MQSeries Everyplace: An introduction

Overview

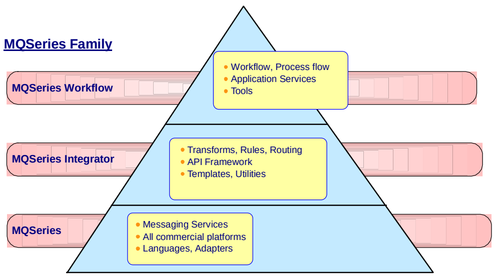
MQSeries Everyplace is a member of the MQSeries family of business messaging products. It is designed to satisfy the messaging needs of lightweight devices, such as sensors, phones, PDAs and laptop computers, as well as supporting mobility and the requirements that arise from the use of fragile communication networks. It provides the standard MQSeries quality of service, i.e. once-only, assured delivery, and exchanges messages with other family members. Since many MQe applications run outside the protection of an Internet firewall, it also provides sophisticated security capabilities.

Lightweight devices require the messaging subsystem to be frugal in its use of system resources and consequently MQe offers tailored function and interfaces appropriate to its customer set; it does not aim to provide all the capabilities offered by other members of the family. On the other hand, it does include a number of unique capabilities in order to support its particular classes of user, such as comprehensive security provision, object messaging together with a rich set of messaging functions..

MQe is a member of the IBM pervasive computing family, designed to integrate well with other members, such as DB2e. Since it provides the messaging service component, it is frequently a pre-requisite for solutions involving those products.

The MQSeries family

The MQSeries family includes many products, offering a range of capabilities, as illustrated below:



1Figure 1: The MQSeries family

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

Both MQSeries Workflow and MQSeries Integrator take advantage of the connectivity provided by the MQSeries messaging layer.

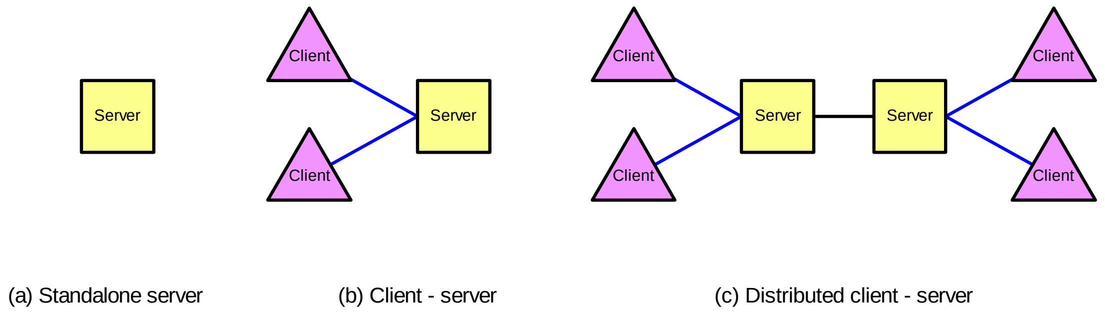
MQSeries messaging is supplied by a number of MQSeries (MQS) and MQSeries Everyplace (MQe) products; each one designed to support one or more hardware platforms and/or associated operating systems. Given the wide variety in platform capabilities, these individual products are organised into product groups, each reflecting common functions and design:

- **Distributed messaging**: MQSeries for Windows NT, AIX, AS/400, HP-UX, Sun Solaris, etc.
- **Host messaging**: MQSeries for OS/390.

- **Pervasive messaging**: MQSeries Everyplace for Windows NT, etc.

Messaging itself, irrespective of the particular product, is based on *queue managers*. Queue managers manage *queues* that can each store *messages*. Applications communicate with a chosen local queue manager, and *get* or *put* messages to queues. If a message is put to a remote queue, i.e. one owned by a remote queue manager, then it is transmitted over *channels* to that 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 decouple the sending application from the receiving application, queuing messages en route if necessary. All MQSeries messaging products are concerned with the same basic elements of queue managers, queues, messages and channels, though there are many differences in detail.

MQSeries *host* and *distributed messaging* products are used to support many different configurations, each of which involve *clients* and *servers*, for example:



.

2Figure 2: Simple host and distributed configurations

In the simplest case a *standalone MQS server* is configured, running a single queue manager. One or more applications run on that server, and exchange messages via queues. An alternative configuration is *client-server*. Here the queue manager only exists on the MQS server, but the clients each have access to it via a *client channel*. The client channel is a bi-directional communications link that flows a unique MQS protocol implementing something akin to a remote procedure call (RPC). Applications can run on the clients, accessing server queues. One advantage of client-server is that the client-messaging infrastructure is lightweight, being dependent upon the server queue manager. The downside is that clients and their associated server operate synchronously and require the client channel to be always available.

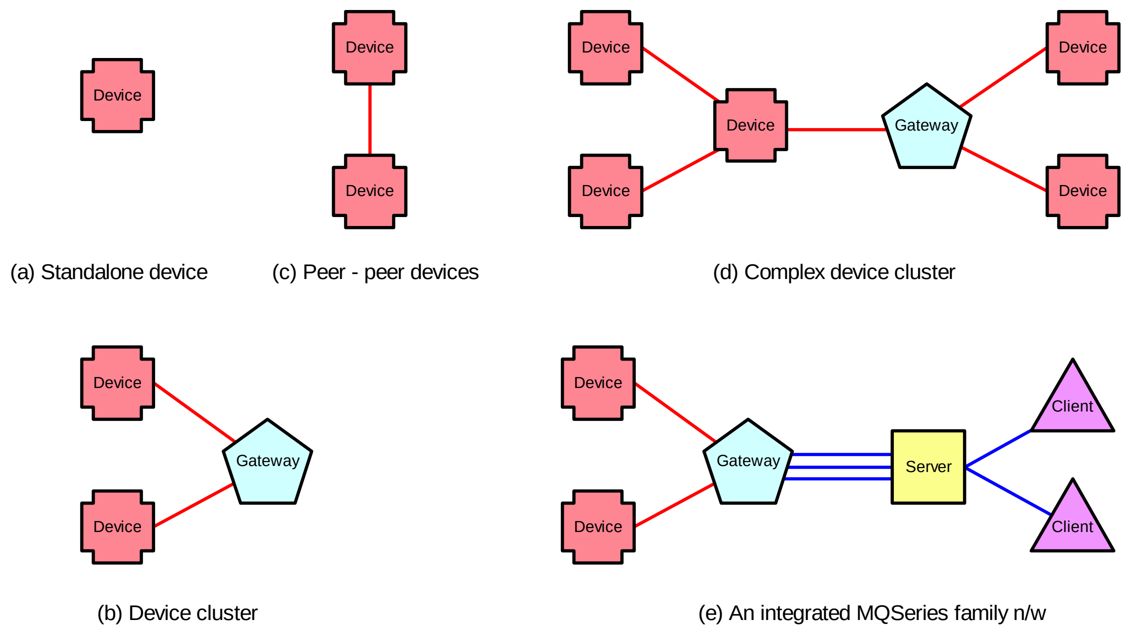
The distributed *client-server configuration* shows a more complex case with multiple MQS servers involved. In these configurations, servers exchange messages through *message channels*. Message channels are uni-directional, with a protocol designed for the safe, asynchronous exchange of message data. These message channels need not be available for the clients to continue processing, though no messages can flow between servers when they are not operational.

The *pervasive messaging* product MQe supports configurations through the provision of *devices* and *gateways*. The MQe device is a computer running the MQe device code. Such device computers can range from the very small (such as a sensor on an oil pipeline), to larger devices (such as a telephone, PDA or laptop computer), through to desktop machines and application departmental servers. Frequently such device computers are known as *pervasive devices*, though this implies a size and capability restriction that is not present in the product. A gateway is a computer running the MQe gateway code. Gateways are used to attach MQe devices. Gateways are also the mechanism through which MQe devices are attached to MQS servers.

To first approximation, devices combine many of the attributes of MQS clients and MQS servers. Thus devices can be configured with a full queue manager and are thus capable of asynchronous operation. They can also access queues held remotely, a feature present in clients. Unlike MQS servers, devices cannot attach clients. Devices can communicate directly with each other, offering a peer-to-peer messaging capability. Devices also communicate through channels, but in this case they are known as *dynamic channels*, to distinguish them from the MQS client and messaging channels. Dynamic channels are bi-directional and support the full range of functions provided by MQe, including both synchronous and asynchronous messaging.

Gateways necessarily support dynamic channels in order to communicate with devices. They can optionally support client channels in order to communicate with servers. Like servers, gateways have queue managers, and can therefore be used to run local messaging applications.

Some typical MQe configurations are:



3Figure 3: Typical MQe configurations

In (b) above a gateway is used to connect devices together; in (d) a device and a gateway are used to link devices. In (e) a gateway has been used to link a network of devices to an MQS server; in such configurations messages can then flow between all constituents, i.e. devices, gateways, servers, workstations and clients.

Requirements

MQe extends the scope of the messaging members of the MQSeries family:

- It expands messaging capabilities to the set of low-end devices, such as PDAs, telephones, sensors, allowing them to participate in an MQSeries messaging network. MQe offers the same once-only assured delivery and permits the two-way exchange of messages with others members of the family.
- It is designed to operate efficiently in hostile communications environments where networks are unstable, or where bandwidth is tightly constrained. Thus it has an efficient wire protocol and automated recovery from communication link failures.
- It supports the mobile user, allowing network connectivity points to change as devices roam. It also allows control of behaviour in conditions where battery resources and networks are failing or constrained.
- It minimises administration tasks for the user, such that the presence of MQe on a device can be hidden as much as possible – making MQe a suitable base on which to build utility-style applications.
- It includes extensive authentication and encryption facilities, making it suitable for applications outside firewalls.

There are many applications of MQe with most being custom applications developed for particular user groups. The list below indicates the possibilities:

- **Consumer applications**: supermarket shopping at home, air travel ticketing and associated services.
- **Business to business**: on-line trading, purchasing & auctions.
- **Control applications**: oil pipeline sensor data collection and/or control.

- **Mobile workforce**: visiting professional (e.g. insurance agent), freight/parcel delivery, fast-food waiter, golf tournament scoring.
- **Personal productivity**: mail/calendar replication, database replication, laptop downsizing.
- **Business integration**: application coupling, systems exploitation.

Technical overview

The most important elements of the MQe programming model are *messages*, *queues* and *queue managers*. An MQe message is an object that contains application-defined content. It is typically held in a queue and messages may be moved across an MQe network. Messages are addressed to a target queue by specifying the target queue manager and queue name pair. Applications place messages on queues through a *put* operation; they retrieve them through a *get* operation. Queues can either be *local* or *remote* and are managed by queue managers. Both devices and gateways store configuration data in the MQe *registry*.

MQe devices and gateways are interconnected through *dynamic channels*. Through MQS client channels, the MQe gateway may be connected to one or more MQS servers. Both MQe and MQS can co-exist on a single machine. An MQS server or network is not compulsory in an MQe configuration.

Dynamic channels can support:

- Connections with a dial-out and answering capability e.g. CellPhone, ScreenPhone etc.
- Dial-in connections
- Dial-out connections, e.g. a standard modem connected to an Internet Service Provider (ISP).
- Permanent connections, e.g. a conventional LAN, leased line, infrared or wireless LAN etc.

The communications protocols are implemented by a set of protocol adapters, one for each of the supported protocols. This enables new protocols to be added when required and the memory footprint on a given environment to be tailored to a particular configuration.

Message objects

The basic building block in MQe is the *fields object*s, an accumulation of *fields*, and where a field comprises a name, a data type and the data itself. Fields objects also have methods, for example they can enumerate their fields and a field can be verified to exist within an object. Similarly such objects can be tested for equality. More important however, is that they have the capability of dumping and restoring their field items to/from a byte array. This mechanism allows the object to dump itself for transmission, and to be restored on receipt. It can be over-ridden to allow fields objects to serialise themselves in other ways – for example to query a database for their field content at the time of transmission. *Attribute objects* (that contain the mechanisms to perform authentication, encryption and compression) may be associated with fields objects allowing the selective securing of content.

Message objects descend from the fields objects and have a unique identifier (UID) which is generated by MQe. No other information is required to be present in a message, though other fields will be certainly be present to carry the message information. Typically this will be achieved by using a descendant of the base message object class.

Queues

Queues are used to hold message objects. They are identified by a name that must be unique within a particular queue manager. If the device (or gateway) is configured to have local queues, then asynchronous, i.e. offline working is possible. In the absence of local queues, only synchronous communication with a remote device (or gateway) is available. Queues may have characteristics such as *authentication*, *compression*, and *encryption* - these attributes are used when a message object is moved to ensure protection of its contents. Queues are also associated with a file descriptor that controls the storage mechanism for the queue, for example that it is to be mapped into the file system, or alternatively into memory.

Queue managers

The MQe queue manager controls how and where the various message objects are stored or sent. Queue managers are identified by a name that must be globally unique. Queue managers can be configured to support queues (a *full queue manager*) or not (a *stub queue manager*). All queue managers support synchronous messaging operations, such as the ability to access queues to get, put and browse messages. A queue manager with local queues also supports asynchronous message delivery. Typically the stub queue manager option is used to reduce the footprint of MQe on small mobile devices.

Asynchronous and synchronous message deliveries have very different characteristics and consequences:

- **Asynchronous message delivery**: the application passes the message to MQe for delivery to a remote queue. An immediate return is made back to the application after the put operation. MQe temporarily holds the message locally, until it can be delivered. Delivery may be staged, with MQe responsible for delivery, i.e. this mode of operation provides *once-only assured delivery*.

  Once-only assured delivery means that MQe will deliver a message once, and not more than once. If a message cannot be delivered (e.g. because the target queue does not exist) the message will not be discarded but will be kept and its non-delivery flagged. It will not lose the message in the event of an application, system or network failure.

- **Synchronous message delivery**: the application puts the message to MQe for delivery. MQe synchronously contacts the target remote queue and places the message. After delivery MQe returns to the application. Contact with the remote queue manager may involve MQe routing through intermediate devices and/or gateways. The application is responsible for error handling.

Queue managers may be configured with the identity of the *home gateway*, i.e. the queue manager through which incoming messages are to be routed. For a device this is the single MQe queue manager that holds messages awaiting delivery to the device. For a gateway it is a list of MQS queue managers and associated transmission queues that hold inbound messages for the MQe network.

Queue managers are also associated with one or more *protocol adapter classes*, in order to support various communications protocol, such as HTTP or native TCP/IP.

Queue managers provide applications with *get*, *put*, *browse*, *wait*, *listen* and *delete* operations. The message get operation will retrieve the first available message, removing it from the queue, unless a *filter* is specified (a filter being a field object that is matched for equality). In this way any fields in the message can be used for selective retrieval. Messages can be *deleted* from a queue without being retrieved through specification of the UID. Queues may also be *browsed* for messages under the control of a filter – browsing retrieves all the message objects matching that filter, but leaves them on the queue. *Browsing under lock* is also supported – which has the additional feature of locking the matching messages on the queue. Messages may be locked individually or using a filter – with the locking operation returning a *lock id*. Locked messages can be got or deleted only if the lock is supplied. Applications can *wait* for a specified time for messages to arrive on a queue, optionally with a filter to identify those messages of interest. Alternatively applications can *listen* for MQe message events, again optionally with a filter.

Administration

Administrative functions control access to the MQe environment. Queue manager and queue-level functions are controlled by administration, as is certification. Message-related functions are regarded as the responsibility of applications. Administration is provided through an administrative interface that handles the generation and receipt of *admin messages* and is designed such that local and remote administration is handled in an identical manner. Requests for administration are sent to an admin queue of the target queue manager; replies may be generated. The admin queue itself is unusual in that it does not have a persistent store – all messages are processed on arrival. Only one such admin queue exists per queue manager and access is subject to any authenticator that may have been associated with the queue. Once an admin message has been received, a reply message is optionally created from it in which the results can be stored, the administrative action is performed, and then the results may be sent back to the originator. The security attributes of the admin queue are used to control access to administrative function, the level of cryptor chosen defining the level of confidentiality required.

Security

MQe provides an integrated set of security features, enabling the protection of message data both when held locally and when it is being transferred. These features provide protection in three different categories:

- Local security - local protection of message (and other) data.
- Queue-based security - protection of messages between initiating queue manager and target queue.
- Message-level security – message level protection of messages between initiator and recipient.

MQe local- and message-level security are used internally by MQe, but 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 application of an *attribute* object. Depending on the category, the attribute is either explicitly or implicitly applied.

Local security facilitates the protection of MQe message data locally (or a fields object or descendant). This is achieved by creating an attribute with an appropriate symmetric cryptor and compressor, creating and setting up an appropriate key and explicitly attaching it to the attribute and the attribute to the message.

MQe queue-based security allows an application to leave all message security considerations to MQe itself. Queues have authentication, encryption and compression characteristics and these are used to determine the level of security needed to protect message flows (as well as for persistent storage).

Rules

Rules are Java classes used to customise the behaviour of MQe when various state changes occur. Default rules are provided where necessary, but these may be replaced with application- or installation-specific rules to meet customer requirements. The rule types supported differ in how they are triggered – not what they can do; rules contain logic and can therefore perform a wide range of functions. Attribute rules are given control whenever change of state is attempted, for example, a change of authenticator, cryptor or compressor. Bridge rules are given control when the MQe to MQS bridge code has a change of state, for example, on message arrival. Other rules are associated with remote access dialing, with queues and with the queue manager.

Connection style

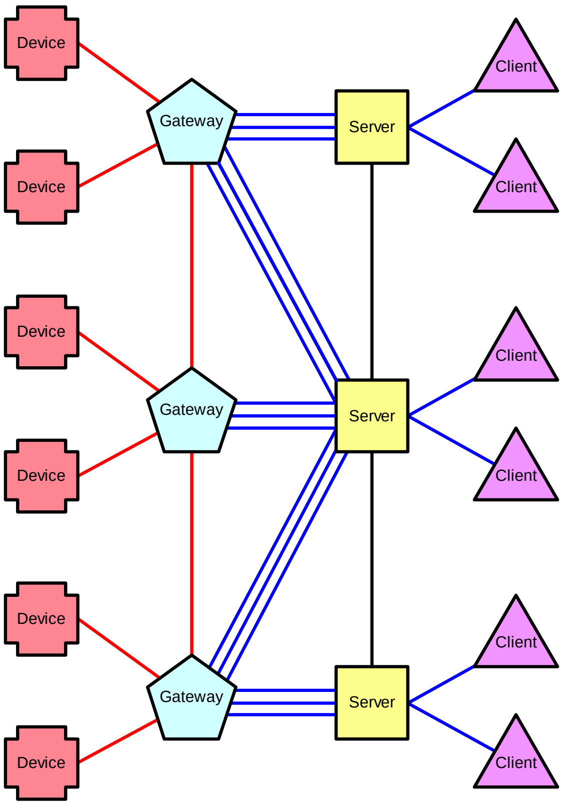
MQe can support client- and/or peer-to-peer operation. A *client* is able to initiate communication with a server; a *server* is only able to respond the requests initiated by a client. In *peer-to-peer* operation, the two peers can initiate flows in either direction. Connection styles require that the appropriate elements of MQe to be active, these being the channel listener (which listens for incoming connection requests) and the channel manager (which supports logical multiple concurrent communication pipes between end points)

Mixed MQe and MQS networks

Although an MQe network can exist standalone, without the need for an MQS server or network, in practice MQe will often be used to complement an existing MQS installation, extending its reach to new platforms and devices, or providing advanced capabilities such as queue-based security or synchronous messaging. From an

MQe application perspective, MQS queues and queue managers can be regarded as just additional remote queues and queue managers, however there are a number of functional restrictions that exist. This is a consequence of the fact that these queues are not accessed directly through MQe dynamic channels and an MQe queue manager, but instead require the involvement of an MQe gateway. This gateway can either talk to an MQS queue manager directly, through MQS client channels if it is so configured, or indirectly, i.e. through MQS client channels to an intermediate MQS queue manager and then onwards through MQS message channels to the target queue manager.

An MQe gateway can be connected to MQS servers through one or more MQS client channels. In order to maximise throughput, in any gateway, a pool of channels is maintained to a given server and these are re-used as required. Thus one MQe gateway not only can connect to multiple MQe devices and gateways but also can connect to multiple MQS servers. Moreover, there is no restriction on how many gateways in an MQe network can be so configured. The following figure shows a possible network configuration.



4Figure 4: An MQe and MQS messaging network

Messages from the MQe network destined for delivery to an MQS destination are passed by the gateway to the relevant server. The gateway remote queue manager definitions provide (in an exactly analogous way to that used for MQe destinations) either the information needed for a client channel connection to that destination or the identity of the intermediate queue manager to which they should be sent. In this way outgoing messages can be routed all over an MQS network, directly and indirectly.

Messages from the MQS network must be sent to specified transmission queues dedicated to MQe at designated servers. Conventional MQS remote queue manager definitions are used to ensure correct routing to these queues. No message channels are defined at the server to remove messages from these transmission queues – this task is performed by the MQe gateway using client channels. A gateway can receive messages from one or more transmission queues – and these may exist at one or more servers.

Summary

MQSeries Everyplace is a member of the MQSeries family providing a complementary messaging system to MQSeries Messaging. It supports a range of devices, from the very smallest through to powerful application servers. It has a small footprint with minimal set-up and associated administration. It offers integrated and comprehensive security features that make it well suited for use outside of corporate firewalls. MQSeries Everyplace integrates seamlessly into a traditional MQSeries network.