MQSeries® Everyplace for Multiplatforms

IBM

Introduction

Version 1.1

GC34-5843-01

MQSeries® Everyplace for Multiplatforms

Introduction

Version 1.1

GC34-5843-01

Take Note!

Before using this information and the product it supports, be sure to read the general information under "Appendix. Notices" on page 57

Licence warning

MQSeries Everyplace Version 1.1 is a toolkit that enables users to write MQSeries Everyplace applications and to create an environment in which to run them.

The licence conditions under which the toolkit is purchased determine the environment in which it can be used:

*If MQSeries Everyplace is purchased for use as a **device** (client) it may **not** be used to create an **MQSeries Everyplace channel manager**, or an **MQSeries Everyplace channel listener.**, or an **MQSeries Everyplace bridge***

*The presence of an **MQSeries Everyplace channel manager**, or an **MQSeries Everyplace channel listener**, or an **MQSeries Everyplace bridge** defines a **gateway** (server) environment, which requires a gateway licence.*

Second Edition (October 2000)

This edition applies to MQSeries Everyplace Version 1.1 and to all subsequent releases and modifications until otherwise indicated in new editions.

This document is continually being updated with new and improved information. For the latest edition, please see the MQSeries family library Web page at `http://www.ibm.com/software/ts/mqseries/library/`.

Contents

Figures

Tables

About this book

This book is a general introduction to MQSeries Everyplace for Multiplatforms product (generally referred to in this book as MQSeries Everyplace). It covers the product concepts and its relationship to other MQSeries products.

For detailed information on the MQSeries Everyplace API and how to use it to create MQSeries Everyplace applications, see the *MQSeries Everyplace for Multiplatforms Programming Reference* and the *MQSeries Everyplace for Multiplatforms Programming Guide*.

| For information relating to MQSeries Everyplace for Multiplatforms clients see *MQSeries*
| *Everyplace for Multiplatforms Native Client Information*

| For MQSeries Everyplace for Multiplatforms installation procedures see *MQSeries*
| *Everyplace for Multiplatforms Read Me First*

This document is continually being updated with new and improved information. For the latest edition, please see the MQSeries family library Web page at `http://www.ibm.com/software/ts/mqseries/library/`.

Who should read this book

This book is intended for anyone interested in using secure messaging on lightweight devices such as sensors, phones, Personal Digital Assistants (PDAs) and laptop computers.

Prerequisite knowledge

No previous knowledge is required to read this information, but an initial understanding of the concepts of secure messaging is an advantage.

If you do not have this understanding, you may find it useful to read the following MQSeries books:

- *MQSeries An Introduction to Messaging and Queuing*
- *MQSeries for Windows NT® V5R1 Quick Beginnings*

These books are available in softcopy form from Book section of the online MQSeries library. This can be reached from the MQSeries Web site, URL address `http://www.ibm.com/software/ts/MQSeries/library/`

Terms used in this book

The following terms are used throughout this book:

MQSeries family
refers to the collection of MQSeries products described in "Chapter 3. The MQSeries family" on page 5.

MQSeries Messaging
refers to the four messaging product groups described in "Chapter 3. The MQSeries family" on page 5.

MQSeries
refers to the following three MQSeries Messaging product groups:

- Distributed messaging
- Host messaging
- Workstation messaging

MQSeries Everyplace
refers to the fourth MQSeries Messaging product group, pervasive messaging.

Device A computer of any size that is running MQSeries Everyplace programs but *does not have* an ***MQSeries Everyplace channel manager*** or an ***MQSeries Everyplace channel listener*** object installed.

Note: For licensing purposes *device* is synonymous with *MQSeries Everyplace client*.

Gateway
A computer of any size that is running MQSeries Everyplace programs and *has* an ***MQSeries Everyplace channel manager*** or an ***MQSeries Everyplace channel listener*** object installed.

Note: For licensing purposes *gateway* is synonymous with *MQSeries Everyplace server*.

Summary of Changes

This section describes changes to this edition of *MQSeries Everyplace for Multiplatforms Introduction*. Changes since the previous edition of the book are marked by vertical lines to the left of the changes.

Changes for this edition (GC34-5843-01)

The following information has been added:

- Information for using MQSeries Everyplace on AIX® and Solaris.
- Storage requirements.
- Readers comment form.

Chapter 1. Overview

MQSeries Everyplace is a member of the MQSeries family of business quality messaging products. It is designed to satisfy not only the messaging needs of lightweight devices, such as sensors, phones, PDAs (Personal Digital Assistant) and laptop computers, but also the demands of mobile attachment and the requirements that arise from the use of fragile communication networks. It provides the standard MQSeries once-only assured delivery, and exchanges messages with other family members. Since many MQSeries Everyplace 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 accordingly MQSeries Everyplace is optimized for system footprint and protocol efficiency. It does not offer identical capabilities to the other messaging members of the MQSeries family but does provide seamless inter-operation. MQSeries Everyplace has extensive provision for mobility, roaming, local and remote message access, security and support for messaging over unreliable networks.

MQSeries Everyplace is a member of the IBM® pervasive computing family and is consequently designed to integrate well with other IBM pervasive and wireless products.

Notes for Version 1.1

- Version 1.1 of MQSeries Everyplace is a toolkit that allows users to write MQSeries Everyplace applications and to create an environment where they can be run.
- In this release, the deployment of MQSeries Everyplace to pervasive devices is the responsibility of the application and solution provider.

Chapter 2. Prerequisites

Table 1 shows the software environments that can be used to run MQSeries Everyplace Version 1.1.[1]

Table 1. Version 1 supported software environments

	Operating system
Device	EPOC
	Palm OS
	Windows CE
	Windows® 95 Windows 98 Windows NT v4 Windows 2000 AIX 4.3 Solaris 7.0
Gateway	Windows NT v4 Windows 2000 AIX 4.3 Solaris 7.0

Notes:

1. Version Version 1.1 is supplied in Java for use across all platforms that support Java.
2. A limited function client, that provides only synchronous access to remote queues, is available as a C codebase for use on the Palm OS only.
3. Java 1.1, at the latest level available for the platform, is recommended. See the MQSeries Everyplace Web site (`www.ibm.com/software/mqseries/everyplace`) for details of the levels of Java that have been tested.
4. The following platforms are supported on the basis that the Java Virtual Machine environment is equivalent to that on supported platforms. If you find a problem on any of these platforms, but you cannot reproduce it on one of the directly supported platforms, you should refer the problem to the JVM owner as a compatibility issue.
   - HP-UX
   - Linux (Intel 32-bit)
   - AS/400
   - OS/2

Table 2 on page 4 shows the storage required to run MQSeries Everyplace.

1. MQSeries Everyplace device code can be run on any device that runs Java®, but it has been tested only with the operating systems listed in Table 1.

Table 2. MQSeries Everyplace storage requirements

Platform	**Standard Gateway**	**High Security Gateway**	**C Client**
Windows NT (file system = NTFS)	9MB	9.5MB	—
AIX	10MB	11MB	—
Solaris	9.5MB	10MB	—
Palm	—	—	88KB

Chapter 3. The MQSeries family

The MQSeries family includes many products, offering a range of capabilities, as illustrated in Figure 1

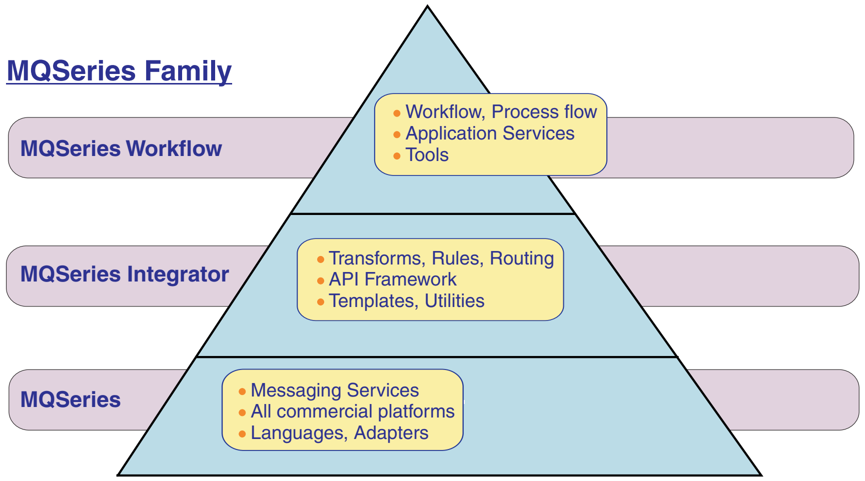

Figure 1. The MQSeries family

- **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 products take advantage of the connectivity provided by the MQSeries messaging layer.

MQSeries family messaging is supplied by both MQSeries and MQSeries Everyplace products; each being designed to support one or more hardware server platforms and/or associated operating systems. Given the wide variety in platform capabilities, these individual products are organized into product groups, reflecting common function and design. Four such product groups exist:

- **Distributed messaging:** MQSeries for Windows NT, AIX, AS/400®, HP-UX, Sun Solaris, and other platforms
- **Host messaging:** MQSeries for OS/390®
- **Workstation messaging:** MQSeries for Windows 3.1, 95, 98
- **Pervasive messaging:** MQSeries Everyplace

Messaging itself, irrespective of the particular product or product group, is based on queue managers. Queue managers manage queues that can each 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, that is one owned by a remote queue manager, the message is transmitted over channels 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 decouple the sending application from the receiving application, queuing messages at intermediate points 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 network configurations, all of which involve clients and servers[2]some examples of which are illustrated in Figure 2.

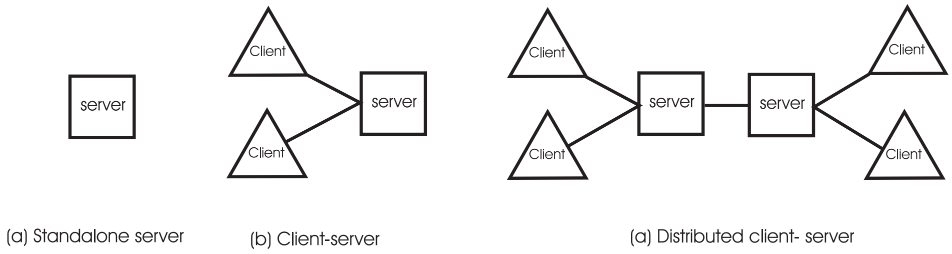

Figure 2. Simple host and distributed configurations

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

The distributed client-server configuration shows a more complex case, with multiple servers involved. In these configurations servers exchange messages through message channels. Message channels are unidirectional, 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 communications are not available.

2. Note that these terms have very specific meanings within MQSeries that do not always correspond to their more common usage. In this document they are always used with their MQSeries semantics.

MQSeries workstation messaging products offer a subset of these configuration options. Instead of servers, they support workstations that have a queue manager but do not support the attachment of clients. However, workstations can attach to other workstations, and also to servers, through MQSeries message channels. Thus workstations are typically regarded as lightweight servers, and are used in place of clients where an asynchronous capability is required.

Two typical workstation configurations are shown in Figure 3. In (b) the workstation applications can run independently of the servers and clients:

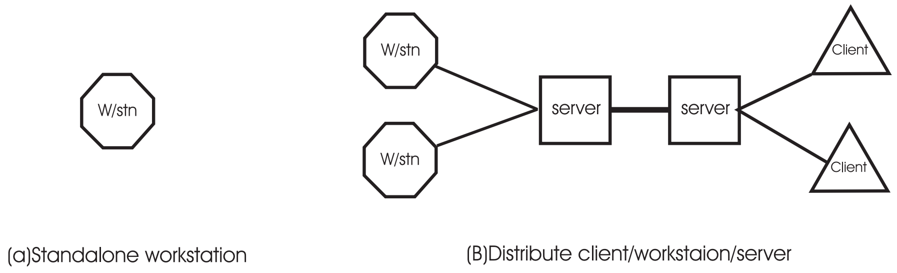
W/stn
W/stn
W/stn
server
server
Client
Client
(a)Standalone workstation
(B)Distribute client/workstaion/server

Figure 3. Typical workstation configurations

The pervasive messaging product MQSeries Everyplace supports configurations through the provision of devices and gateways.

The MQSeries Everyplace device is a computer running MQSeries Everyplace code *without a channel manager*. This means that a device is restricted to communicating with only one other device or gateway at a time. MQSeries Everyplace devices can range from the very small (such as a sensor on an oil pipeline), through larger devices (such as a telephone, personal data assistant (PDA), or laptop computer), up to desktop machines and workstations. Frequently such device computers are known as pervasive devices, though this implies size and capability restriction that is not present in the product.

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

To a first approximation, devices combine many of the attributes of clients and servers. They can be configured with a full queueing capability making them capable of asynchronous operation. They can also access queues held remotely, a feature that has some similarities with client access to server queues. Unlike servers, devices cannot attach clients. Devices can communicate directly with each other, through peer-to-peer messaging capability. Devices also communicate through channels but these channels are unique to MQSeries Everyplace and are called *dynamic channels* in order to distinguish them from the MQSeries client channels and MQSeries message channels. Dynamic channels are bidirectional and support the full range of functions provided by MQSeries Everyplace, including both synchronous and asynchronous messaging.

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

Some typical pervasive configurations are shown in Figure 4

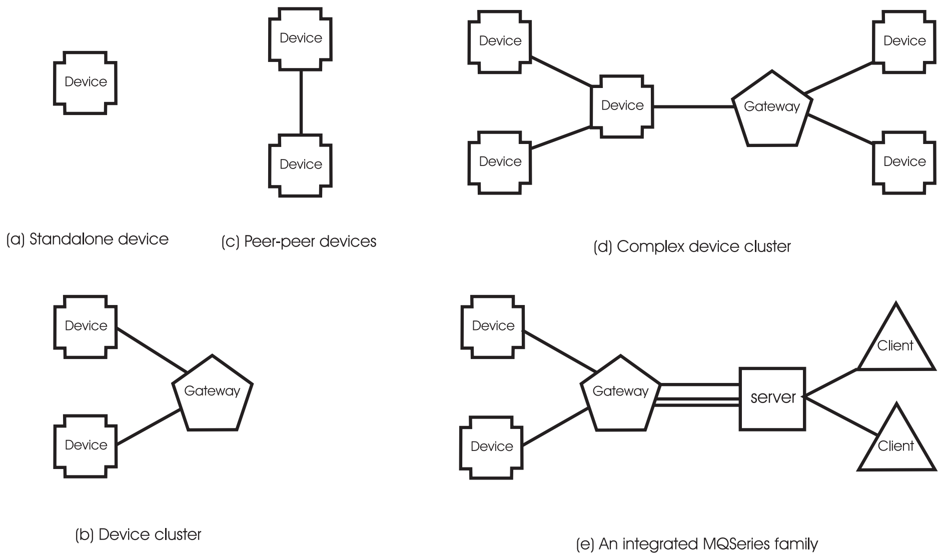

Figure 4. Typical device configurations

In Figure 4 (b) a gateway is used to connect devices together. A feature of gateways is that they can handle multiple simultaneous incoming connection requests, as opposed to devices that can only handle one such request at a time. Both gateways and devices can make multiple simultaneous outgoing requests. If in configuration (b) a device had been used in place of the gateway, then the two terminal devices would have had to take turns in contacting this intermediate device, though it could have contacted them simultaneously. In (d) a device and a gateway are both used to link devices. In (e) a gateway is used to link a network of devices to an MQSeries server, a configuration in which messages can flow between all constituents, devices, gateways, servers, workstations and clients. The most important characteristics of these components are shown in Table 3 on page 9.

Table 3. MQSeries Everyplace and MQSeries elements

Component	**Characteristics**	**Offered by**
Device (MQSeries Everyplace)	Provides assured messaging for applications through dynamic channels allows both synchronous local and remote queue access allows asynchronous delivery to remote queues restricted to handling one incoming request at a time	Pervasive
Client (MQSeries)	Provides assured messaging for local applications requires a synchronous client channel connection to a server allows synchronous access to queues on the attached server only allows asynchronous delivery to remote queues via the attached server	Distributed Host
Gateway (MQSeries Everyplace)	Provides assured messaging for applications through dynamic channels allows both synchronous local & remote queue access allows asynchronous delivery to remote queues can handle multiple incoming requests at a time Supports the attachment of multiple MQSeries servers through client channels	Pervasive
Server (MQSeries)	Provides assured messaging for applications through message channels allows synchronous local queue access allows asynchronous delivery to remote queues Supports the attachment of multiple MQSeries clients through client channels	Distributed Host
Workstation (MQSeries)	Provides assured messaging for applications through message channels allows synchronous local queue access allows asynchronous delivery to remote queues	Workstation

Chapter 4. Requirements

This chapter describes the requirements that have shaped the MQSeries Everyplace design and implementation.

Capabilities

MQSeries Everyplace extends the messaging scope of the MQSeries family:

- By supporting low-end devices, such as PDAs, telephones, and sensors, allowing them to participate in an MQSeries messaging network. It also supports intermediate devices such as laptops, workstations and certain distributed platforms. MQSeries Everyplace offers the same quality of service, once only assured delivery of messages, and permits the exchange of messages with other family members.
- By providing extensive security features to protect messages, queues and related data, whether in storage or in transmission.
- By operating efficiently in hostile communications environments where networks are unstable, or where bandwidth is tightly constrained. It has an efficient wire protocol and automated recovery from communication link failures.
- By supporting the mobile user, allowing network connectivity points to change as devices roam. It also allows control of behavior in conditions where battery resources and networks are failing or constrained.
- By operating through suitably configured firewalls
- By minimizing administration tasks for the user, so that the presence of MQSeries Everyplace on a device can be substantially hidden. This makes MQSeries Everyplace a suitable base on which to build utility-style applications.
- By being easily customized and extended, through the use of application-supplied rules and other classes that modify behavior, or through the sub-classing of the base object classes, for example, to represent different message types.

Applications

There is no restrictive list of possible MQSeries Everyplace applications since the choices are many and varied, but a substantial number may be expected to be custom applications developed for particular user groups. The following list gives some examples of those that have been considered:

- **Consumer applications:** supermarket shopping from home using a PDA, gathering of travellers 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 professional (insurance agent), rapid publication of proof of customer receipt for parcel delivery companies, fast-food waiter exchanging information with the kitchen, golf tournament scoring, mobile secure systems messaging systems for the police, job information to utility workers in situations where communication is frequently lost, domestic meter reading.

- **Personal productivity:** mail/calendar replication, database replication, laptop downsizing

Customer requirements

Requirements that have influenced the design of MQSeries Everyplace include:

- **Administration:** minimal setup and maintenance; support of both local and remote administration; an ability to extend and customize the administration functions to meet the needs of particular applications; an emphasis on automatic discovery and recovery; the provision of independent administration elements that can be selectively used.
- **Communications:** a very efficient wire protocol; minimal headers; no compulsory fields in messages (excepting a unique identifier); the ability to change the data encoding; compression, encryption and authentication support; end-to-end negotiation of compression and security characteristics; an ability to easily pass through firewalls; pluggable communications adapters.
- **Compatibility:** MQSeries quality of service and seamless messaging interchange; the ability to communicate to existing MQSeries systems without application change; flexible control of message interchange between MQSeries and MQSeries Everyplace .
- **Footprint:** substantially below 100K bytes for a minimally configured device system.
- **Function:** synchronous and asynchronous messaging capabilities, access to messages held in either local or remote queues; the ability to use any field in the message for selective retrieval; selective control of the backing medium for a queue.
- **Rule support:** control of many aspects of behavior through rules, for example, when to send messages, how often to retry a communication link, what to do with a message that is too big, or how to behave when a target queue is full.
- **Security:** full support for security, authentication and non-repudiation; message-level and queue level security; protection of the messaging system from security attacks; pluggable security using industry standard algorithms; ability to integrate with operating system user credentials; the capability to comply with the national security requirements, allowing security support to change as messages cross country boundaries.

Chapter 5. Product concepts

Introduction

The fundamental elements of the MQSeries Everyplace programming model are messages, queues and queue managers. MQSeries Everyplace messages are objects that contain application-defined content. When stored, they are held in a queue and such messages may be moved across an MQSeries Everyplace 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 and typically 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 a registry.

Applications on devices can make use of any or all of the APIs or functions available on the device, they are not restricted to the MQSeries Everyplace programming interfaces. Through dynamic channels MQSeries Everyplace devices may be connected to other MQSeries Everyplace devices and/or to an MQSeries Everyplace gateway.

Applications on gateways can also make use of any or all of the APIs or functions available on the gateway, not just the MQSeries Everyplace programming interfaces. Through dynamic channels, a gateway can be connected to other gateways and/or to MQSeries Everyplace devices. Through MQSeries client channels, a gateway may be connected to one or more MQSeries servers (but not to other MQSeries Everyplace gateways). Both MQSeries Everyplace and MQSeries can coexist on a single machine although the presence of MQSeries is entirely optional.

The capabilities of full function devices and gateways are the same, except that:
- Gateways can handle multiple simultaneous inbound requests (from other devices and/or gateways)
- Gateways can simultaneously interface to multiple MQSeries servers

Dynamic channels support the following network connections:
- Dial-in connections
- Permanent connections, for example a conventional LAN, leased line, infrared or wireless LAN.

The communications protocols are implemented by a set of 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.

Queues are individually mapped to storage media through another set of adapters. Thus a queue can be stored in the file system or in memory, depending on the chosen adapter.

The MQSeries Everyplace programming interface is designed so that applications can be written with no dependence on the location of queues. Thus a program designed to access local queues should be able to run unchanged from a remote queue manager (subject to satisfying any security considerations in force and accepting that certain MQSeries Everyplace operations are not supported against remote queues). This independence includes any use of administrative functions.

Message objects

MQSeries Everyplace message objects differ fundamentally from the messages supported by MQSeries. In MQSeries, messages are byte arrays, divided into a message header and a message body. The message header is understood by MQSeries and 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 is not understood.

In contrast, messages in MQSeries Everyplace are *message objects* inherited from an MQSeries Everyplace object known as the *fields object.* Messages are true objects, with no concept of a header or a message body. The real nature of the message object becomes clearer only when the base fields object is understood. These fields objects, used extensively in MQSeries Everyplace , are an accumulation of fields, where a field comprises a name, a data type and the data itself. Field names are ASCII character strings (barring a number of reserved characters) of unlimited length.

Field types may be:
- **ASCII** string or a dynamic array of ASCII strings
- **Boolean** value
- **Byte**, fixed array, or a dynamic array of byte values
- **Double floating point**, 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), fixed array, or a dynamic array of integers
- **Long integer** (8 byte, fixed array, or a dynamic array of long integers
- **Short integer** (2 byte), fixed array, or a dynamic array of short integers
- **UNICODE** string or a dynamic array of UNICODE strings

Fields objects have a *type* where the type corresponds to the programming object class name. Descendants of this object class are used by application programs as message objects and their type is used by MQSeries Everyplace to instantiate the correct object class when required, for example after a message object has been flowed across a channel.

Fields objects supply a number of methods, for example, fields can be enumerated or their existence can be verified. Likewise fields objects can be compared for equality. They have the capability to dump and restore their field items to and from a byte array, used for example to provide the data for transmission of the object over a link, and to restore the object after transmission. The dump and restore methods can be overridden to allow fields objects to serialize themselves in other ways, for example, to query a database for their field content at the time of transmission.Table 4 lists the properties of fields objects and/or their constituent fields.

Table 4. Fields objects and their constituent field properties

Property	Presence	
	Fields objects	**Fields**
Associated attribute object	Optional	
Constituent field(s)	yes	
Hidden		yes
Name		yes
Type	yes	yes
Value		yes

The hidden property of a field enables that field to be ignored for the purposes of a comparison operation.

Attribute objects contain the mechanisms to perform authentication, encryption and compression and may be associated with fields objects.

- **Authentication:** controls access
- **Compression:** reduces storage requirements (for transmission and/or storage)
- **Encryption:** protects the contents when the object is dumped (and allows restoration)

Attribute objects are fundamental to the MQSeries Everyplace security model and allow selective access to content and the protection of data on backing storage amongst many other uses.Table 5 lists the properties of attribute objects. The Rule value, when present, controls which operations are allowed.

Table 5. Attribute object properties

Property	Presence
Cryptor	optional (may be required in some circumstances)
Authenticator	optional
Compressor	optional
Rule	optional
Type	optional

Message objects are derived from fields objects and include a UID (unique identifier) that is generated by MQSeries Everyplace. This UID uniquely identifies a message object and is constructed from:

- **Name** of the originating queue manager (added by the queue manager on receipt of the object). This name must be globally unique.
- **Time** that the message object was created (added at creation)

Message objects have the basic properties listed in Table 6 in addition to those that they inherit as fields objects.

Table 6. Message object properties

Property	**Explanation**
Msg_OriginQMgr	The name of the queue manager that sent the message
Msg_Time	Time the message object was created by the application

These two properties make up the unique identifier (UID) of the message object.

No other information is required in a message destined for another MQSeries Everyplace queue manager, though other fields are usually included to carry the messages information content. Typically messages are descendants of the base message object class and thus have additional fields as befits their purpose. Of these additional fields, a number will be common to a wide range of applications, such as 'reply to queue manager. Accordingly MQSeries Everyplace provides some measure of support for them.

Table 7 lists the supported fields.

Table 7. Message object fields for which provision is made

Field name	**Usage**
Msg_CorrelID	Byte string typically used to correlate a reply with the original message
Msg_ExpireTime	Time after which message may be deleted (even if it is not delivered)
Msg_LockID	The key necessary to unlock a message
Msg_MsgID	Used by the application for correlation with the original message
Msg_Priority	Priority of the message
Msg_ReplyToQ	Name of the queue to which a message reply should be addressed
Msg_ReplyToQMgr	Name of the queue manager to which a message reply should be addressed
Msg_Resend	Indicates that the message is a re-send of a previous message
Msg_Style	Distinguishes commands from request/reply etc.

In all cases a defined constant is available that allows the field name to be carried in a single byte. For some fields more extensive provision is made - for example: priority (if present) affects the order in which messages are transmitted; correlation id triggers indexing of a queue on those field values for fast retrieval; expire time triggers the expiry of the message, and so on.

Message objects have a number of methods defined on them, for example the ability to extract the message UID, the originating queue manager name and the object creation time. Other useful methods are inherited from the fields object class, for example, various methods for getting and putting field values. Of particular interest is the *dump* method, which is used to dump the object data to a byte string. MQSeries Everyplace calls this method when a message is to be saved to persistent storage and when it is to be transmitted over a dynamic channel. By this means the message object itself is responsible for determining the external representation of its data value and this can be exploited in many ways. For example, an object can simply dump the values of its constituent fields or it could choose to query a database instead. The complementary *restore* method offers similar control possibilities when an object is recreated from its dumped format. Note that if the message object has an attached attribute object, the attribute's data encoder, encryptor and compressor are invoked on dump; similarly the decoder, decryptor and decompressor are invoked on restore.

When MQSeries Everyplace flows a message object, in order to reduce footprint over the wire, it does not flow the associated class file. Accordingly the appropriate message class must be available at each queue manager where the message object is to be instantiated.

The default message object dump method has been optimized to minimize the size of the generated byte string in order to achieve efficient message storage and transmission.

Dump data format

The default dump data format encodes fields as follows:

```
{Length  Identifier  Fence  {Data}}  {Length  Identifier  Fence  {Data}}  { ...}
```

where:

- *Data:* the data value. Integers are compressed with leading 0s and Fs removed. Booleans have no associated data bytes
- *Fence:* a special byte delimiting the boundary between the identifier and the optional Data item. This byte also indicates the Data item type
- *Identifier:* holds the field name in a variable length ASCII string of bytes, terminated with an end byte
- *Length:* indicates the length of the data field. A variable number of bytes between 1 and 4 are used. The first byte has the first two bits reserved to indicate the length of the length field. Lengths in the range 0 - 1,073,741,823 are supported

This results in a highly compact data stream. Further savings can be achieved by compressing the data. XOR compression with a previous byte stream might be expected to produce good results but, because of the variable nature of these fields and the fact that the order of the fields can change, a simple XOR does not always produce the desired effect. MQSeries Everyplace includes an intelligent XOR, working on a field-by-field basis, that is much more likely to improve compression.

Queues

Queues are typically used to hold message objects pending their removal by application programs. Like messages, queues also derive from the fields objects. Direct access by applications to the queue object is not permitted[3]; instead the queue manager acts as an intermediary between application programs and queues. Queues are identified by name and the name can be an ASCII character string of unlimited length[4]but must be unique within a particular queue manager.

MQSeries Everyplace supports a number of different queue types:

Local queues

Local queues are used by applications to store messages in a safe and secure manner. They have a message store that is accessed via an adapter class, typically the disk adapter class. However a memory adapter class is supplied with MQSeries Everyplace that holds the message store in memory for fast access (at the cost of message loss if the system crashes). By creating the appropriate adapter messages can be stored anywhere, on a queue-by-queue basis, for example in a relational database, on a writable CD etc. Local queues can be used on or offline, that is connected or not to a network. Access and security are owned by the queue and may be granted for use by a remote queue manager, when connected to a network, allowing others to send or receive messages to/from the queue. Local queue access is always synchronous.

Remote queues

Remote queues do not reside in the local environment; instead a definition exists locally that identifies the owning queue manager and the real queue. Remote queues may be accessed either synchronously or asynchronously. If there is a definition of the remote queue held locally, then the mode of access is based on this definition. If not, then queue discovery occurs, such that the characteristics are discovered and the mode of access is forced to be synchronous.

Synchronous queues are queues that can only be accessed when connected to a network that has a path to the owning queue manager. If the network is not established then the operations such as put, get, and browse, (see Table 11 on page 24) cause an exception to be raised. The owning queue controls the access permissions and security requirements needed to access the queue. It is the application's responsibility to handle any errors or retries when sending or receiving messages, and in this case MQSeries Everyplace is no longer responsible for once-only assured delivery.

Asynchronous queues are queues that can have messages put into them but not retrieved. If the network connection is established then the messages are sent to the owning queue manager and queue. If however the network is not connected the messages are stored locally until there is a network connection

3. Direct access is permitted inside a queue rule.

4. For interoperability it is recommended that the MQSeries naming restrictions are observed, including a maximum name length of 48 characters. The length may also be restricted by the file system you are using.

and then the messages are transmitted. This allows applications to operate on the queue when the device is offline; it does however require that these types of queue have a message store in order to temporarily store messages.

Store and forward queues
This type of queue stores messages until they can be forwarded to the next (but not necessarily the owning) queue manager. This type of queue is normally (but not necessarily) defined on a gateway and the device would have to collect its messages when it is connected to the network. Store and forward queues may hold messages for many target queue managers or there may be one store and forward queue per target queue manager. When a sender wishes to send a message to a recipient that may be disconnected the sender still addresses the message to the recipient’s queue manager/queue; the intermediate server detects that the recipient is not connected and stores the message in its local message store. The sending application does not require any changes to send a message to a connected or disconnected target queue.

Home server queues
This type of queue normally resides on a device (assumed to be occasionally connected) and points to a store and forward queue on a queue manager known as the home server. The home server queue pulls messages from the home server whenever the device connects to the network. When the queue has pulled a message from the server it gives it to the local queue manager using the putMessage and confirmputMessage method calls (see “Queue manager operations” on page 23). It is then the responsibility of the queue manager to place the message in the correct local queue. The pull method of getting messages from the server can be more efficient in terms of flows over the network than the server pushing the messages; this is because the home server queue uses the acknowledgement of the first message as the request for the next one message (if any), whereas the server push would require a request/response to send the message and a second request/response for the confirmation flow. Home server queues normally having a polling interval set that causes it to check for any pending messages on the server whilst the network is connected. The poll interval is an administration configuration option.

Administration queues
This type of queue receives MQSeries Everyplace administration messages. An optional administration message reply queue can also be used to receive replies to administration messages sent by the MQSeries Everyplace system. The administration queues do not understand how to perform the administration, they handle the messages that encapsulate the administration details.

MQSeries bridge queues
This is a specialized form of remote queue with the definition on a gateway and the target queue on an MQSeries queue manager. This form of queue provides a pathway between the MQSeries Everyplace and the MQSeries environments. Transformers are used to perform any necessary data or

message reformatting. A very basic transformer is supplied with MQSeries Everyplace; programmers are expected to customize this transformer to suit their own requirements.

MQSeries Everyplace stores data securely on queues, ensuring that messages are physically written to the media and not simply buffered by the operating system. However MQSeries Everyplace does not independently log changes to messages and queues. If recovery from media failure is required then hardware solutions must be deployed, such as the use of RAID disk systems. Alternatively the queue must be mapped into recoverable storage such as certain database subsystems.

MQSeries Everyplace does not require that a queue manager has defined queues. However provision is made for four system queues, if required:

- **AdminQ:** required for the receipt of administration messages
- **AdminReplyQ:** optionally used for receiving replies to administration messages
- **DeadLetterQ:** used to store messages that cannot otherwise be delivered
- **SYSTEM.DEFAULT.LOCAL.QUEUE:** a queue that shares a common name with the mandatory system queue on MQSeries servers

Queue properties are shown in Table 8, note however that not all the properties shown apply to all the queue types:

Table 8. Queue properties

Property	**Explanation**
Admin_Class	Queue class
Admin_Name	Ascii queue name
Queue_Active	Indicates that the queue is active
Queue_AttRule	Rule class controlling security operations
Queue_Authenticator	Authenticator class
Queue_BridgeName	Owning MQSeries-bridge name
Queue_ClientConnection	Client connection name
Queue_CreationDate	The date that the queue was created
Queue_Compressor	Compressor class
Queue_Cryptor	Cryptor class
Queue_CurrentSize	Number of messages on the queue
Queue_Description	Unicode description
Queue_Expiry	Expiry time for messages
Queue_ FileDesc	The location where the queue is stored
Queue_MaxMsgSize	Maximum length of messages allowed on the queue
Queue_MaxQSize	Max. no. of messages allowed
Queue_Mode	Synchronous or asynchronous
Queue_MQQMgr	MQSeries queue manager proxy

Table 8. Queue properties (continued)

Property	Explanation
Queue_Priority	Priority to be used for messages (unless overridden by a message value)
Queue_QAliasNameList	Alternative names for the queue
Queue_QMgrName	Queue manager owning the real queue
Queue_RemoteQName	Remote MQSeries field name
Queue_Rule	Rule class for queue operations
Queue_TargetRegistry	The target registry type
Queue_Transporter	Transporter class
Queue_Transformer	Transformer class

Administrative functions are used to create and delete queues, and to inquire on or modify their properties.

Queues are not limited to use as a message store. Sub-classed queues can be used in process control application scenarios, for example the queue object could directly control a valve. A message of the right type would cause the valve to be opened, the volume of the flow to be changed etc. An application would not be pulling messages off the queue and performing the action, the queue object would itself controls the action. Other queues could, for example, update spreadsheets or do text to speech conversion. The advantages of this technique are that the security aspects of the queues are still in place and effective, as also is assured messaging. So MQSeries Everyplace would still assure the once-only delivery of the messages, and an associated authenticator and cryptor would guarantee that only the authorized sender of the message could send such messages, with the contents highly secure in transit. No applications would be permitted access to the queue and none would be required.

Queue managers

The MQSeries Everyplace queue manager provides application access to the messages and queues and controls any channels. In MQSeries Everyplace Version 1.1 only one queue manager can be active on a single Java virtual machine at any one time. If there are multiple JVMs on a machine, there can be the same number of queue managers as JVMs. Queue managers are identified by name and the name must be globally unique[5] and an ASCII character string that can be of unlimited length.[6]Queue managers can be configured with or without local queueing. All queue managers support synchronous messaging operations; a queue manager with local queueing also supports asynchronous message delivery.

5. This restriction is not enforced by MQSeries Everyplace or MQSeries, but duplicate queue manager names may cause messages to be delivered to the wrong queue manager.

6. For interoperability it is recommended that the MQSeries queue manager name rules are observed, including limiting the maximum name length to 48 characters. The length may also be restricted by the file system you are using.

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

Asynchronous message delivery[7]
With asynchronous message delivery, the application passes the message to MQSeries Everyplace for delivery to a remote queue. An immediate return is made back to the application after the put operation. MQSeries Everyplace temporarily holds the message locally until it can be delivered. Delivery may be staged, with MQSeries Everyplace responsible for delivery. This mode of operation provides *once-only assured delivery*. See "Asynchronous message delivery" on page 31 for further discussion.

Synchronous message delivery:
Synchronous messaging can be used to address:

- Target queues on an MQSeries Everyplace queue manager routed over an MQSeries Everyplace network
- target queues on an MQSeries queue manager directly attached to an MQSeries Everyplace gateway
- Target queues on an indirectly attached MQSeries server

With synchronous messaging, the application puts the message to MQSeries Everyplace for delivery. MQSeries Everyplace synchronously contacts the target remote queue and places the message. After delivery MQSeries Everyplace returns to the application.

Contact with the remote queue manager may involve MQSeries Everyplace routing through intermediate devices and/or gateways.See "Synchronous message delivery" on page 31 for further discussion.

Thus asynchronous message delivery means that the local application gives the message to MQSeries Everyplace and its delivery onwards from that local queue manager is the responsibility of MQSeries Everyplace . It means that the network and/or the receiving application need not be available. The time of the actual delivery is unknown to the sending application. Synchronous message delivery requires the network to be running but the sending application knows that it has been delivered to the receiving application's queue. The receiving application does not need to be available in either the asynchronous or the synchronous case.

A local queue manager has properties that reflect the local management of queues. It also needs a *connection definition* for each remote queue manager with which it must make contact. Hence connection definitions are sometimes referred to as *remote queue manager definitions*. These definitions may include all the information needed for a direct communication between the queue managers (including a network address), or they may simply indicate that the communication is indirect, going via an intermediate queue manager. In the latter case all that is required is the name of the next hop queue manager.

7. MQSeries Everyplace does not distinguish between the persistent and non-persistent modes offered by MQSeries, only persistent is supported. However, if required, the choice of queue backing storage allows a trade-off to be made between performance and recovery.

Queue managers properties are shown in Table 9 and Table 10.

Table 9. Local queue manager properties

Property	Explanation
QMgr_ChnlAttrRules	Channel attribute rules
QMgr_ChnlTimeout	Channel time-out
Admin_Class	Queue manager class
QMgr_Description	Unicode description
Admin_Name	Queue manager name
QMgr_Rules	Rule class for queue manager operations

Table 10. Connection (remote queue manager) properties

Property	Explanation
Con_Adapter	The adapter file descriptor
Con_AdapterOptions	Adapter options (such as use history)
Con_AdapterParm	ASCII data to be use by an adapter (such as servlet name)
Con_Aliases	Alternative names for the queue manager
Con_Channel	The type of channel that this connection should use
Con_Description	Unicode description
Queue_QMgrName	Owner of the definition
Admin_Name	Queue manager name

Multiple adapters are supported in a connection definition.

Queue manager operations

Queue managers support messaging operations and optionally manage queues. Applications have access to messages through operations performed on a queue manager. Unless a filter is specified, the first available message on the queue is retrieved. A filter is a field object that is matched for equality and any fields in the message can be used for selective retrieval. The get operation, like all message sending and retrieval operations, can optionally be given an attribute object to be used in the encoding and decoding of a message.

In MQSeries Everyplace , as in MQSeries, get is normally a destructive operation. If assured messaging is required between MQSeries Everyplace and the application, then the get followed by confirm method sequence should be used. First a get is issued with a confirm id (its value being chosen by the application) - that operation gets the message but hides it on the queue rather than deleting it immediately. A subsequent confirm operation, specifying the original message UID, indicates that the get was successful for the application, and it is then that the message is deleted. Failure of the get allows the message to be recovered. Put operations behave in a similar manner.

By specifying the UID, messages can be *deleted* from a queue, without being retrieved.

If nondestructive read is required, queues may be *browsed* for messages (optionally under the control of a filter). Browsing retrieves all the message objects that match the filter, but leaves them on the queue. *Browsing under lock* is also supported. This has the additional feature of locking the matching messages on the queue. Messages may be locked individually, or in groups identified through a filter, and the locking operation returns a *lock id*. Locked messages can be got or deleted only if the lock id is supplied. An option on browse allows either the full messages, or only the UIDs, to be returned.

Applications can *wait* for a specified time for messages to arrive on a queue. Optionally a filter can be used to identify those of interest and a *confirm id* can also be specified. Alternatively applications can listen for MQSeries Everyplace message events, again optionally with a filter. Listeners are notified when messages arrive on a queue.

Queues are enabled for messaging operations as shown in Table 11

Table 11. Messaging operations on MQSeries Everyplace queues

	Local queue	**Remote queue**[1]	
		Synchronous	**Asynchronous**
Browse (±lock, ±filter)	Yes	Yes	
Delete	Yes	Yes	
Get (±filter)	Yes	Yes	
Listen (±filter)	Yes		
Put	Yes	Yes	Yes
Wait (±filter)	Yes	Yes	

Notes:

1. The synchronous remote wait operation is implemented through a pole of the remote queue, so the actual wait time is a multiple of the poll time
2. [1]The MQSeries Everyplace MQSeries Bridge supplied with MQSeries Everyplace Version 1.1 only supports the 'put' operation.

Queue managers can optionally load applications (classes) immediately after initiation; similarly they can terminate applications on shutdown. Queue managers raise events to reflect status or error; by default these appear in the event log.

Administration

Administration provides facilities to configure and manage MQSeries Everyplace resources such as queues and connections. Message-related functions are regarded as the responsibility of applications. Administration is enabled through an interface that handles the generation and receipt of administrative messages and is designed so that local and remote administration is handled in an identical manner. Requests are sent to the administration queue of the target queue manager and replies may be received if required. Any local or remote MQSeries Everyplace application program can create and process administration messages directly or indirectly through helper methods.

Administration messages can also be generated indirectly through MQSeries Everyplace Explorer[8], a management tool that provides a graphical user interface for system administration.

The administration queue does not understand how to perform the administration of individual resources; this knowledge is encapsulated in each resource and its corresponding administration message.

Administration messages

Administration messages extend the base MQSeries Everyplace message object.Table 12 lists the message classes provided for administration of MQSeries Everyplace resources. These base administration messages can be sub-classed to provide for the administration for other objects; for example a different type of queue could be managed using a subclass of MQeQueueAdminMsg. The MQSeries Everyplace bridge to MQSeries uses subclasses of the MQeAdminMsg in this way.

Table 12. Administration message classes

Administration message class	Use
MQeAdminMsg	Abstract class used as the basis of all administration messages
MQeQueueManagerAdminMsg	Administration of queue managers
MQeQueueAdminMsg	Administration of local queues
MQeRemoteQueueAdminMsg	Administration of remote queues
MQeAdminQueueAdminMsg	Administration of the administration queue
MQeHomeServerQueueAdminMsg	Administration of home server queues
MQeStoreAndForwardQueueAdminMsg	Administration of store and forward queues
MQeConnectionAdminMsg	Administration of connections between queue managers
MQeClientConnectionAdminMsg	Administration of a bridge client connection object, used to connect to MQSeries
MQeListenerAdminMSg	Administration of a bridge transmission queue listener object, used to collect messages from MQSeries
MQeBridgeAdminMsg	Administration of a bridge to MQSeries
MQeMQBridgesAdminMsg	Administration of a list of MQSeries-bridges
MQeMQQMgrProxyAdminMsg	Administration of a bridge representation of an MQSeries queue manager
MQeMQBridgeQueueAdminMsg	Administration of an MQSeries-bridge queue

The structure of an administration message depends upon its particular class, that is the nature of the resource that it is managing, and the details of the operation to be performed on that resource. Generically however, the administration messages are

8. MQSeries Everyplace Explorer is not included in Version 1.1 but will be available from the MQSeries software download site on the World Wide Web (http://www-4.ibm.com/software/ts/MQSeries/).

structured as shown in Table 13:

Table 13. Generic structure of an administration message

Level 1 fields	**Level 2 and below fields**	**Use**
Admin_Action		Create, delete, inquire, etc.
Admin_Errors		Fields object parent
	Multiple fields	Detailed information on a per-error basis
Admin_MaxAttempts		Maximum number of times the administration action should be attempted
Admin_Parameters		Fields object parent
	Resource	Name of resource to be managed
	Multiple fields	Detailed parameter data specific to the message class and action
Admin_Reason		Text message indicating reason for failure
Msg_ReplyToQ		Name of the queue to which the response should be sent
Msg_ReplyToQMgr		Name of the queue manager to which the response should be sent
Admin_RC		Numeric return code indicating the outcome
Msg_Style		Command or request/reply
Admin_TargetQMgr		Name of the queue manager owning the target resource

Three styles of administration message are supported, namely commands (datagrams) that indicate an administration action that does not require a reply, requests that require a reply, and the replies themselves. The reply is constructed from a copy of the original message; thus additional fields can be added by the sender for use by the receiver.

In addition to the basic administration message support, helper classes that encapsulate the message construction and interpretation of the reply are also provided for the most common administration operations . These classes can optionally supply user dialogues, which makes them useful for building simple administration tools.[9]

Selective administration

Access to administration can be controlled through the authenticator on the administration queue. For local applications the supplied authenticator considers them

9. These classes are not included in Version 1.1 but will be available from the MQSeries software download site on the World Wide Web (http://www.ibm.com/software/MQSeries/).

all to represent the same local user and therefore either allows or disallows administration for them all. Remote administration applications are controlled by the invocation of the authenticator on the channel before any administration messages flow. Different remote users can thus be distinguished and separately enabled or disabled. In all cases for any user, administration is enabled or disabled in its entirety. If a finer level of administration control is required, for example certain administration users are to be given access to some queues and not others, then additional programming is required. A more sophisticated authenticator can keep track of permissions associated with user identities, and administration messages can be subsequently be processed on the basis of these permissions (see security section). Rules associated with queues can also be exploited to allow or disallow actions in a similar manner (see “Rules” on page 38).

Monitoring and related actions

Administration is often concerned with more than object creation and modification, for example with monitoring the state of the system and with the handling of error situations; such as informing an operator when a queue is almost full, or by taking appropriate action when a message arrives that is too large for its target queue. These aspects are handled in MQSeries Everyplace through the use of rules, that is classes that are invoked whenever objects significantly change their status or when certain types of error situations arise. A default set of rules classes is provided with MQSeries Everyplace but typically these are replaced with custom classes (see “Rules” on page 38).

Dynamic channels

MQSeries Everyplace communicates between devices and/or gateway queue managers through logical links known as dynamic channels. These support bidirectional flows and are established by the queue manager as required. Asynchronous and synchronous messaging both use the same channels and the protocol used is unique to MQSeries Everyplace . By contrast MQSeries usually uses client channels for its synchronous traffic and a pair of message channels for bidirectional asynchronous messaging. MQSeries *cluster message channels* have some similar characteristics to the MQSeries Everyplace dynamic channels, but there are a number of differences.

A dynamic channel is a logical connection between two queue managers, established for the purpose of sending or receiving data. Multiple concurrent channels can exist, even between the same parties. They have characteristics, for example authentication, cryptography, compression, and the transport protocol used. These characteristics are pluggable, (different versions may be used on different channels) and consequently each channel has its own quality of service attributes of:

- **Authenticator:** either null or an *authenticator* object that can perform user or channel authentication
- **Channel:** the class providing the transport services.
- **Compressor:** either null or a *compressor* object that can perform data compression and decompression
- **Cryptor:** either null or a *cryptor* object that can perform encryption and decryption
- **Destination:** the target for this channel, for example SERVER.XYZ.COM

The authenticator is typically only used when setting up the channel. Compressors and cryptors are typically used on all flows.

The simplest type of cryptor is MQeXorCryptor, which encrypts the data being sent by performing an exclusive-OR of the data. This encryption is not secure, but it modifies the data so that it cannot be viewed. In contrast, MQe3DESCryptor implements triple DES. The simplest type of compressor is the MQeRleCompressor, which compresses the data by replacing repeated characters with a count. Other authenticators, compressors, and cryptors are supplied, see Table 14 on page 33.

Channel establishment uses protocol adapter specifications to determine the links and protocols to be used for a particular channel. At each intermediate node the channel definitions are searched to resolve the addressing needed for the next link. Where no onwards definition exists, the channel ends and any messages flowing through are passed to the queue manager at that point.

Channels are not directly visible to applications or administrators and are established by the queue manager as required. Channels link queue managers together and their characteristics are negotiated and renegotiated by MQSeries Everyplace dependent upon the information to be flowed. Transporters are the MQSeries Everyplace components that exploit channels to provide queue level communication. Again, these are not visible to the application programmer or administrator.

When assured messaging is demanded MQSeries Everyplace delivers messages to the application once, and once-only. It achieves this by ensuring that a message has successfully passed from one queue manager to another, and been 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 may be retransmitted (once-only delivery does not imply once-only transmission) but duplicates are not delivered.

Adapters

Adapters are used to map MQSeries Everyplace to device interfaces. Channels exploit the protocol adapters to run over HTTP, native TCP/IP, and other protocols. Similarly queues exploit fields storage adapters to interface to a storage subsystem such as memory or the file system.Adapters provide a mechanism for MQSeries Everyplace to both extend its device support and to allow versioning.

A *file descriptor* is a string that is used to identify, load and activate an adapter.

Dialup connection management

Dialup networking support for devices is handled by the device operating system. When MQSeries Everyplace on a disconnected device attempts to use the network, for example because a message must be sent, then if 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 therefore no explicit support needed for dialup networking in MQSeries Everyplace device implementations.

Trace

Trace is enabled by running an independent program that performs tracing actions. Embedded within MQSeries Everyplace are calls to trace for information, warning and error situations with system and user variants. Applications may also call trace directly and may add new messages or modify existing trace messages. The supplied sample trace program allows selected messages to be displayed, printed and/or directed to the event log. Other trace programs can be written with additional capabilities or be designed to format and deliver their output in other ways.

Most MQSeries Everyplace exceptions are passed to the application for handling, and the application exception handler may also route these to trace.

Event log

MQSeries Everyplace provides event log mechanisms and interfaces that may be used to log status, queue manager started for example. Logging can be initiated and by default written out to a file, however this can be intercepted and directed elsewhere. The MQSeries Everyplace event log does not log message data and cannot be used to recover messages or queues.

MQSeries Everyplace networks

MQSeries Everyplace networks are connected devices and gateways. They can span multiple physical networks and route messages between them. In general they provide synchronous and asynchronous access to queues with a programming model that is independent of queue location.

Configurations and scalability

A selection of basic MQSeries Everyplace network topologies is shown inFigure 5 on page 30. For these purposes it is assumed that each is configured with both synchronous and asynchronous communication capabilities.

The simplest case is where a standalone device supports synchronous inter-application communication through local queues, as in (a) above. More interesting however is case (b), illustrating a peer-to-peer network. This requires both devices to speak the same communications protocol, and at least one of the devices to be configured with listening capabilities, so that it can respond to the other attempting to make contact. Obviously in this simple case communication is possible only when both are available on the same network. Asynchronous messaging allows applications to run when the devices are not connected, synchronous messaging is possible only when the devices are actually connected.

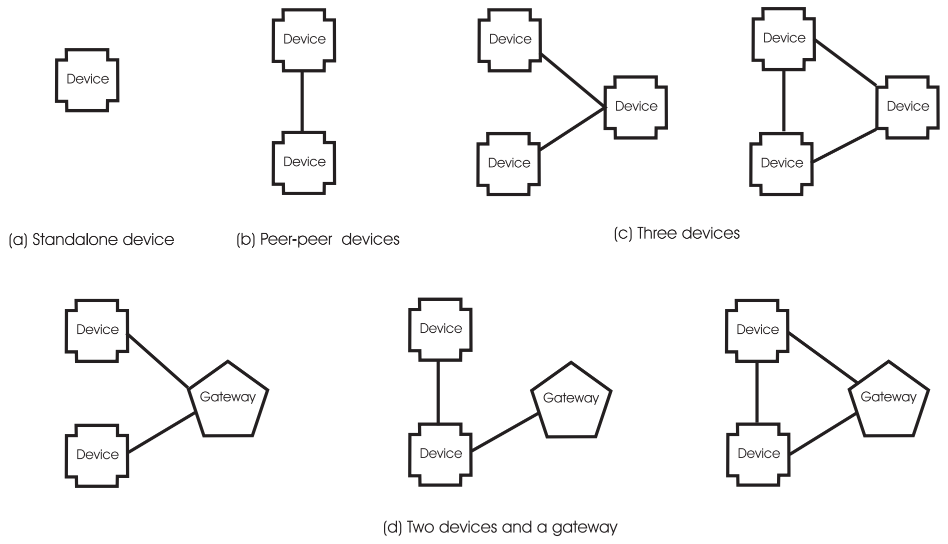

Figure 5. Simple MQSeries Everyplace networks

With three devices there are many more possibilities. Either one can play the role of a linking device, or all three can be interconnected. If they are configured to interchange messages to each other on a peer-to-peer basis then the considerations just given apply. If they communicate through a third device then there is much greater flexibility. The two communication devices do not need to speak the same communications protocol, each must speak one that is understood by the bridging device. Moreover asynchronous connectivity allows messages to flow if the sender and receiver are never on the network at the same time (provided each is on the network at a time when the linking device is also available). Synchronous communication requires all three devices to be available at the same time. For devices that are frequently disconnected, a configuration that goes through a third node of some kind is very appropriate, provided that the intermediary is generally available.

In practice devices are likely to be linked by a gateway three examples of which are shown in Figure 5(d). The preference for a gateway as a link node is based on the fact that gateways support multiple concurrent incoming connection requests. The first configuration shown is most likely, although the second and third are feasible, if somewhat unusual. In the third configuration note that only one route can be configured for a particular remote queue manager and so, although two routes appear to exist, one must be chosen.

For larger networks a number of gateways can be used, each gateway supporting a number of devices. The gateways can be interconnected in any way, but if full interconnection is defined then no routes between devices involve more than two gateways. Figure 6 on page 31 shows an example of a larger network.

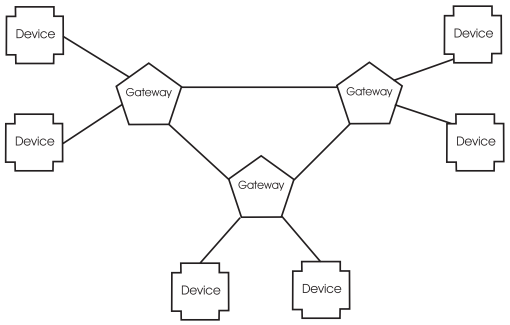

Figure 6. A star MQSeries Everyplace network

Asynchronous message delivery

When a message is asynchronously put to a remote queue, the message object is logically placed on the backing store associated with the local definition of that queue, along with its destination queue manager and queue names, and with the compressor, authenticator and cryptor characteristics that match the target destination of the message. The object's dump method is called as the object is saved to persistent storage in a secure format, as defined by its destination queue. The queue manager controls message delivery. It identifies (or establishes) a channel with appropriate characteristics to the queue manager for the next hop, then creates (or reuses) a transporter to the target queue. The transporter dumps the object and transmits the resulting byte string. Note that the target queue manager and queue name are not part of that message flow.

If appropriate, the message is encrypted and compressed over the channel. If it has reached its destination queue manager, it is decrypted and decompressed. A new message object is created, using the restore method of that object class, with the resultant object being placed on the destination queue. If the message has not reached its destination queue manager, it is decrypted and decompressed, then placed on a store and forward queue with the appropriate characteristics for onwards transmission. In both cases it is held on its respective queue in a secure format, as defined by its destination queue.

A characteristic of asynchronous message delivery is that messages are passed to the queue manager at intermediate hops, being queued for onwards transmission. Messages are taken off the intermediate queues firstly in priority order, then in timestamp sequence.

Synchronous message delivery

Synchronous message delivery is similar to the asynchronous case described above, but the queue manager involvement in intermediate hops takes place at a much lower level, involving the transporter and channels. A channel is established end-to-end, using the adapters defined in the protocol specifications at each intermediate node, to identify the next link. At the end of the last link, where no further relevant file descriptors exist, the message gets passed to the higher layers of the queue manager for processing. Thus the sending node does not queue the message but passes it along the channel, through intermediate hops, and then gives it to the destination queue manager to place it on the target queue.

The link into MQSeries uses a bridge queue at the gateway, which transforms the message to an MQSeries format. This mechanism means that synchronous MQSeries Everyplace -style messaging from a device is possible to MQSeries, with the dynamic channel terminating at the gateway. The message is delivered in real time from the gateway, through a client channel, to an MQSeries server. From there its destination may require it to be routed asynchronously along MQSeries message channels

In a similar manner a device capable of only synchronous messaging can send messages to an asynchronous MQSeries Everyplace queue, provided that a suitable intermediary is available.

Security

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

MQSeries Everyplace security 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

MQSeries Everyplace local and message-level security are used internally by MQSeries Everyplace , but are also made available to MQSeries Everyplace applications. MQSeries Everyplace queue-based security is an internal service.

The MQSeries Everyplace security features of all three categories protect message data by use of an attribute (MQeAttribute or descendent). Depending on the category, the attribute is applied either explicitly or implicitly.

Each attribute can contain the following objects:
- Authenticator
- Cryptor
- Compressor
- Key
- Target Entity Name

These objects are used differently, depending on the category of MQSeries Everyplace security feature, but in all cases, the MQSeries Everyplace security feature's protection is applied when the attribute attached to a message object is invoked. This occurs when an MQSeries Everyplace message's 'dump' method is invoked (when the attribute's ' encodeData' method is used, for example to encrypt and compress the message data). The MQSeries Everyplace security feature's unprotect occurs when the MQSeries Everyplace message's 'restore' method is invoked (when the attribute's 'decodeData' method is used, for example to decompress and decrypt the message data).

The algorithms supported by MQSeries Everyplace Version 1.1 for authentication, encryption and compression are detailed inTable 14.

Table 14. Authentication, encryption and compression support

Function	Algorithm
Authentication	Mini Certificate based, (derived from WAP forum WTLS Mini Certificate)
	Validation Windows NT/2000, AIX, or Solaris identity
Compression	LZW
	RLE
Encryption	Triple DES
	DES
	MARS
	RC4
	RC6
	XOR

MQSeries Everyplace local security

Local security protects MQSeries Everyplace message (or MQeFields or MQeFields descendent) data locally. This is achieved by creating an attribute with an appropriate symmetric cryptor and compressor, creating and setting up an appropriate 'key' (by providing a password or passphrase) and explicitly attaching the key to the attribute, and then attaching the attribute to the MQSeries Everyplace message. MQSeries Everyplace provides MQeLocalSecure class to assist with the setup of local security, but in all cases it is the responsibility of the local security user (MQSeries Everyplace internally or an MQSeries Everyplace application) to set up an appropriate attribute and manage the password or passphrase key.

MQSeries Everyplace queue-based security

Queue-based security can be applied to synchronous and asynchronous messages.

Synchronous queue-based security

Use of synchronous queue-based security allows an application to leave all message security considerations to MQSeries Everyplace . 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).

When a message is to be sent, the security characteristics of the target queue are retrieved from the local registry. If these are not present , the queue manager attempts to discover the target characteristics from the target queue manager and caches them for subsequent reuse. If a channel exists to that queue manager it is used; if not, a new channel is created. The target queue attributes are retrieved.

Based on the quality of service required, the channel attributes to the target queue manager are dynamically changed. This is subject to any rules that have been established. Typically a rule allows an upgrade in the level of security, (for example from no protection to weak protection, or from weak to strong). If the channel cannot be upgraded, or the security level is deemed excessive (for example no protection is required and the available channel implements strong protection) then a new channel is created. A pool of channels exists, reused where possible, with dynamically changing characteristics according to the demands of the traffic. Channels are automatically destroyed when not required. Messages are always placed on queues at the security level defined by the target queue characteristics.

Authentication takes place at the channel level, keeping the overhead per message to a minimum. Synchronous queue-based security is also typically used with symmetric cryptors since this results in fast encryption/decryption. However, in these symmetric cases, MQSeries Everyplace uses RSA asymmetric encryption initially, to protect the flows necessary to establish a shared key at the sender and receiver. After that point symmetric encryption is used to protect the confidentiality of the data flowed. MQSeries Everyplace makes the cryptographic attack of this data more difficult by changing the key dynamically on each channel flow. MQSeries Everyplace also ensures the integrity of the data flowed by generating and appending the digest to the data before sending, and regenerating and validating it on receipt.

Asynchronous queue-based security

Asynchronous messaging differs from the synchronous case described above in as far as there is no guarantee that the target queue is accessible at the time the putMessage is executed. In this case the queue manager cannot send the message immediately and places it on the transmission queue; however it is encrypted in accordance with its target queue characteristics. When it can be transmitted, it is decrypted, and then sent down a channel with suitable characteristics. Thus messages are always protected, even while awaiting transmission. Asynchronous messaging requires a remote queue definition - otherwise the target queue characteristics cannot be determined.

In the asynchronous case, authentication is not possible between originator and target. Where authentication is important, for example for a recipient to determine the message’s originator (to determine acceptance or establish non-repudiation) or for an initiator to ensure that message can only be processed by the intended recipient, message-level security must be used.

Queue-based security can be used at the same time as message-level security, but it is not necessary, since message data is already protected.

Message-level Security

Message-level security provides the protection of message data between an initiating and receiving MQSeries Everyplace application.

Message-level security is an application layer service that requires the initiating MQSeries Everyplace application to set up a message-level attribute and provide it when using putMessage to put the message to a target queue. The receiving application must set up and pass a matching message-level attribute to the receiving queue manager so that the attribute is available when the application invokes getMessage to get the message from the target queue.

Like local security, message-level security exploits the application of an attribute on a message object. The initiating application's queue manager handles the putMessage with the 'dump' method, which uses the attribute's 'encodeData' method to protect the message data. The receiving application's queue manager handles the application's getMessage with the 'restore' method which uses the attribute's 'decodeData' method to recover the original message data.

MQSeries Everyplace supplies two alternative attributes for Message-level security:

MQeMAttribute
This is used for business-to-business communications where mutual trust is tightly managed in the application layer and requires no trusted third party. All available MQSeries Everyplace symmetric cryptor and compressor choices can be used. Like local security, the attribute's key must be preset before it is provided with putMessage or getMesssage. MQeAttribute provides a simple and powerful method for message-level protection enabling the use of strong encryption to protect message confidentiality, without the overhead of any public key infrastructure (PKI).

MQeMTrustAttribute
This attribute provides a more advanced solution using digital signatures and exploiting the default public key infrastructure. It uses ISO9796 digital signature/validation to enable the receiving application to establish proof that the message comes from the purported sender. The supplied attribute's cryptor is used to protect message confidentiality. SHA1 digest guarantees message integrity and RSA encryption/decryption ensures that the message can only be restored by the intended recipient. As with MQeMAttribute, all available MQSeries Everyplace symmetric cryptor and compressor choices can be used. Chosen for size optimization, the certificates used are mini-certificates based on the WTLS Certificate proposed by the WAP forum WTLS Specification. The mutual availability of the information necessary to authenticate (validate signatures) and encrypt/decrypt is provided through the MQSeries Everyplace default infrastructure.

A typical MQeMTrustAttribute protected message has the format:

```
RSA-enc{SymKey}, SymKey-enc {Data, DataDigest, DataSignature}
```

where:

RSA-enc:	RSA encrypted with the intended recipient's public key

SymKey	generated pseudo-random symmetric key
SymKey-enc	symmetrically encrypted with the SymKey
Data	message data
DataDigest	digest of message data
DigSignature	initiator's digital signature of message data

Message-level security is independent of queue-level security.

The registry

The registry is the primary store for queue manager-related information; one exists for each queue manager. Every queue manager uses the registry to hold its:

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

Access to the registry is normally restricted to the legitimate queue manager user and is PIN protected, but a configurable option enables this to be bypassed by users more concerned with footprint size than security.

MQSeries Everyplace Authenticatable entities

Queue-based security, which uses mini-certificate based mutual authentication, and message-level protection, which uses digital signature, have triggered the concept of 'authenticatable entity'. In the case of mutual authentication it is normal to think about the authentication between two users (people), but in general, messaging has no concept of a user. Usually this concept is managed at the application level, that is, by the user of messaging services. MQSeries Everyplace deliberately abstracts the concept of 'target of authentication' from user to 'authenticatable entity'. This does not exclude the possibility of authenticatable entities being people, but this would be an application selected mapping. Internally, MQSeries Everyplace defines all queue managers that can either originate or be the target of mini-certificate dependent services as an authenticatable entity. In addition, MQSeries Everyplace also defines queues defined to use mini-certificate based authenticators to be an authenticatable entity. So queue managers that support these services may have one authenticatable entity, the queue manager, or a set of authenticatable entities, the queue manager and every queue that uses certificate based authenticator.

Private Registry and credentials

To be useful, every authenticatable entity needs its own credentials. This provides two challenges. Firstly how to execute registration to get the credentials, and secondly where to manage the credentials in a secure manner. Classically, these challenges are more difficult to solve than the underlying cryptographic techniques. MQSeries Everyplace provides default services that can be used to enable authenticatable entities to perform auto-registration, private registry (a descendent of base registry) to enable secure management of an authenticatable entity’s private credentials, and public registry (also a descendent of base registry) to manage set of public credentials. The private registry provides base registry with many of the qualities of a secure or cryptographic token, for example, it can be a secure repository for public objects like mini certificates, and private objects like private keys. It provides a mechanism to allow only the authorized user to access the private objects. It provides support for services (for example digital signature, 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 is currently is restricted to the local file system, but may well be extended to map to portable tokens in the future.

Auto-registration

MQSeries Everyplace 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 authenticatable entity’s private registry. Auto-registration steps include generating a new RSA key pair, protecting and saving the private key in the private registry; and 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 authenticatable entity’s new mini certificate, along with its own mini certificate and these, together with the protected private key, are stored in the authenticatable entity’s private registry as its new credentials. While auto-registration provides a simple mechanism to establish an authenticatable entity’s credentials, for message-level protection (MqeMTrustAttribute, see above), access to the intended recipient’s public key (mini certificate) is also required.

Public registry and certificate replication

MQSeries Everyplace provides default services that enable the sharing of authenticatable entity public credentials (mini certificates) between MQSeries Everyplace components. These are a prerequisite for MQeMTrust based message-level security. MQSeries Everyplace 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 facilitates the availability of the right mini certificate at the right time.

Application use of registry services

While the MQSeries Everyplace queue manager is designed to exploit the advantages of using private and public registry services, access to these services is not restricted. MQSeries Everyplace solutions may wish to define and manage their own authenticatable entities, for example users. Private-registry services can then be used to auto-register and manage the credentials of the new authenticatable entities, and public-registry services to make the public credentials available where needed. All registered authenticatable entities can be used as the initiator or recipient of message-level services protected using MQeMTrustAttribute

Default mini certificate issuance service

MQSeries Everyplace provides a default mini-certificate issuance service that can be configured to satisfy private-registry auto-registration requests. With the tools provided with MQSeries Everyplace , a solution can setup 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 (mini certificate based on WAP WTLS mini certificate)
- Mini Certificate Repository management

The tools provided with MQSeries Everyplace enable a mini-certificate issuance service administrator to authorize mini-certificate issuance to a given entity by registering its entity name and registered address and defining a one-time-use certificate request PIN. This is normally done after offline checking has validated the authenticity of the requestor. The certificate request PIN is posted to the intended user (for example in a similar way to the way that bank card PINs are posted to users when a new bank card is issued). The user of the private registry (for example the MQSeries Everyplace Application or MQSeries Everyplace queue manager) can then be configured to provide this certificate request PIN at startup time. When the private registry triggers auto-registration, the mini-certificate issuance service validates the resulting new certificate request' (based on a match of the presented entity name and certificate request PIN with their preregistered values), issues the new mini certificate and resets the registered certificate request PIN so that it cannot be reused. All auto-registration new mini-certificate requests are processed on a secure channel.

The set of mini certificates issued by a mini-certificate issuance service is held in the issuance service's own registry. When a mini certificate is reissued (for example as the result of expiry) then the expired mini certificate is archived.

The security interface

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

- Addition or removal of class aliases
- Definition of adapters
- Mapping of file descriptors
- Processing of channel commands

Configuration and customization

Rules

Rules are Java classes that are used to customize the behavior of MQSeries Everyplace 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

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

- Authenticator
- Compressor
- Cryptor

The rule would normally allow or disallow the change.

MQSeries bridge rules

These rules classes are given control when the MQSeries Everyplace to MQSeries-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 onto MQSeries Everyplace , when it is coming from MQSeries. For instance because the message is too big, or the queue does not exist.
- The state bridge administered objects should start in once the server is instantiated
- What to do when the bridge finds something wrong with the Sync queue on MQSeries (the persistent store used for crash recovery). The default rule just displays the problem.
- How to convert an MQSeries Everyplace message to an MQSeries message, and vice-versa. Transformers to do message conversion between MQSeries Everyplace and MQSeries messages are not derived from any MQeRule classes, instead they must implement the MQeTransformerInterface interface. Apart from this, transformers act like rules and are invoked when a message requires format conversion.

RAS dialer rules

This rule class is given control when the RAS dialer has a change of state, for example:

- What to do if the number to be called does not connect
- What to do once an error threshold is exceeded
- Dialling is attempted and only certain types of connections should be used, based on time of day. For example, only use the telephone if off-peak

Queue rules

This rule class is given control whenever a change of state of the associated queue occurs, for example:

- Adding a message to a queue. For example to see if a threshold is exceeded (number of messages, size of message, invalid priority)
- Queue characteristics assigned or changed
- Queue is opened or closed
- Queue is to be deleted

Queue manager rules

This rule class is given control whenever a change of state of the queue manager occurs, for example:

- Queue manager is opened. For example, start a background timer thread running to allow timed actions to occur
- Queue manager is closed. For example terminate the background timer thread
- A new queue is added

Connection styles

MQSeries Everyplace can support client-server[10] and/or peer-to-peer operation. A *client* is able to initiate communication with a server; a *server* is only able to respond to the requests initiated by a client. In *peer-to-peer* operation, the two peers can initiate flows in either direction. These connection styles require different components of MQSeries Everyplace to be available and active. The components involved are:

- **Channel listener:** that listens for incoming connection requests.
- **Channel manager:** that supports logical multiple concurrent communication pipes between end points.
- **Queue manager:** that supports applications through the provision of messaging and queuing capabilities.

Table 15 shows the relationship between these components and the connection style. The client/server connection style describes the situation where MQSeries Everyplace can operate in either client or server mode. The servlet option describes the case where MQSeries Everyplace is configured as an HTTP servlet with the HTTP server itself responsible for listening for incoming connection requests.

Table 15. Connection styles

	Queue manager	Channel manager	Channel listener
Client	Yes		
Client/server	Yes	Yes	Yes
Peer	Yes		
Server	Yes	Yes	Yes
Servlet	Yes	Yes	

Use of an MQSeries Everyplace channel manager or MQSeries Everyplace channel listener determines, for licensing purposes, that an MQSeries Everyplace instance is a gateway.

MQSeries Everyplace applications are not directly aware of the connection style used by the queue managers. However style is significant in that it affects what resources

10. In this section the terms ’client and ’server’ reflect general usage, not their MQSeries semantics.

are available to the parties, which queue managers can connect with other queue managers, the MQSeries Everyplace footprint at a device or gateway, and which connections can concurrently exist.

Peer-to-peer connection

A peer-to-peer channel includes the capabilities of a channel manager and a channel listener for a single channel. When a peer-to-peer channel is created between two queue managers, one queue manager must act as a listener and the other as the connection initiator. A peer-to-peer connected queue manager can initiate multiple peer-to-peer connections with other queue managers, but it can only respond to one incoming connection request and then must wait for that peer-to-peer channel to be closed before responding to another such request. Over any one peer-to-peer channel the two participating queue managers can both initiate actions, thus for example, applications on each queue manager can access queues on the other.

Peer-to-peer channels may not be usable through firewalls since the target of the incoming connection request may not be acceptable to the firewall.

Client-server connection

Standard channels, used for the client-server connection style, have no listening capabilities but depend on an independent listener at the server and the server requires a channel manager to handle multiple concurrent channels. The client initiates the connection request and the server responds. A server can usually handle multiple incoming requests from clients. Over a standard channel the client has access to resources on the server. If an application on the server needs synchronous access to resources on the client, a second channel is required where the roles are reversed. However, since standard channels are themselves bidirectional, messages destined for a client from its server’s transmission queue, are delivered to it over the standard (client-server) channel that it initiated.

A client can be a client to multiple servers simultaneously. Note that a channel manager is not required to support this configuration because channel managers handle multiple inbound channels.

The client-server connection style is generally suited for use through firewalls since the target of the incoming connection is normally identified as being acceptable to the firewall.

Multiple connection styles

A single queue manager can be capable of initiating either peer-to-peer or client/server connections, and of responding either as a server or a peer. In this case, the peer channel listener and the standard channel listener must have different port numbers.

Classes

MQSeries Everyplace provides a choice of classes for certain functions to allow the behavior of MQSeries Everyplace to be customized to meet specific application requirements. In some cases the interfaces to classes are documented so that additional alternatives can be developed. Table 16 on page 42 summarizes the possibilities. Classes can be identified either explicitly or through the use of alias names.

Table 16. Class options

Class	Alternates supplied	Interfaces documented
administration	no	yes
Authenticators	yes	no
Communications adapter	yes	yes
Communications style	yes	no
Compressors	yes	no
Cryptors	yes	no
Event log	sample provided	yes
Messages	no	yes
Queue storage	yes	no
Rules	default classes provided	yes
Trace	samples provided	yes

Application loading

When MQSeries Everyplace is configured to operate as a client (or peer) the initiating application is responsible for loading any other applications into the JVM. Standard Java facilities can be used for this, or the class loader included as part of MQSeries Everyplace is available. Thus, multiple applications can run against a single queue manager in the same JVM. Alternatively multiple JVMs can be used but each requires its own queue manager and each of these must have a unique name.

When MQSeries Everyplace is configured as a server MQSeries Everyplace is itself the initiating application. MQSeries Everyplace supports a preload class list and these classes are loaded in turn, before the queue manager is itself loaded.

Chapter 6. MQSeries Everyplace and MQSeries networks

Although an MQSeries Everyplace network can exist standalone, without the need for an MQSeries server or network, in practice MQSeries Everyplace is often used to complement an existing MQSeries installation, extending its reach to new platforms and devices, or providing advanced capabilities such as queue or message based security or synchronous messaging. From an MQSeries Everyplace application perspective, MQSeries queues and queue managers can be regarded as simply additional remote queues and queue managers. However, a number of functional restrictions exist because these queues are not accessed directly through MQSeries Everyplace dynamic channels and an MQSeries Everyplace queue manager, but require the involvement of an MQSeries Everyplace gateway. The gateway can send messages to multiple MQSeries queue managers either directly or indirectly, through MQSeries client channels. If the connection is indirect, the messages pass through MQSeries client channels to an intermediate MQSeries queue managers and then onwards through MQSeries message channels to the target queue manager.

Messages from an MQSeries application destined for MQSeries Everyplace are addressed to the MQSeries Everyplace queue manager and queue as normal, with the MQSeries routing (remote queue manager definitions) defined such that the MQSeries Everyplace messages arrive on specific MQSeries transmission queues. MQSeries channels are not defined for the transmissions queues, as would be normal practice, instead the MQSeries Everyplace gateway pulls the messages off these queues and ensures their delivery to the MQSeries Everyplace destination. The number of transmission queues to be used (that is the number of routes) is configurable and should be set to reflect the volume of messages to be delivered.

Interface to MQSeries

The architecture of MQSeries Everyplace supports the concept of one or more optional bridges between MQSeries Everyplace and other messaging systems.

In MQSeries Everyplace Version 1.1 only one such bridge is supported, the *MQSeries bridge* that interfaces between MQSeries Everyplace and MQSeries networks. This bridge uses the MQSeries Java client to interface to one or more MQSeries queue managers, thereby allowing messages to flow from MQSeries Everyplace to MQSeries and vice versa. In MQSeries Everyplace Version 1 one such bridge is recommended per gateway, and each is associated with multiple *MQSeries queue manager proxies* (definitions of MQSeries queue managers). A queue manager proxy definition is required for each MQSeries queue manager that communicates with MQSeries Everyplace . Each of these definitions can have one or more associated *client connection services*, where each represents a connection to a single MQSeries queue manager. Each of these may use a different MQSeries server connection to the queue manager, and optionally a different set of properties such as user exits or ports.

A gateway client connection service may have a number of *listeners* that use that gateway service to connect to the MQSeries queue manager and retrieve messages from MQSeries to MQSeries Everyplace . A listener uses only one service to establish its connection, with each listener connecting to a single transmission queue on the MQSeries queue manager. Each listener moves messages from a single MQSeries transmission queue to anywhere on the MQSeries Everyplace network, via its parent gateway queue manager. Thus a single gateway queue manager can funnel multiple MQSeries message sources into the MQSeries Everyplace network.

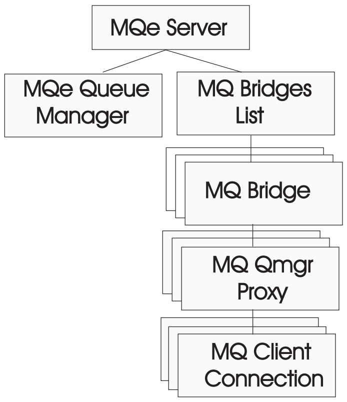

Figure 7. MQSeries Bridge object hierarchy

When moving messages in the other direction, from MQSeries Everyplace to MQSeries, the gateway queue manager configures one or more *bridge queue* objects. Each bridge queue object can connect to any queue manager directly and send its messages to the target queue. In this way a gateway can dispatch MQSeries Everyplace messages routed through a single MQSeries Everyplace queue manager to any MQSeries queue manager, either directly or indirectly. The bridges object has the properties shown in Table 17.

Table 17. Bridges object properties

Property	**Explanation**
Bridgename	List of bridge names
Run state	Status: running or stopped

The bridges object, and the other gateway objects can be started and stopped independently of the MQSeries Everyplace queue manager. If such a gateway object is started (or stopped) the action also applies to all of its children (all bridges, queue manager proxies, client connections, and transmission queue listeners). The bridge object has the properties shown in Table 18 on page 45.

Table 18. Bridge properties

Property	Explanation
Class	Bridge class
Default transformer	The default class (rule class) to be used to transform a message from MQSeries Everyplace to MQSeries (or vice versa) if no other transformer class has been associated with the destination queue
Heartbeat interval	The basic timing unit to be used for performing actions against bridge objects
Name	Name of the bridge object
Run state	Status: running or stopped
Startup rule class	Rule class used when the bridge object is started
MQSeries Queue Manager Proxy Children	List of all Queue Manager Proxies that are owned by this bridge

In simple cases a default transformer (rule) can be used to handle all message conversions. Additionally a transformer can be set on a per listener basis (for messages from MQSeries to MQSeries Everyplace) that overrides this default. For more specific control the transformation rules can be set on a target queue basis using bridge queue definitions on the gateway; this applies both to MQSeries Everyplace and MQSeries target queues.

The MQSeries queue manager proxy holds the properties specific to a single MQSeries queue manager. The proxy properties are shown in Table 19.

Table 19. MQSeries queue manager proxy properties

Property	Explanation
Class	MQSeries queue manager proxy class
MQSeries host name	IP host name used to create connections to the MQSeries queue manager via the Java client classes. If not specified then the MQSeries queue manager is assumed to be on the same machine as the bridge and the Java bindings are used
MQSeries queue manager proxy name	The name of the MQSeries queue manager
Name of owning bridge	Name of the bridge object that owns this MQSeries queue manager proxy
Run state	Status: running or stopped
Startup rule class	Rule class used when the MQSeries queue manager object is started
Client Connection Children	List of all the Client Connection objects that are owned by this proxy

The bridge client connection service definition holds the detailed information required to make a connection to an MQSeries queue manager. The connection properties are shown in Table 20 on page 46.

Table 20. Client connection service properties

Property	**Explanation**
Adapter class	Class to be used as the gateway adapter
CCSID*	The integer MQSeries CCSID value to be used
Class	Bridge client connection service class
Max connection idle time	The maximum time a connection is allowed to be idle before being terminated
MQSeries password*	Password for use by the Java client
MQSeries port*	IP port number used to create connections to the MQSeries queue manager via the Java client classes. If not specified then the MQSeries queue manager is assumed to be on the same machine as the bridge and the Java bindings are used
MQSeries receive exit class*	Used to match the receive exit used at the other end of the client channel; the exit has an associated string to allow data to be passed to the exit code
MQSeries security exit class*	Used to match the security exit used at the other end of the client channel; the exit has an associated string to allow data to be passed to the exit code
MQSeries send exit class*	Used to match the send exit used at the other end of the client channel; the exit has an associated string to allow data to be passed to the exit code
MQSeries user id*	User id for use by the Java client
Client connection service name	Name of the server connection channel on the MQSeries machine
Name of owning queue manager proxy	The name of the owning queue manager proxy
Startup rule class	Rule class used when the bridge client connection service object is started
Sync queue name	The name of the MQSeries queue that is used by the bridge for synchronization purposes
Sync queue purger rules class	The rules class to be used when a message is found on the sync queue
Run state	Status: running or stopped
Name of owning Bridge	The name of the Bridge object that owns this client connection
MQ XmitQ Listener Children	List of all the listener objects that use this client connection
*Details of these parameters can be found in the *MQSeries Using Java* documentation	

The *adapter class* is used to send messages from MQSeries Everyplace to MQSeries and the *sync queue* is used to keep track of the status of this process. Its contents are used in recovery situations to guarantee assured messaging; after a normal shutdown the queue is empty. It can be shared across multiple client connections and across multiple bridge definitions provided that the receive, send and security exits are the same. This queue can also be used to store state about messages moving from MQSeries to MQSeries Everyplace , depending upon the listener properties in use. The *sync queue purger rules class* is used when a message is found on the sync queue, indicating a failure of MQSeries Everyplace to confirm a message.

The maximum connection idle time is used to control the pool of Java client connections maintained by the bridge client connection service to its MQSeries system. When an MQSeries connection becomes idle, through lack of use, a timer is started and the idle connection is discarded if the timer expires before the connection is reused. Creation of MQSeries connections is an expensive operation and this process ensures that they are efficiently reused without consuming excessive resources. A value of zero indicates that a connection pool should not be used.

The listener object, which moves messages from MQSeries to MQSeries Everyplace , has the properties shown in Table 21.

Table 21. Listener properties

Property	**Explanation**
Class	Listener class
Dead letter queue name	Queue used to hold messages from MQSeries to MQSeries Everyplace that cannot be delivered
Listener state store adapter	Class name of the adapter used to store state information
Listener name	Name of the MQSeries XMIT queue supplying messages
Owning client connection service name	Client connection service name
Run state	Status: running or stopped
Startup rule class	Rule class used when the listener object is started
Transformer class	Rule class used to determine the conversion of an MQSeries message to MQSeries Everyplace
Undelivered message rule class	Rule class used to determine action when messages from MQSeries to MQSeries Everyplace cannot be delivered
Seconds wait for message	An advanced option that can be used to control listener performance in exceptional circumstances

The *undelivered message rule class* determines what action is taken when a message from MQSeries to MQSeries Everyplace cannot be delivered. Typically it is placed in the *dead letter queue* of the MQSeries system.

In order to provide assured delivery of messages, the listener class uses the *listener state store adapter* to store state information, either on the MQSeries Everyplace system or in the sync queue of the MQSeries system.

In order to complete the configuration of the bridge, both remote queue manager and remote queue definitions are required. Remote queue manager definitions for remote MQSeries Everyplace queue managers follow standard MQSeries Everyplace practice; definitions for remote MQSeries queue managers have the channel definition set to null to indicate that a normal MQSeries Everyplace dynamic channel is not used - instead a connection is defined to the MQSeries queue manager as detailed above.

The remote queue definition for an MQSeries Everyplace queue again follows standard practice; however for an MQSeries queue it is significantly changed from that used for MQSeries Everyplace queues. Table 22 shows the properties for MQSeries remote queues.

Table 22. MQSeries remote queue properties

Property	Explanation
Alias names	Alternative names for the queue
Authenticator	Must be null
Class	Object class
Client connection	Name of the client connection service to be used
Compressor	Must be null
Cryptor	Must be null
Expiry	Passed to transformer
Maximum message size	Passed to the rules class
Mode	Must be synchronous
MQ queue manager proxy	Name of the MQSeries queue manager to which the message should first be sent
MQSeries bridge	Name of the bridge to convey the message to MQSeries
Name	Name by which the remote MQSeries queue is known to MQSeries Everyplace
Owning queue manager	Queue manager owning the definition
Priority	Priority to be used for messages (unless overridden by a message value)
Remote MQSeries queue name	Name of the remote MQSeries queue
Rule	Rule class used for queue operations
Queue manager target	MQSeries queue manager owning the queue
Transformer	Name of the transformer class that converts the message from MQSeries Everyplace format to MQSeries format
Type	MQSeries bridge queue

The *cryptor*, *authenticator*, and *compressor* classes define a set of queue attributes that dictate the level of security for any message passed to this queue. From the time on MQSeries Everyplace that the message is sent initially, to the time when the message is passed to the MQSeries bridge queue, the message is protected with at least the queue level of security. These security levels are *not* applicable when the MQSeries bridge queue passes the message to the MQSeries system, the security send and receive exits on the client connection are used during this transfer. No checks are made to make sure that the queue level of security is maintained.

MQSeries bridge queues are synchronous only; asynchronous applications must therefore send messages to these queues via MQSeries Everyplace Store and Forward/Home Server Queues, or via asynchronous remote queue definitions.

Administration of the gateway is handled in the same way as the administration of a normal MQSeries Everyplace queue manager - through the use of administration messages. New classes of messages are defined as appropriate to the managed object. Table 12 on page 25 shows the gateway administration message classes.

Message conversion

MQSeries Everyplace messages destined for MQSeries pass through the bridge and are converted into an MQSeries format, using either a default transformer or one specific to the target queue. A custom transformer offers much flexibility, for example it would be good practice to use a subclass of the MQSeries Everyplace message object class to represent messages of a particular type over the MQSeries Everyplace network. At the gateway a transformer could convert the message into an MQSeries format using whatever mapping between fields and MQSeries values that was appropriate as well as add specific data to represent the significance of the subclass.

The default transformer from MQSeries Everyplace to MQSeries 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 MQSeries Everyplace to MQSeries:**

  The default transformer from MQSeries Everyplace to MQSeries works in conjunction with the MQeMQMsgObject class. This class is a representation of all the fields you could find in an MQSeries message header. Using the MQeMQMsgObject, your application can set values (priority for example) using set() methods. Thus, when an MQeMQMsgObject (or an object derived from the MQeMQMsgObject class) is passed through the default MQSeries Everyplace transformer, the default transformer (MQeBaseTransformer) gets the values from inside the MQeMSMsgObject, and sets the corresponding values in the MQSeries message (for example, the priority value is copied over to the MQSeries message).

  If the message being passed is not an MQeMQMsgObject, and is not derived from the MQeMQMsgObject class, the whole MQSeries Everyplace message is copied into the body of the MQSeries message (*funneled*). The message format field in the MQSeries message header is set to indicate that the MQSeries message holds a message in MQSeries Everyplace ″funneled″ format.

- **MQSeries to MQSeries Everyplace message flow:**

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

  If the MQSeries header indicates a ″funneled″ MQSeries Everyplace message, then the MQSeries message body is reconstituted as the original MQSeries Everyplace message that is then posted to the MQSeries Everyplace network.

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

This MQeMQMsgObject class and the default transformer behavior mean that :

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

Function

MQSeries remote queues are enabled for synchronous MQSeries Everyplace put messaging operations, from an MQSeries Everyplace queue manager; all other messaging operations must be asynchronous.

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

Compatibility

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

- **Addressing and naming:**
  - identical addressing semantics using a queue manager/queue address
  - Common use of an ASCII name space.
- **Applications:**MQSeries Everyplace is able to support existing MQSeries applications without application change.
- **Channels:**MQSeries Everyplace gateways use MQSeries client channels.
- **Message interchange and content:**
  - interchange of messages between MQSeries Everyplace and MQSeries
  - message network invisibility (messages from either MQSeries Everyplace or MQSeries can cross the other network without change).
  - mutual support for identified fields in the MQSeries message header
  - once-only assured message delivery
- **Sample applications:**Interoperability of the MQSeries Postcard and the MQSeries Everyplace Postcard [11]applications

MQSeries Everyplace Version 1.1 does not support all the functions of MQSeries. Apart from environmental, operating system and communication considerations, some of the more significant differences are detailed below. Note however that within MQSeries

11. This application is not included in Version 1.1 but is available from the MQSeries software download site on the World Wide Web (http://www-4.ibm.com/software/ts/MQSeries/).

Everyplace many application tasks can be achieved through alternative means using MQSeries Everyplace features, or through the exploitation of sub-classing, the replacement of the supplied classes or the exploitation of the rules, interfaces and other customization features built into the product.

- No clustering support
- No distribution list support.
- No grouped/segmented messages.
- No load balancing/warm standby capabilities.
- No reference message.
- No report options.
- No shared queue support
- No triggering.
- No unit of work support, no XA-coordination

Scalability and performance characteristics are different.

Assured delivery

Although both MQSeries Everyplace and MQSeries offer assured delivery, they each provide for different levels of assurance. When a message is travelling from MQSeries Everyplace to MQSeries, the message transfer is only assured if the combination of putMessage and confirmPutMessage is used (see “Queue manager operations” on page 23). When a message is travelling from MQSeries to MQSeries Everyplace , the transfer is assured only if the MQSeries message is defined as persistent.

Chapter 7. Applications and utilities

Note: These applications and utilities are not included in Version 1.1 but will be available from the MQSeries software download site on the World Wide Web (`http://www-4.ibm.com/software/ts/MQSeries/`).

Postcard

Postcard is an MQSeries Everyplace application that may be used to validate the operation of a standalone MQSeries Everyplace network or the inter-operation of an MQSeries Everyplace and MQSeries networks. Postcard is a Java application that allows text messages to be sent to a user at a remote queue manager. It inter-operates with Postcard such that a Postcard message sent to an MQSeries target results in a postcard being received, and vice versa.

There is also a C version of the MQSeries Everyplace Postcard application that runs on PalmOS and can interoperate with the Java version.

MQSeries Everyplace Explorer

MQSeries Everyplace Explorer is a management tool, written in Java, that allows the configuration and exploration of local and remote queue managers, queues and messages. It uses the Microsoft® foundation classes to present a standard Windows graphical user interface - but as a consequence it cannot execute on non-Windows platforms . However it can be used to manage all MQSeries Everyplace queue managers since it operates exclusively through the sending and receiving of administration messages. It presents a two-pane view of an MQSeries Everyplace network; a tree view of objects in the left hand pane and a list view of object details in the right hand pane.

MQSeries Everyplace Explorer has the following capabilities:

- Display or modify queue manager properties
- Create, delete or modify connections and display their properties
- Create, delete or modify queues and display their properties and/or contents
- Browse or delete messages, display their properties and inspect their fields
- Send test messages
- Configure the MQSeries Everyplace bridge to MQSeries

MQSeries Everyplace Explorer typically uses an already configured queue manager and can load other classes for execution. If no such queue manager exists, it creates one with user-selected characteristics.

Multiple copies of the tool can be run on a single machine, with each running in its own JVM. This arrangement allows the simulation of an MQSeries Everyplace network and can be used to investigate and demonstrate MQSeries Everyplace networking and operations.

Chapter 8. Programming interfaces

The *MQSeries Everyplace Systems Programming Interface (SPI)* is the programming interface to MQSeries Everyplace . Two implementations are available depending upon the operating system. The Java version provides access to all MQSeries Everyplace functions; the C interface in MQSeries Everyplace Version 1.1 only provides access to a subset. The detailed classes, methods and procedures are detailed in the *MQSeries Everyplace for Multiplatforms Programming Reference*; examples of MQSeries Everyplace programming are given in the *MQSeries Everyplace for Multiplatforms Programming Guide*.

Appendix. Notices

Trademarks

Glossary

This glossary describes terms used in this book and words used with other than their everyday meaning. In some cases, a definition may 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, see the index or the *IBM Dictionary of Computing*, New York:. McGraw-Hill, 1994.

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 communicating 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 checks that verifies the senders and receivers of messages.

bridge. An MQSeries Everyplace object that allows messages to flow between MQSeries Everyplace and other messaging systems, including MQSeries.

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

channel manager. An MQSeries Everyplace object that supports logical multiple concurrent communication pipes between end points.

class. A class is 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 MQSeries, a client is a run-time component that provides access to queuing services on a server for local user applications.

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

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

dynamic channel. A dynamic channel connects MQSeries Everyplace devices and transfers synchronous and asynchronous messages and responses in a bidirectional manner.

encapsulation. Encapsulation is 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.

gateway. An MQSeries Everyplace gateway (or server) is a computer running the MQSeries Everyplace code including a channel manager.

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

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

interface. An interface is 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. The Internet is 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).

Java Developers 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.

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

message. In message queuing applications, a message is 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. Method is the object-oriented programming term for a function or procedure.

MQI channel. An MQI channel connects an MQSeries client to a queue manager on a server system and transfers MQI calls and responses in a bidirectional manner.

MQSeries. MQSeries is a family of IBM licensed programs that provide message queuing services.

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 MQSeries, an object is a queue manager, a queue, or a channel.

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 addistant (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

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

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

server. (1) An MQSeries Everyplace server is a device that has an MQSeries Everyplace channel manager configured. (2) An MQSeries 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. (3) The computer on which a server program runs.

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*.

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.

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.

Bibliography

Related publications:

- *MQSeries Everyplace for Multiplatforms Read Me First*, GC34–5862–00
- *MQSeries Everyplace for Multiplatforms Programming Reference*, SC34–5846–00
- *MQSeries Everyplace for Multiplatforms Programming Guide*, SC34–5845–00
- *MQSeries An Introduction to Messaging and Queuing*, GC33-0805-01
- *MQSeries for Windows NT V5R1 Quick Beginnings*, GC34-5389-00

Index

Sending your comments to IBM

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM.

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book.

Please limit your comments to the information in this book and the way in which the information is presented.

To make comments about the functions of IBM products or systems, talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate, without incurring any obligation to you.

You can send your comments to IBM in any of the following ways:

- By mail, to this address:

  User Technologies Department (MP095)
  IBM United Kingdom Laboratories
  Hursley Park
  WINCHESTER,
  Hampshire
  SO21 2JN
  United Kingdom
- By fax:
  - From outside the U.K., after your international access code use 44–1962–870229
  - From within the U.K., use 01962–870229
- Electronically, use the appropriate network ID:
  - IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
  - IBMLink™: HURSLEY(IDRCF)
  - Internet: idrcf@hursley.ibm.com

Whichever method you use, ensure that you include:

- The publication title and order number
- The topic to which your comment applies
- Your name and address/telephone number/fax number/network ID.

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

GC34-5843-01