQ, messages are byte arrays, divided into a message header and a message body. MQ creates the message header, which contains vital information, such as the identity of the reply-to queue, the reply-to queue manager, the message ID, and the correlation ID. The message body contains data that is useful only to the application.
- Messages in MQe have no concept of a header or a message body. They are of type *MQeFields*, which consists of a name, a *data type*, and the data itself. Message names are ASCII character strings of unlimited length, excluding any of the characters:

  `{ } [ ] # ( ) : ; , ' " =`

  Table 1 describes the different data types:

Table 1. Data types

Type	Description
ASCII	String or a dynamic array of invariant ASCII strings, excluding any of the characters { } [ ] # ( ) : ; , ' " =
Boolean	True or false value
Byte	Fixed array, or a dynamic array of byte values
Double floating point	Value, fixed array, or a dynamic array of double floating point values
Fields	Object or a dynamic array of fields objects (thus nesting of fields objects is supported)
Floating point	Value, fixed array, or a dynamic array of floating point values
Integer	4 byte value, fixed array, or a dynamic array of integers
Long integer	8 byte value, fixed array, or a dynamic array of long integers
Short integer	2 byte value, fixed array, or a dynamic array of short integers
Unicode	String or a dynamic array of Unicode strings

Additionally, messages include a UID (unique identifier) which is generated by MQe. This UID uniquely identifies each individual message object in the entire MQe network and is constructed from the:

Originating queue manager
: This is the name of the originating queue manager, which must be unique. It is added by the queue manager on receipt of the message. As it is ASCII, every character is one byte long.

Creation time
: This is the *timestamp* of a message. Therefore, in Java, this is the time on the local system that the message was created, and in C, this is the time on the local system that a field's item is added to a queue. The field item then becomes a message.

A message destined for another MQe queue manager does not require any additional information, though other properties are almost certainly present. Additional properties can:

- Reflect current status
- Be associated with a particular message subclass
- Allow you to customize a message.

Note: In the C code base a field's item only becomes a message upon arrival on a queue.

MQe adds property related information to a message (and subsequently removes it) in order to implement messaging and queuing operations. When sending a message between queue managers, you can add resend information to indicate that data is being retransmitted.

Messages can also have *attributes*. Attributes are fundamental to the MQe security model and allow selective access to content and the protection of content. They have the following properties:

Table 2. Attribute object properties

Property	Description
Authentication	Controls access
Encryption	Protects the contents when the object is dumped (and allows restoration)
Compression	Reduces storage requirements (for transmission and storage)
Rule (Not applicable to C code base)	Controls permitted operations

For more information on the properties in Table 2, see "Security" on page 17.

Queues

Queues are used to hold messages. Applications do not access queues directly but use the queues via the queue manager.

Queues are identified by name, and the name can be an ASCII character string of unlimited length, excluding any of the following characters:

```
{ } [ ] # ( ) : ; , ' " =
```

However, queue names must be unique within a particular queue manager. For interoperability with MQ, we recommend that you also observe MQ naming restrictions, including a maximum name length of 48 characters. The name length may also be restricted by the file system you are using. MQe supports a number of different queue types:

Local queues

Applications use local queues to store messages in a safe and secure manner (excluding hardware failure or loss of the device). Local queues belong to a specific queue manger. This can be either a standalone queue manager or a queue manager that is connected to a network.

Remote queues

Remote queues are local references to queues that reside on another queue manager in the MQe network. The local reference has the same name as the target queue, but the remote queue definition identifies the owning queue manager of the real queue. Remote queues also have properties concerned with access, security characteristics, and transmission options. Their mode of access can be either synchronous or asynchronous. Synchronous remote queues have no storage, the message is passed immediately to the remote queue manager. An asynchronous queue stores the message on local storage, a background thread is then used to send the message to the remote queue manager.

Store-and-forward queues

A store-and-forward queue stores messages on behalf of one or more queue managers until they are ready to receive them. This type of queue is not present in the C code base.
Store-and-forward queues have two main uses:

1. To enable the intermediate storage of messages in a network, so that they can proceed to their destination (a forwarding role)
2. To hold messages awaiting collection by a Home-server queue.

This type of queue is normally (but not necessarily) defined on a server or gateway. Store-and-forward queues can hold messages for many target queue managers, or there may be one store-and-forward queue for each target queue manager.

Home-server queues

While remote queues and store-and-forward queues push messages across the network, with the sending queues initiating the transmission, home-server queues pull messages from a remote queue. Messages are never addressed to a home-server queue.

A home-server queue definition identifies a store-and-forward queue on a remote queue manager. The home-server queue then pulls messages that are destined for its local queue manager from the store-and-forward queue. Multiple home-server queue definitions can be defined on a single queue manager, where each one is associated with a different remote store-and-forward queue.

Administration queues

An administration queue is a type of local queue that accepts administration messages. An administration message contains instructions, processed internally by the application, relating to a particular element of MQe. Each administration action can, optionally, cause an administration reply message to be sent back to the originating application. These reply messages inform you of the success or failure of the administration action. In this way, using administration queues allows an element on one queue manager to control the configuration of a second queue manager, either synchronously or asynchronously. Administration messages are processed in order of arrival on the administration queue. For further information, refer to the section on "Administration" on page 13.

MQ bridge queues

A bridge queue is a specialist form of remote queue, describing a queue on an MQ remote queue manager. Bridge queues put or get from the MQ queue they reference. In Java only, it uses a transformer to perform any necessary data or message reformatting as each message is exchanged between the MQe and MQ systems. You can only create a bridge queue on a gateway queue manager.

MQe stores data securely on queues, ensuring that messages are physically written to the media and not simply stored by the operating system. However, MQe does not independently log changes to messages and queues. Therefore, to recover from media failure, you need to deploy hardware solutions, such as RAID disk systems. Alternatively, map the queue into recoverable storage, for example database subsystems. The MQe clients are on lightweight systems, but often server queue managers are required to run 24/7 where failover is required.

MQe has four commonly used system queues:

Administration queue
Receives administration messages

Dead letter queue
Stores messages that cannot otherwise be delivered

Administration reply queue
Receives replies to administration messages

SYSTEM.DEFAULT.LOCAL.QUEUE
Shares a common name with the mandatory system queue on MQ servers.

Queue managers

The MQe queue manager allows MQe to support a variety of network configurations. It provides:

- A central point of access to a messaging and queueing network for MQe applications
- Optional client-side queuing
- Connection control
- Optional administration functions

- Once and once-only assured delivery of messages
- Automated recovery from failure conditions
- Customizable rules-based behavior

In MQe, you can only have one queue manager active on a single Java virtual machine (JVM), or in a single native application process at any one time. To have multiple queue managers on a machine, you require either multiple JVM or multiple native application processes.

Queue managers are identified by a globally unique name and an ASCII character string of unlimited length, excluding any of the following characters:
`{ } [ ] # ( ) : ; , ' " =`

This restriction is not enforced by MQe or MQ, but duplicate queue manager names may cause messages to be delivered to the wrong queue manager. For interoperability, we recommend that you limit the maximum name length to 48 characters. The file system that you are using may also restrict the name length.

You can configure queue managers with or without local queueing. All queue managers support synchronous messaging operations. A queue manager with local queueing also supports asynchronous message delivery. Asynchronous message delivery and synchronous message delivery have very different characteristics and consequences:

Synchronous message delivery
With synchronous message delivery the application puts the message to MQe for delivery to the remote queue. MQe simultaneously contacts the target queue and delivers the message. After delivery, MQe returns immediately to the application. If the message cannot be delivered, the sending application receives immediate notification. MQe does not assume responsibility for message delivery in the synchronous case (non-assured message delivery).

Asynchronous message delivery
With asynchronous message delivery the application puts the message to MQe for delivery to a remote queue. MQe immediately returns to the application. If the message can be delivered immediately, or moved to a suitable staging post, then it is sent. If not, it is stored locally. Asynchronous delivery provides once, and once-only assured delivery, because the message has been passed to MQe and it has become responsible for delivery (assured message delivery).

Queue manager configuration: A MQe queue manager needs an application to create the required environment before the queue manager is loaded. This means the application that starts the queue manager needs to have access to certain information before it can load the queue manager. For instance, if you are setting the packet size for the communications adapter in Java, the required Java property needs to be set before loading the queue manager. Two pieces of information that are required by the application are the location of the MQe Registry and the location of the queue store. The registry (not to be confused with the Windows Registry) is where the definition of all the objects belonging to the queue manager are held, for instance queues and connection definitions. This allows a queue manager to create the correct objects when it is loaded. The queue store is the location where the queues are located, and allows a queue manager to hold messages on local queues that persist between a queue manager being stopped and started.

You can configure the MQe environment using the API, utilities shipped with MQe, or management tools such as MQe_Explorer. These methods can capture the environment parameters in an initialization file, but this is optional.

You can configure a queue manager with MQ bridge capabilities. This is called a gateway and, in Java, it exchanges messages with MQ host and distributed products. The C code base uses a device queue manager only.

Queue manager operations: Queue managers support messaging operations and manage queues. Applications access messages through the services of the queue manager using methods such as:

Get This operation removes messages from a queue.

Put This operation places messages on a queue.

Delete By specifying the UID, you can delete messages from a queue without using the get operation.

Browse
You can browse queues for messages using a *filter* (see below). Browsing retrieves all the messages that match the filter, but leaves them on the queue. MQe also supports *Browsing under lock*. This allows you to lock the matching messages.

Wait In Java, applications can *wait* for a specified time for messages to arrive on a queue. This does not apply to the C code base.

Listen In Java, applications can listen for MQe message events, again with an optional filter. However, in order to do this, you must add a listener to the queue. Listeners are notified when messages arrive on a queue. This does not apply to the C code base.

Many of these operations take a **filter** as one of their parameters. A filter matches an element for equality and any parts of the message can be used for selective retrieval. Most method calls also include an attribute to be used in the encoding or decoding of a message.

Administration

The MQe interface handles the generation and receipt of administration messages, enabling administration. While applications are responsible for message-related functions, administration provides facilities to configure and manage MQe resources such as queues and connections.

Requests are sent to the administration queue of the target queue manager and replies can be received, if required. Any local or remote MQe application program can create and process *administration messages* directly or indirectly through helper methods.

The C code base provides an administrator which allows you to perform some administration actions. These actions are performed only on resources that are managed by the local queue manager.

The administration queue itself cannot perform the administration of individual resources. The relevant information is contained in each resource and its corresponding administration message.

Administration messages: Once created, queue managers are configured by the sending of administration messages to the target queue manager administration queue. A queue manager that does not have an administration queue cannot be administered. The intent behind using administration messages is that both local and remote administration is performed in an identical manner.

An administration message is created and sent to the administration queue of the queue manager to be administered. You can apply queue-based security attributes to control access. An administration message includes details of the request, indicates whether or not a response is required, and contains the address identifying the target queue manager and queue. Therefore, MQe has the following styles of administration message:

- Commands that indicate an administration action that does not require a reply
- Requests that require a reply
- Reply messages constructed from a copy of the original message

The sender can add additional fields for use by the receiver. The administration queue itself acts upon the message. Administration messages can inquire on, create, delete, or update objects. For a subset of the objects they can perform additional functions, such as stop and start.

Administration messages can also be generated indirectly through the MQe_Explorer, a management tool that provides a graphical user interface for system administration. MQe_Explorer is not included with MQe but is available for free download as a SupportPac™.

Selective administration: The authenticator on the administration queue can control access to administration. The supplied authenticator considers local applications to represent the same local user and, therefore, either enables or prevents administration for all of the applications.

Starting the authenticator on the connection, before any administration messages flow, controls remote administration applications. This distinguishes different remote applications from each other, and then enables or prevents administration for each remote application. In all cases, administration is either completely enabled or prevented.

An authenticator can keep track of permissions associated with user identities, and administration messages can subsequently be processed on the basis of these permissions. See "Security" on page 17 for more information on authentication. You can also use rules that are associated with queues to enable or prevent actions in a similar manner. See "Customizing rules" on page 20 for more information on rules.

Monitoring and related actions: Administration involves more than creating and modifying elements. It can include monitoring a system and informing an operator when a queue is full, or dealing with an error situation, for example taking appropriate action when a message arrives that is too large for its target queue. MQe handles these aspects using rules, whenever elements significantly change their status or when certain types of error situations arise. MQe provides a default rule implementation, which users can customize if they wish. See "Customizing rules" on page 20 for more information on this.

Connections

A connection provides a queue manager with information to establish communication links with a remote queue manager. Queue managers then use connections to exchange information. Connection definitions are stored locally at each queue manager.

Note: The C code base is a device queue manager only.

Some of the key features of connections are:

Support for both *synchronous* and *asynchronous* messaging
: Synchronous messaging provides a transmission service directly from the source application to the target queue, without queuing at the source queue manager. Asynchronous messaging is a transmission service from the source queue manager to the target queue, with possible queuing at the source queue manager.

***End-to-end service* provision**
: Connections go from the source queue manager to a destination queue manager, possibly running through intermediate queue managers. The underlying transport protocol used can change as the connection passes through these intermediates. Several connections can link together to form end-to-end connections.

Support for *compression*, *encryption*, and *authentication*
: Connections have these security characteristics to protect the data in transit.

Support for *client*/*server* operation
: Client/server connections are request/response. The client makes a request of the server and the server responds to that request. Note that this does not restrict the message flow. Messages can flow from client to server and from server to client.

The following diagrams show some typical MQe configurations. For the purpose of clarity, the diagrams show only the direct connections that have been defined. You can also define indirect connections that exploit the direct connections. In the diagrams, a line with the arrow pointing to the server represents a client/server connection. Clients can use the client/server connection both to send messages to the server

and to pull messages destined for themselves from that server. Lines with no arrows indicate MQ client channels that enable communications between MQe and MQ.

Figure 1. A stand-alone MQe queue manager

Figure 1 shows a standalone queue manager being used to support one or more applications that use queues to exchange data.

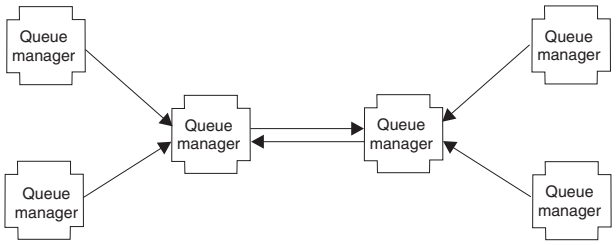

Figure 2. Small network configuration

Figure 2 shows a small network configuration, where the central server queue managers use a pair of direct client/server connections to exchange information. Each client queue manager uses a direct client/server connection to link to one of the server queue managers.

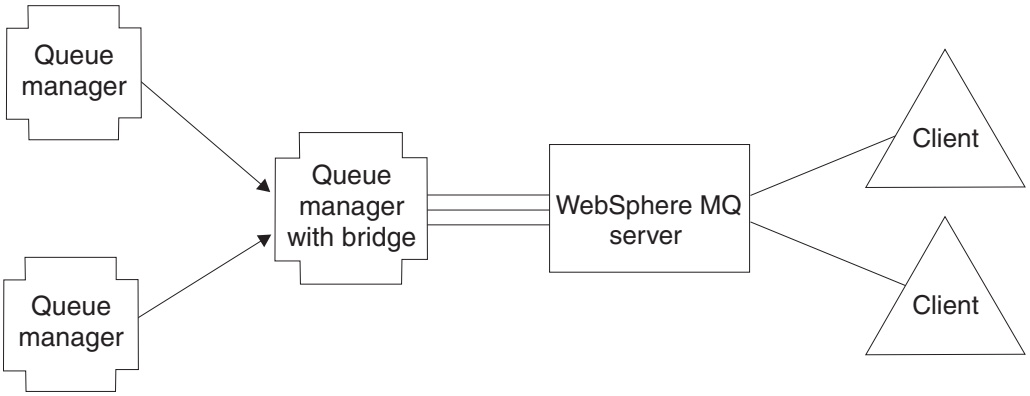

Figure 3. An integrated MQ family network

Figure 3 (Figure 3) shows an MQe configuration where one of the queue managers has been configured with the bridge option and the pool of client channels has been directed at a single target MQ host/distributed server.

As connections typically define the access to a remote queue manager, they are sometimes referred to as remote queue manager definitions.

You can also specify indirect connections. In this case, MQe routes the connection through other queue managers (which can be chained), and the protocol can change en route. Indirect connections are particularly useful in enabling devices to have a single point of entry to an MQe network.

As with most MQ elements, you can define aliases for connections. Use a local connection, defined as a connection with a name matching that of the local queue manager, to define alias names for the local queue manager itself.

Connections support bidirectional flows and are established by the queue manager as required. Asynchronous and synchronous messaging both use the same connections and the protocol used is unique to MQe.

Connection definitions determine the links and protocols to be used for a particular connection. At each intermediate node any messages flowing through are passed to the queue manager at that point. The queue manager will handle the messages according to the resources it has. So a message may be placed on a queue which might be a local queue, a remote queue, or a store-and-forward queue. Messages placed on remote queues will continue their journey according to the type of remote queue. Synchronous remote queues will move the messages onward immediately. Asynchronous remote queues will store their messages before moving them.

Connections are not directly visible to applications or administrators and are established by the queue manager as required. Connections link queue managers together and their characteristics are changed by MQe, depending upon the information to be flowed. Transporters are the MQe components that exploit connections to provide queue level communication. Again, these are not visible to the application programmer or administrator.

When assured messaging is demanded, MQe delivers messages to the application once, and once-only. It achieves this by ensuring that a message has been successfully passed from one queue manager to another, and acknowledged, before deleting the copy at the transmitting end. In the event of a communications failure, if an acknowledgment has not been received, a message can be retransmitted, as once-only *delivery* does not imply once-only *transmission*. However, duplicates are not delivered.

Connection styles: MQe supports client/server operation. A *client* can initiate communication with a server. A *server* can respond only to the requests initiated by a client. The components involved are:

Listener
Listens for incoming connection requests.

Queue manager
Supports applications through the provision of messaging and queuing capabilities.

Table 3. Connection styles

	Queue manager	**Listener**
Client	Yes	No
Client/server	Yes	Yes
Server	Yes	Yes
Servlet	Yes	No

Table 3 shows the relationship between these components and the connection style. The client/server connection style describes the situation where MQe can operate in either client or server mode. The servlet option describes the case where MQe is configured as an HTTP servlet with the HTTP server itself responsible for listening for incoming connection requests.

MQe applications are not directly aware of the connection style used by the queue managers. However, the style is significant in that it affects what resources are available to the parties, which queue managers can connect with other queue managers, how much memory MQe uses, and which connections can concurrently exist.

Adapters

Adapters are used to map MQe to device interfaces. For example:

- Channels exploit protocol adapters to run over HTTP, native TCP/IP, UDP, and other protocols.
- Queues exploit field storage adapters to interface to a storage subsystem such as memory or the file system.

Adapters provide a mechanism for MQe to extend its device support and allow version control.

Note: Unlike the MQe Java code base, the C code base uses the HTTP adapter only.

Dialup connection management

Dialup networking support for devices is handled by the device operating system.

When MQe on a disconnected device attempts to use the network (for example because a message must be sent) and the network stack is not active, the operating system itself initiates remote access services (RAS). Typically this takes the form of a panel displayed to the user, offering a dialup connection profile.

Until the connection is established, the operating system is in control. Consequently the device user must ensure that appropriate dialup connection profiles are available for the operating system to use. There is no explicit support for dialup networking in MQe.

Trace

Trace is enabled by running an independent program that performs tracing actions.

Calls to trace for information, warning, and error situations are embedded within MQe.

Applications can also call trace directly and, using the MQe Java code base only, add new messages.

Because the interface that a trace handler must implement is published, solutions can implement this interface to collect MQe and application trace, interleave it, and direct the output to where it can be collected. Several trace handlers are supplied as part of the product code.

Also, because most MQe exceptions are passed to the application for handling, the application exception handler can also route these to trace.

Event log

This does not apply to the C code base.

MQe provides event log mechanisms and interfaces that can be used to log status. For example, warning messages are logged if the messages on an asynchronous remote queue are unable to be delivered.

By default, logging is written to a `system.out`, but you can intercept this and direct it elsewhere.

The MQe event log does not log message data and cannot be used to recover messages or queues.

Security

MQe provides an integrated set of security features enabling the protection of message data both when held locally and when being transferred.

MQe provides security under three different categories:

Local security
: Protects message-related data at a local level

Message-level security
: Protects messages between the initiating and receiving MQe application

Queue-based security
Protects messages between the initiating queue manager and the target queue

Local and message-level security are used internally by MQe and are also made available to MQe applications. MQe queue-based security is an internal service.

The MQe security features of all three categories protect message data by use of an attribute, for example MQeAttribute. Depending on the category, the attribute is applied either externally or internally.

Each attribute can contain the following:

Authenticator
Provides additional controls to prevent access to the local data by unauthorized users

Cryptor
Controls the strength of protection required

Compressor
Optimizes the size of the protected data

Key Controls access by requesting a password

Target entity name
Requests the target queue name

These elements are used differently, depending on the MQe security category, but in all cases the MQe security feature's protection is applied when the attribute attached to a message is invoked.

The registry: The registry is the main store for queue manager-related information. Each queue manager has at least one registry. Every queue manager uses the registry to hold its:

- Queue manager configuration data
- Queue definitions
- Remote queue definitions
- Connection definitions
- User data (including configuration-dependent security information)

Registry information is stored using an adapter, usually the MQeDiskFields adapter.

Private registry and credentials: This section does not apply to the C code base.

As every entity needs its own credentials to be authenticated, we need to know:

1. How to execute registration to get the credentials
2. Where to manage the credentials in a secure manner

The private registry enables the secure management of an entity's private credentials, and the public registry manages the set of public credentials.

The private registry provides a base registry with secure or cryptographic tokens. For example, it can be a secure repository for public elements like mini-certificates, and private elements like private keys.

The private registry allows only authorized users to access the private elements. Normally, only the legitimate queue manager user can access the registry using a PIN. However, configuration options enable you to bypass this if you are not overly concerned with security issues.

The private registry provides support for services, for example digital signature and RSA decryption, in such a way that the private objects never leave the private registry. By providing a common interface, it hides the underlying device support, which currently is restricted to the local file system.

Auto-registration: MQe provides default services that support auto-registration. These services are automatically triggered when an authenticatable entity is configured, for example when a queue manager is started or when a new queue is defined. In both cases registration is triggered and new credentials are created and stored in the entity's private registry. Therefore, auto-registration provides a simple mechanism to establish credentials for message-level protection.

Auto-registration steps include:

1. Generating a new RSA key pair
2. Protecting and saving the private key in the private registry
3. Packaging the public key in a *new certificate* request to the default mini-certificate server

Assuming the mini-certificate server is configured and available, it returns the entity's new mini-certificate, along with its own. These servers and the protected private key are stored in the entity's private registry as its new credentials.

Public registry and certificate replication: MQe provides default services that enable MQe components to share mini-certificates. The MQe public registry provides a publicly accessible repository for mini-certificates. This is analogous to the personal telephone directory service on a mobile phone, the difference being that, instead of phone numbers, it is a set of mini-certificates of the authenticatable entities that are the most frequently contacted.

The public registry is not purely passive in its services. If accessed to provide a mini-certificate that it does not hold, and if configured with a valid home-server component, the public registry automatically attempts to fetch the requested mini-certificate from the public registry of the home server. These services can be used to provide an intelligent automated mini-certificate replication service that makes the right mini-certificate available at the right time.

Application use of registry services: The MQe queue manager exploits the advantages of using private and public registry services, but access to these services is not restricted. MQe solutions can define and manage their own entities, such as users. You can then use private registry services to auto-register and manage the credentials of the new entities, and public registry services to make the public credentials available where needed.

Default mini-certificate issuance service: The SupportPac *MQe Server Support* contains a WTLS Mini-Certificate Server and is available as a separate free download.

This software package provides a certificate issuance service for WTLS certificates. You can configure queue manager and queue entities on this certificate issuance server to provide a default mini-certificate issuance service that satisfies private-registry auto-registration requests with the issuance of WTLS certificates. You can use the MQe certificate issuance service to set up and manage a mini-certificate issuance service to issue mini-certificates to a carefully controlled set of entity names. The characteristics of this issuance service are:

- Management of the set of registered authenticatable entities
- Mini-certificate issuance
- WAP WTLS mini-certificate repository management

The security interface: An optional interface is provided that can be implemented by a custom security manager. The methods allow the security manager to authorize or reject requests associated with:

- Adding and removing class aliases
- Defining adapters
- Mapping file descriptors
- Processing connection commands

Customizing rules

Rules allow users to customize the behavior of some of the main MQe components.

MQe provides default rules where necessary, but you can replace these with application or installation-specific rules to meet customer requirements.

The rule types supported differ in how they are triggered and in what they can do.

Rules contain logic and can therefore perform a wide range of functions.

Attributes rules: Attribute rules apply to the Java code base only.

This rule class is given control whenever a change of state is attempted, for example, a change of:

- Authenticator
- Compressor
- Cryptor

The rule would normally allow or disallow the change.

MQ bridge rules: MQ bridge rules apply to the Java code base only.

These rule classes are given control when the MQe to MQ bridge code has a change of state. There is a separate bridge rule class to determine each of the following:

- What to do with a message when a listener cannot deliver it to MQe, when it is coming from MQ, for example, because the message is too big or the queue does not exist
- What state the bridge-administered queues should start in once the server is instantiated
- What to do when the bridge finds something wrong with the MQ Sync queue, that is the persistent store used for crash recovery (the default rule displays the problem only)
- How to convert an MQe message to an MQ message, and vice-versa using transformers

Queue rules: This rule class is invoked at key points in the life cycle of a queue, for example when:

- A message is added to a queue to see if a threshold is exceeded (that is, the number of messages or size of the message has been exceeded).
- A queue is opened or closed.
- A queue is removed from a queue manager. This does not apply to the native C code base.
- A message on a queue has exceeded either the queue's or its own expiry interval.

Queue manager rules: This rule class is invoked at key points in the life cycle of a queue manager, for example when:

- A queue manager is opened, for example, to start a background timer thread running to allow timed actions to occur.
- A queue manager is closed, for example, to terminate the background timer thread.
- The transmission of the queue manager's pending messages is triggered.

Classes

This section does not apply to the MQe native C code base.

MQe provides a choice of classes for certain functions that allow you to customize MQe behavior to meet specific application requirements. In some cases the interfaces to classes are documented so that additional alternatives can be developed. The table below summarizes the possibilities. Classes can be identified either explicitly or through the use of alias names.

Note: Some of the classes are not provided in the C Bindings API. See Java Programming Reference and C Programming Reference for definitive lists of the supported classes.

Many of these classes are automatically given an alias by MQe. These are documented in the Java Programming Reference in com.ibm.mqe.MQe.alias.

Table 4. Class options

Class	Alternatives supplied	Interfaces documented	MQe package	How to implement
Administration	No	Yes		
Authenticators	Yes	No	com.ibm.mqe.attributes	extend com.ibm.mqe.MQeAuthenticator
Communications adapter	Yes	Yes	com.ibm.mqe.adapters	extend com.ibm.mqe.adapters .MQeCommunicationsAdapter
Communications style	Yes	No		
Compressors	Yes	No	com.ibm.mqe.attributes	extend com.ibm.mqe.MQeCompressor
Cryptors	Yes	No	com.ibm.mqe.attributes	extend com.ibm.mqe.MQeCryptor
Event log	Sample provided	Yes		implement com.ibm.mqe.MQeEventLogInterface
Messages	No	Yes	com.ibm.mqe	extend com.ibm.mqe.MQeMsgObject
Queue storage	Yes	No		Normally the default as defined by the alias MsgLog: should be used.
Rules	Default classes provided	Yes		extend com.ibm.mqe.MQeRule
Storage adapter	Yes	Yes	com.ibm.mqe.adapters	extend com.ibm.mqe.adapters.MQeAdapter
Trace	Samples provided	Yes	com.ibm.mqe.trace	

Application loading:

Note: This section does not apply to the C code base.

When an MQe queue manager is loaded, the initiating application must load any other applications into the JVM.

Standard Java facilities can be used for this, or you can use the class loader included as part of MQe.

Therefore:

- Multiple applications can run against a single queue manager in the same JVM.
- Alternatively, you can use multiple JVMs, but each requires its own queue manager and each of these must have a unique name.

MQe SupportPacs

MQe is a family of products that collectively provide the tools needed to develop, deploy, and manage MQe messaging and queuing solutions. The family comprises:

1. The **MQe licensed product**, available on physical media from IBM® or as a Web download from:
   `http://www.ibm.com/software/integration/wmqe/`

   The licensed product includes:

- MQe Java classes
- Helper classes
- MQe C Bindings files and native C code base
- Application source code examples
- Utilities
- Reference manuals
- License information

The physical Program Product also includes entitlement to use the product for non-development use on certain platforms. Further capacity units need to be purchased for use on larger machines, or with the MQ bridge.

2. **MQe SupportPacs**, available as Web downloads from:

   `http://www.ibm.com/software/integration/support/supportpacs/`

   or

   `http://www.ibm.com/software/integration/wmqe/`

   The management tools in the MQe SupportPacs play an important role in all phases of application development and rollout. They are more sophisticated than the utilities included with the licensed product and are an essential aid to getting started, configuring, inspecting pilot networks, and managing production systems.

   EA01: WebSphere MQ Everyplace - XML conversion utility
   Software that can convert from an MQeFields object to an XML representation and vice-versa.

   EP02: WebSphere MQ Everyplace - DB2® Adapter User Guide
   Extends the persistent storage of messages to within DB2 databases.

   ES06: WebSphere MQ Everyplace Server Support
   Bundles MQe_Explorer, MQe_Script, MQe_MiniCertServer and MQe_Service which provide administration and security functions for MQe.

   MS0B: MQSeries® Java classes for PCF
   Java code that provides PCF message support. See how to use it in "MS0B - MQSeries Java classes for PCF"

MS0B - MQSeries Java classes for PCF

PCF messages are administration messages used by MQ queue managers. This SupportPac contains Java code, which supplies PCF message support.

If you download and install it, and put the `com.ibm.mq.pcf.jar` file on your class path environment variable, you have access to Java classes, which can dynamically manipulate MQ resources. When PCF messages are combined with MQe administration messages, complete programmatic configuration of bridge resources, and corresponding resources on an MQe queue manager are possible. Example code contained in the `examples.mqbridge.administration.programming.AdminHelperMQ` class, used in conjunction with the `examples.mqbridge.administration.programming.MQAgent` demonstrates how to do this. This example code has been added to the `examples.awt.AwtMQeServer program`, such that selecting **View → Connect local MQ default queue manager** will:

- Ensure that a bridge object exists, creating one as required.
- Query properties from the default MQ queue manager.
- Attempt to connect that queue manager to the currently running MQe queue manager.
- Ensure that a proxy object representing the default MQ queue manager exists, creating one if necessary.
- Ensure an MQe client connection exists, and that a corresponding MQ server connection channel exists also, creating these resources if necessary.
- Ensure that a *sync queue* exists on the MQ queue manager.
- Ensure that a transmit queue on MQ exists, and create if necessary.

- Ensure that a matching MQ transmit queue listener exists in the configuration of the current MQe queue manager, creating one if necessary.
- Ensure that all the bridge resources are started.
- Ensure that a test queue on the MQ queue manager exists, creating one if necessary.
- Ensure that a matching MQe bridge queue exists, which refers to that test queue.
- Send a test MQeMQMsgObject to the test queue to make sure the configuration is working.
- Get the test MQeMQMsgObject from the test queue to make sure the configuration is working.

Planning your implementation

Licenses

Licenses required for deployment of your MQe applications

MQe is a toolkit that enables users to write MQe applications and to create an environment in which to run them. Before deploying this product, or applications that use it, please make sure that you have the necessary licenses.

1. The pricing of licenses for use of the Program on **servers** is based on ***Processor License Units***. Use of each copy of the Program on a server requires one *Processor License Unit* to be acquired for each processor or symmetric multiprocessor contained in the server on which the copy of the Program is to run. Different types of *Processor License Units* and ***Device Use Authorizations*** are required, depending on whether the Program is running on point-of-sale, that is retail, equipment or on another type of computer. Use of the Program on retail equipment requires a ***Retail*** server license, whereas use on other (non-retail) equipment requires a ***Network*** server license.
2. Additional *Device Use Authorization* is required for any use of the Program on a separate **client device**, except those included in the *Network* server license described in 3. below.
3. Each *Network* server license includes authorization for the restricted use of the Program with no more than one hundred (100) client devices, on condition that all such copies are used in the same economic enterprise or organization as the server copy.

Please refer to http://www.ibm.com/software/integration/wmqe/ for details of these restrictions.

Device platform use authorizations, which are recorded on Proof of Entitlement documents and valid to support the use of MQe, are required to use the product (other than for purposes of code development and test) on specified client platforms. These licenses do not entitle the user to use the MQe Bridge, or to run on the server platforms specified in the MQe pricing group lists published by IBM and also available on the Web via the URL mentioned below.

Please refer to http://www.ibm.com/software/integration/mqfamily/ for details of these restrictions.

What machines to use

What machines to use for developing and deploying your applications

You will need:

- A PC to write and compile your application.

  A Windows system is recommended because then you can run any of the MQe SupportPacs, in particular the MQe Explorer which is very useful while developing.

  This computer should have access to the internet for downloading MQe, SupportPacs, documentation, and so on.
- At least one of the computers or devices, that you intend to deploy on, to use for testing
- Any interface devices and cables for connecting your device to your development PC.

Which code base to use

The MQe Application Programming Interface (API) is the programming interface to MQe. Two languages are supported, Java and C.

The Java version provides access to all MQe functions. The detailed classes, methods, and procedures are described in the Java Programming Reference. Examples of MQe programming are given throughout this information center.

There are three versions of the C support:

The Native C code base provides access to a major subset of MQe functions. As the C code base is a device queue manager only it:

- Does not support store-and-forward queues or bridge queues
- Supports the HTTP adapter only
- Supports the RLE compressor only
- Supports the RC4 cryptor only
- Supports the *MAttribute* and local security features only

The detailed methods and procedures are described in the C Programming Reference. Examples of programming MQe for the C bindings are given throughout this information center.

The C Bindings are supplied for use until the Native C code base provides full functionality. They provide access to a major subset of MQe functions. The detailed methods and procedures are described in the C Programming Reference. Examples of programming MQe for the C bindings are given in the C Bindings Programming Reference.

Your MQe development cycle

Given the wide range of uses for MQe, the product is not installed, configured, and deployed in the same way as other members of the MQ family. There are three phases in the adoption of MQe:

1. Development and prototyping phase

MQe is available for installation and use without charge, subject to the conditions of the MQe development license. MQe applications are developed, using the functions provided by the JMS API or the MQe Java and C API. Please note the following regarding the creating and usage of a MQe queue manager. These issues may affect how you implement your application.

- Only one MQe queue manager may be used at any time in a JVM or a C process. A check is made within MQe and an error thrown if an application tries to start more than one queue manager. If more than one application needs to use a single queue manager, these must reside in the same JVM. When running multiple applications in a single JVM, the applications can use the MQeQueueManager.getDefaultQueueManager() method in order to check if a queue manager is already running. Any number of applications and queue managers may be used on a single machine, however care needs to be taken with regard to expected performance and system resources.
- Only one MQe queue manager should be started over a single message store. When using a graphical user interface, it is relatively easy to start up more than one instance of an application. It is therefore possible for multiple instances of the same queue manager running over the same message store to be started. This may have indeterminate results, especially if asynchronous queueing is being used.

  C The C API provides access to the functionality commensurate with the role of a client queue manager. The .NET environment may be used by using the Platform Invocation Services to call out to the MQe C DLLs. The required DLLs must be downloaded onto the device.

During the prototyping phase, it is strongly suggested that tests are run over the network intended to be used during production, with production level data. This will enable you to set correct expectations on performance and to assess the default communications settings in MQe.

Support from IBM is not included with the development license. However, support during application development and beyond is provided with the deployment license (see below).

2. **Deployment phase**
   The deployment phase refers to how you use the developed applications and, therefore, under the terms of the MQe license, capacity units are required to use the product. The Java classes and C API can only be distributed with the application with agreement from IBM, or where the users already have entitlement to use them. Otherwise, in Java, users must customize the necessary classes themselves and, in C, copy the MQe to the device.
3. **Management phase**
   Subsequently, when MQe queue managers are active within a network, tools are needed to inspect and manage them. Support for MQe is provided under the terms of the International Program License Agreement.

Support levels

This adoption life cycle explains the variation in level of support with platforms. For the MQe with capacity units, and Category 3 SupportPacs, IBM distinguishes between:

- Platforms where installation and application development is supported:
  - Problem reports on install, application development, and use are accepted
- Platforms where the application deployment is permitted but not directly supported:
  - Problem reports might be required to be reproduced on a supported platform
- Platforms where application deployment is supported:
  - Problem reports resulting from application deployment are accepted

Gaining experience on MQe

There are many ways to get started with MQe.

- Getting a queue manager up and running, followed by setting up a simple MQe network, is a productive way to become familiar with the product and its concepts.
- Writing a simple application is sound preparation for in-depth study of the product details.
- In the early stages it is generally not helpful to examine other members of the MQ family. Later, when the bridge functionality is of interest, this understanding becomes essential.

With this strategy in mind, new users are recommended to understand the essentials of the concepts presented in this introductory part of the documentation.

If you have access to a machine running a Windows operating system, download SupportPac ES06, MQe Server Support, which contains MQe_Explorer and follow the instructions given to get started with MQe. You do not have to install the licensed product beforehand, but if you do not, you are restricted by the terms of the license.

Using MQe with MQ

Introduction

Although an MQe network can exist standalone, without the need for an MQ server or network, in practice MQe is often used to complement an existing MQ installation.

This extends MQ's reach to new platforms and devices, and provides advanced capabilities, such as queue or message based security and synchronous messaging.

From an MQe application perspective, MQ queues and queue managers act as additional remote queues and queue managers. However, a number of functional restrictions exist because these queues are not accessed directly through MQe connections and an MQe queue manager, but require the involvement of an MQe gateway.

The gateway can send messages to multiple MQ queue managers either directly or indirectly, through MQ client channels. If the connection is indirect, the messages pass through MQ client channels to an intermediate MQ queue manager and then onwards through MQ message channels to the target queue manager.

Gateway (bridge) to MQ

This section does not apply to the C code base.

MQe supports the *MQ bridge*, which acts as an interface between MQe and MQ networks.

This bridge uses the MQ Java client to interface to one or more MQ queue managers, thereby allowing messages to flow from MQe to MQ and vice versa.

In the current version of MQe:

- one such bridge is recommended per server
- each is associated with multiple *MQ queue manager proxies* (definitions of MQ queue managers)
- a *queue manager proxy* definition is required for each MQ queue manager that communicates with MQe
- each of these definitions can have one or more associated *client connection services*, where each represents a connection to a single MQ queue manager
- each of these may use a different MQ server connection to the queue manager, and optionally a different set of properties such as user exits or ports

Message conversion

MQe messages destined for MQ pass through the bridge and are converted into an MQ format, using either a default transformer or one specific to the target queue. A custom transformer offers much flexibility, for example it is good practice to use a subclass of the MQe message class to represent messages of a particular type over the MQe network. On the gateway a transformer can convert the message into an MQ format using appropriate mapping between fields and MQ values and adding specific data to represent the significance of the subclass.

The default transformer from MQe to MQ cannot take advantage of subclass information but has been designed to be useful in a wide range of situations. It has the following characteristics:

- **Message flow from MQe to MQ:**

  The default transformer from MQe to MQ works in conjunction with the *MQeMQMsgObject* class. This class is a representation of all the fields you could find in an MQ message header.

  Using the MQeMQMsgObject, your application can set values using set() methods. Therefore, when an MQeMQMsgObject, or an object derived from it, is passed through the default MQe transformer, (that is the MQeBaseTransformer), the MQeBaseTransformer gets the values from inside the MQeMQMsgObject, and sets the corresponding values in the MQ message, for example, the priority value is copied over to the MQ message.

  If the message being passed is not an MQeMQMsgObject, and is not derived from the MQeMQMsgObject class, the whole MQe message is copied into the body of the MQ message. This is referred to as *funneling*. The message format field in the MQ message header is set to indicate that the MQ message holds a message in MQe *funneled* format.

- **MQ to MQe message flow:**

  MQ messages for MQe are handled similarly to those travelling in the other direction. The default transformer inspects the message type field of the MQ header and acts accordingly.

If the MQ header indicates a *funneled* MQe message, then the MQ message body is reconstituted as the original MQe message that is then posted to the MQe network.

If the message is not a *funneled* MQe message, then the MQ message header content is extracted, and placed into an MQeMQMsgObject. The MQ message body is treated as a simple byte field, and is also placed into the MQeMQMsgObject. The MQeMQMsgObject is then posted to the MQe network.

This MQeMQMsgObject class and the default transformer behavior mean that:

- An MQe message can travel across an MQ network to an MQe network without change.
- An MQ message can travel across an MQe network to an MQ network without change.
- An MQe application can drive any existing MQ application without the MQ application being changed.

Function

MQ remote queues are enabled for synchronous MQe put messaging operations from an MQe queue manager.

All other messaging operations must be asynchronous.

MQe administration messages cannot be sent to an MQ queue manager. The administration queue does not exist there and the administration message format differs from that used by MQ.

Compatibility

An MQe network can exist independently of MQ, but in many situations the two products together are needed to meet the application requirements. MQe can integrate into an existing MQ network with compatibility including the aspects summarized below:

Addressing and naming:

- Identical addressing semantics using a queue manager or queue address
- Common use of an ASCII name space

Applications:

- MQe is able to support existing MQ applications without application change.

Connections:

- The MQe gateway uses MQ client channels.

Message interchange and content:

- Interchange of messages between MQe and MQ
- Message network invisibility (messages from either MQe or MQ can cross the other network without change)
- Mutual support for identified fields in the MQ message header
- Once and once-only assured message delivery

MQe does not support all the functions of MQ. Apart from environmental, operating system and communication considerations, these are some of the more significant differences:

- No clustering support
- No distribution list support
- No grouped or segmented messages
- No load balancing or warm standby capabilities
- No reference message
- No report options
- No shared queue support
- No triggering
- No unit of work support, no XA-coordination
- Different scalability and performance characteristics

However, within MQe many application tasks can be achieved through alternative means using MQe features, or through the exploitation of subclassing, the replacement of the supplied classes, or the exploitation of the rules, interfaces, and other customization features built into the product.

Assured delivery

Although both MQe and MQ offer assured delivery, they each provide for different levels of assurance.

- When a message is travelling from MQe to MQ, the message transfer is only assured if the combination of *putMessage* and *confirmPutMessage* is used.
- When a message is travelling from MQ to MQe, the transfer is assured only if the MQ message is defined as persistent.

Further information

Related information on MQ

The following are related MQ publications, which you might find useful:

WebSphere MQ: An Introduction to Messaging and Queuing (GC33–0805)

This book describes briefly what MQ is, how it works, and how it can solve some classic interoperability problems.

WebSphere MQ: Quick Beginnings series

There are MQ Quick Beginnings books for each platform supported by MQ. These books contain platform-specific planning and installation information for MQ.

Websites

The MQe home page is at:

`http://www.ibm.com/software/integration/wmqe/`

By following the links from this home page, you can:

- Find out more about the features and benefits of MQe
- Obtain information about training and certification
- Access the MQe manuals in PDF and HTML format
- Download the latest upgrades and trial code.

You can download the MQe SupportPacs by choosing the product *WebSphere MQ Everyplace* on this page:

`http://www.ibm.com/software/integration/support/supportpacs/`

You might also be interested in the home page for MQ, which you can find at:

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

and the home page for the MQ family:

`http://www.ibm.com/software/integration/mqfamily/`

You can access the library of books for the MQ family of products at:

`http://www.ibm.com/software/integration/websphere/library/books/`

Translated documentation

The *MQe Introduction* book has been translated into languages other than English. These translated documents are available for download from the MQ library Web site at

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

Newsgroups

These newsgroups are all on news.software.ibm.com, and will also be on many other public newsservers.

For MQe:

- ibm.software.websphere.mqeveryplace

For MQ:

- ibm.software.websphere.mq
- ibm.software.websphere.mq.administration
- ibm.software.websphere.mq.programming

Other related:

- ibm.software.websphere.mqintegrator
- ibm.software.websphere.studio
- ibm.software.websphere.studio.*various*

MQe Certification

Training and certification on MQe is available. For more details, start here:

`http://www.ibm.com/software/integration/websphere/education/`

Notices &Trademarks

Notices

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

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

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

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

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

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM United Kingdom Laboratories,
Mail Point 151,
Hursley Park,
Winchester,
Hampshire
England
SO21 2JN

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

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Trademarks

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

AIX Everyplace IBM IBMLink iSeries MQSeries SupportPac WebSphere z/OS zSeries

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

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

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

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

Glossary

This glossary describes terms used in this book, and words used with other than their everyday meaning. In some cases, a definition might not be the only one applicable to a term, but it gives the particular sense in which the word is used in this book.

If you do not find the term you are looking for, try a softcopy search, or see the hardcopy index, or see the *IBM Dictionary of Computing*, New York:. McGraw-Hill, 1994.

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

A

application programming interface (API)
An application programming interface consists of the functions and variables that programmers are allowed to use in their applications.

asynchronous messaging
A method of communication between programs in which programs place messages on message queues. With asynchronous messaging, the sending program proceeds with its own processing without waiting for a reply to its message. Contrast with synchronous messaging.

authenticator
A program that verifies the senders and receivers of messages.

B

bridge A component that can be added to an MQe queue manager to allow it to communicate with MQ. See MQe queue managers.

C

channel
See *dynamic channel* and *MQI channel*.

channel manager
an MQe object that supports logical multiple concurrent communication pipes between end points.

class An encapsulated collection of data and methods to operate on the data. A class may be instantiated to produce an object that is an instance of the class.

client In MQ, a client is a run-time component that allows local user applications to send messages to a server.

compressor
A program that compacts a message to reduce the volume of data to be transmitted.

connection
Links MQe devices and transfers synchronous and asynchronous messages and responses in a bidirectional manner.

cryptor
A program that encrypts a message to provide security during transmission.

D

device platform
A small computer that is capable of running MQe only as a client, that is, with a device queue manager only.

device queue manager
See MQe queue managers.

E

encapsulation
An object oriented programming technique that makes an object's data private or protected and allows programmers to access and manipulate the data only through method calls.

G

gateway
A computer of any size running an MQe gateway queue manager, which includes the MQ bridge function. See MQe queue managers.

gateway queue manager
A queue manager with a listener and a bridge. See MQe queue managers.

H

Hypertext Markup Language (HTML)
A language used to define information that is to be displayed on the World Wide Web.

I

instance
An object. When a class is instantiated to produce an object, the object is an instance of the class.

interface
A class that contains only abstract methods and no instance variables. An interface provides a common set of methods that can be implemented by subclasses of a number of different classes.

internet
A cooperative public network of shared information. Physically, the Internet uses a subset of the total resources of all the currently existing public telecommunication networks. Technically, what distinguishes the Internet as a cooperative public network is its use of a set of protocols called TCP/IP (Transport Control Protocol/Internet Protocol).

J

Java Development Kit (JDK)
A package of software distributed by Sun Microsystems for Java developers. It includes the Java interpreter, Java classes and Java development tools: compiler, debugger, disassembler, appletviewer, stub file generator, and documentation generator.

Java Naming and Directory Service (JNDI)
An API specified in the Java programming language. It provides naming and directory functions to applications written in the Java programming language.

L

Lightweight Directory Access Protocol (LDAP)
A client/server protocol for accessing a directory service.

M

message
In message queuing applications, a communication sent between programs.

message queue
See *queue*.

message queuing
A programming technique in which each program within an application communicates with the other programs by putting messages on queues.

method
The object oriented programming term for a function or procedure.

MQ bridge
A computer with a gateway queue manager that can communicate with MQ. See MQe queue managers.

MQ and MQ family
Refers to **WebSphere MQ**, which includes these products:

- **WebSphere MQ Workflow** simplifies integration across the whole enterprise by automating business processes involving people and applications.
- **WebSphere MQ Integrator** is message-brokering software that provides real-time, intelligent, rules-based message routing, and content transformation and formatting.
- **WebSphere MQ Messaging** provides any-to-any connectivity from desktop to mainframe, through business quality messaging, with over 35 platforms supported.

MQ Messaging
Refers to the following **WebSphere MQ** messaging product groups:

- **Distributed messaging:** MQ for Windows NT and Windows 2000, AIX, iSeries, HP-UX, Solaris, and other platforms
- **Host messaging:** MQ for z/OS
- **Pervasive messaging:** MQe

MQe Refers to **WebSphere MQ Everyplace**, the MQ pervasive messaging product group .

MQI channel
Connects an MQ client to a queue manager on a server system and transfers MQI calls and responses in a bidirectional manner.

O

object (1) In Java, an object is an instance of a class. A class models a group of things; an object models a particular member of that group. (2) In MQ, an object is a queue manager, a queue, or a channel.

P

package
A package in Java is a way of giving a piece of Java code access to a specific set of classes. Java code that is part of a particular package has access to all the classes in the package and to all non-private methods and fields in the classes.

personal digital assistant (PDA)
A pocket sized personal computer.

private
A private field is not visible outside its own class.

protected
A protected field is visible only within its own class, within a subclass, or within packages of which the class is a part.

public A public class or interface is visible everywhere. A public method or variable is visible everywhere that its class is visible.

Q

queue A queue is an MQ object. Message queueing applications can put messages on, and get messages from, a queue.

queue manager
A queue manager is a system program that provides message queuing services to applications.

queue queue manager
This term is used in relation to a remote queue definition. It describes the remote queue manager that owns the local queue that is the target of a remote queue definition.

device queue manager
On **MQe**: A queue manager with no listener component, and no bridge component. It therefore can only send messages, it cannot receive them.

server queue manager
On **MQe**: A queue manager that can have a listener added. With the listener it can receive messages as well as send them.

gateway queue manager
On **MQe**: A queue manager that can have a listener and a bridge added. With the listener it can receive messages as well as send them, and with the bridge it can communicate with MQ.

R

registry
Stores the queue manager configuration information.

S

server
1. An MQe server is a device that has an MQe listener configured, and responds to requests for information in a client-server setup.
2. An MQ server is a queue manager that provides message queuing services to client applications running on a remote workstation.
3. More generally, a server is a program that responds to requests for information in the particular two-program information-flow model of client-server.
4. The computer on which a server program runs.

server queue manager
A queue manager with a listener that can therefore receive messages as well as send them. See MQe queue managers.

server platform
A computer of any size that is capable of running MQe as a server or client.

servlet
A Java program which is designed to run only on a Web server.

subclass
A subclass is a class that extends another. The subclass inherits the public and protected methods and variables of its superclass.

superclass
A superclass is a class that is extended by some other class. The superclass's public and protected methods and variables are available to the subclass.

synchronous messaging
A method of communicating between programs in which programs place messages on message queues. With synchronous messaging, the sending program waits for a reply to its message before resuming its own processing. Contrast with *asynchronous messaging*.

T

Transmission Control Protocol/Internet Protocol (TCP/IP)
A set of communication protocols that support peer-to-peer connectivity functions for both local and wide area networks.

transformer
A piece of code that performs data or message reformatting.

W

Web See World Wide Web.

Web browser
A program that formats and displays information that is distributed on the World Wide Web.

World Wide Web (Web)
The World Wide Web is an Internet service, based on a common set of protocols, which allows a particularly configured server computer to distribute documents across the Internet in a standard way.

Index