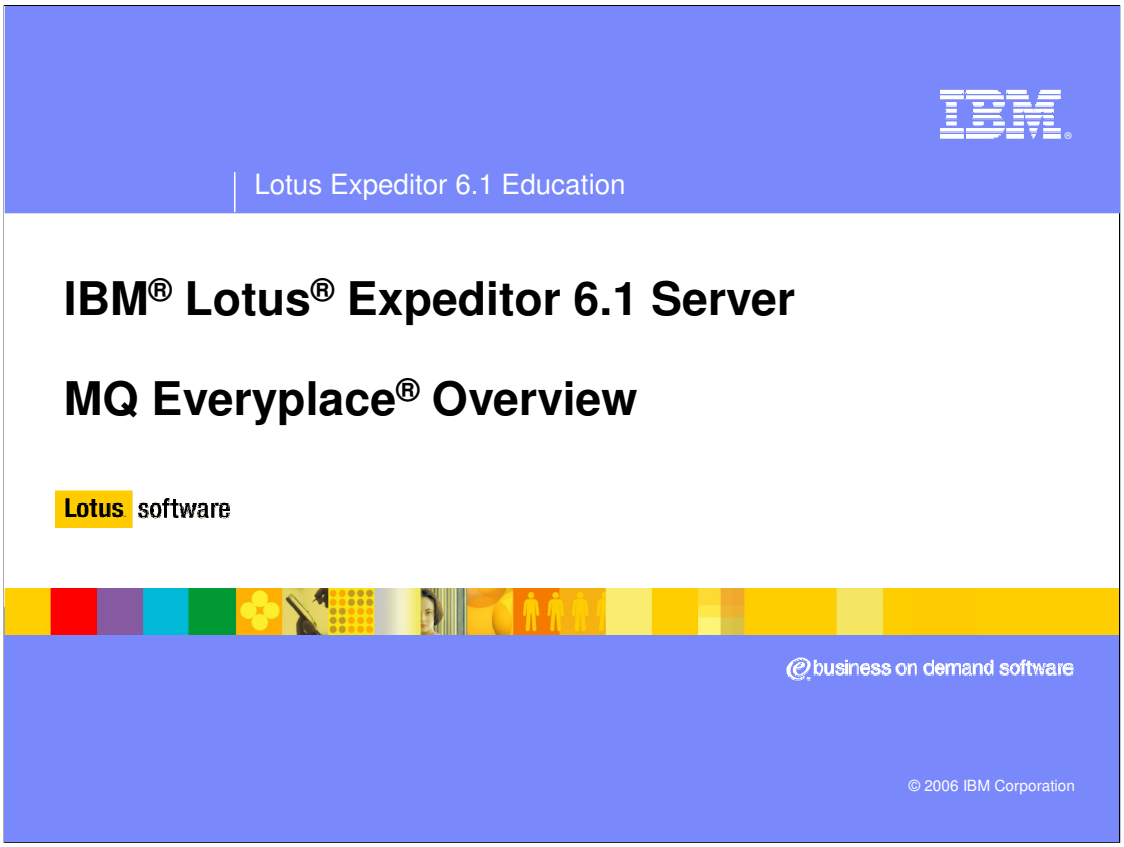
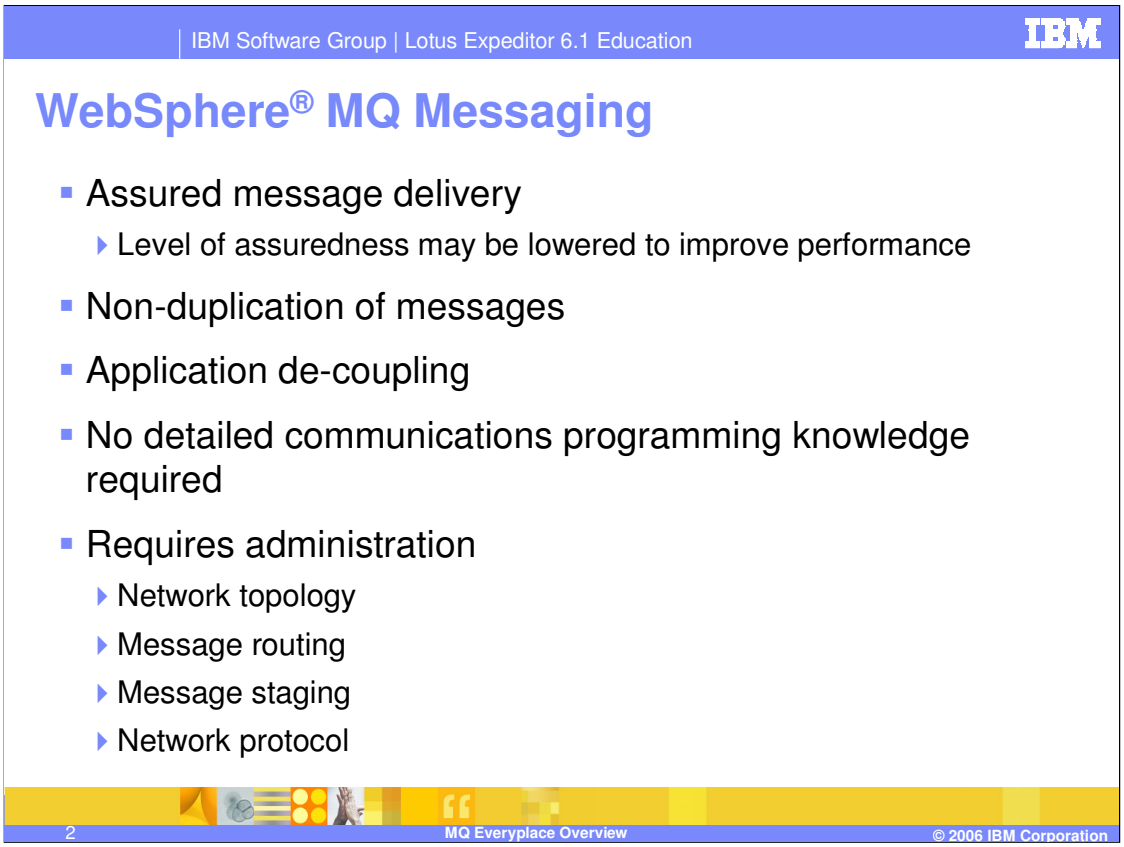
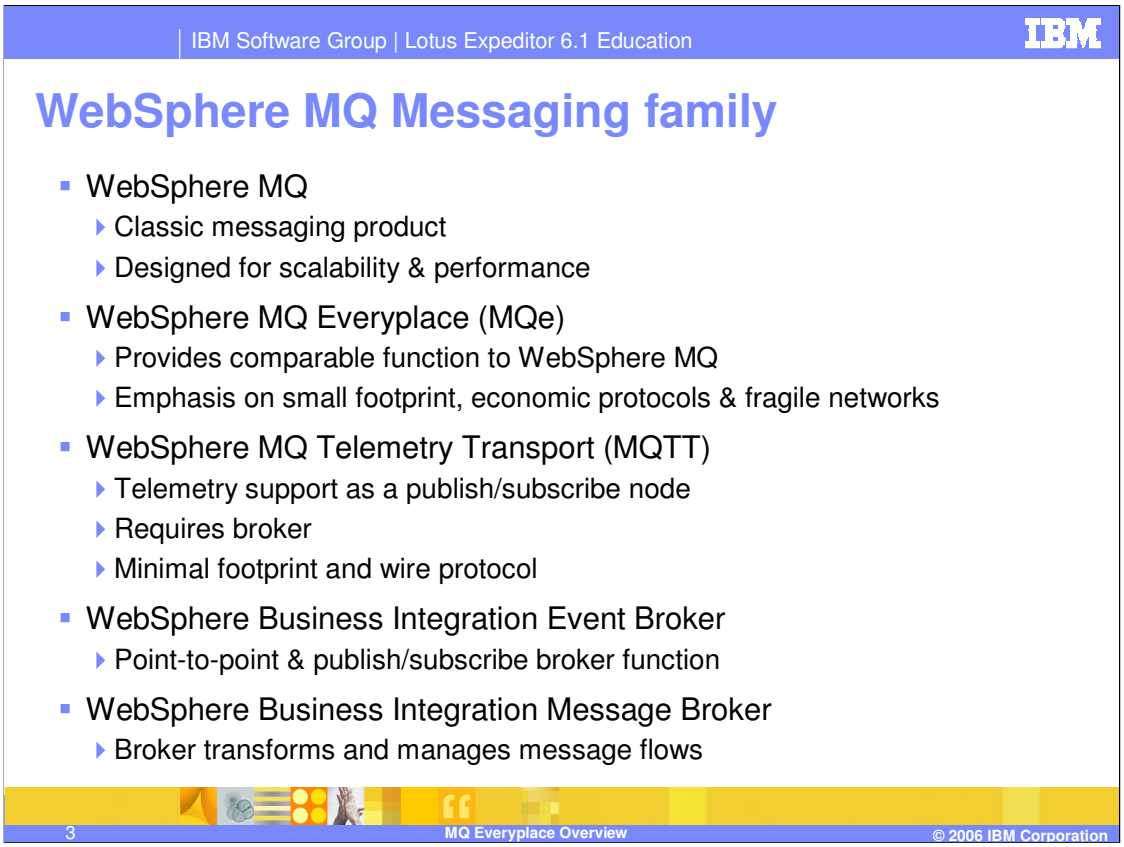


Hello, and welcome to this overview of MQ Everyplace for Lotus Expeditor 6.1 Server.



A message is a collection of data sent by one application and intended for another application. WebSphere MQ Messaging provides assured, once-and-only-once delivery of messages. Assured means that once the WebSphere MQ Messaging system receives a message to process, it guarantees delivery. The level of assuredness can be lowered to improve performance. Messages can be delivered on a “at most once” basis. In this mode a message may be lost; however, in all cases, duplicate messages will not be received. By providing asynchronous message delivery, the sending application is de-coupled from the receiving application. In addition, message delivery is handled by the WebSphere MQ subsystem, shielding the application programmer from having to understand anything about the underlying communications systems.

The WebSphere MQ Messaging system must be configured to define the topology of the network and how messages will be routed from the sending application to the receiving application. The administrator defines whether or not messages will pass through staging servers before arriving at the target destination. The administrator configures the protocol that will be used to communicate between server queues.



The WebSphere messaging family consists of several products:

•WebSphere MQ, the core of application integration, integrates many platforms. It provides the messaging foundation for an enterprise service bus, and assures reliable message delivery. It can be used alone or combined seamlessly with WebSphere Application Server. Both WebSphere MQ and WebSphere Application Server provide messaging resources that can form the foundation for a company's ESB, which can grow incrementally with their business needs. WebSphere MQ is designed for scalability and performance.

•WebSphere MQ Everyplace, which brings the benefits of assured message delivery and rock-solid security to the failure-prone environment of mobile working. The emphasis is on small footprint, economic protocols and fragile networks. MQ Everyplace has a client, server, and gateway to MQ functionality.

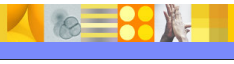
•The WebSphere MQ Telemetry transport is a lightweight Publish/Subscribe protocol that can be used for integrating devices with WebSphere MQ brokers. This support, along with MQ Everyplace, is provided with the Lotus Expeditor client.

•WebSphere Business Integration Event Broker, provides point-to-point messaging capabilities and one to many publish and subscribe broker support. The event broker coordinates and filters message flow to optimize efficiency.

•WebSphere Business Integration Message Broker, builds upon WebSphere Business Integration Event Broker, is a powerful information broker that includes a one-to-many connectivity model plus transformation, intelligent routing, and information flow modeling across multiple, disparate business systems. It also supports publications and subscriptions including mobile clients and remote telemetry devices.

Section

WebSphere MQ Everyplace



Now let's talk about WebSphere MQ Everyplace.

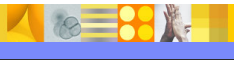
WebSphere MQ Everyplace provides industry strength messaging optimized for the mobile environment with intermittent network connectivity. It operates efficiently in hostile communications environments where networks are unstable or where bandwidth is tightly constrained. MQe has an efficient wire protocol and automated recovery from communication link failures.

MQ Everyplace provides synchronous and asynchronous, once-only assured delivery of messages for the devices supported by Lotus Expeditor. MQ Everyplace provides extensive security features to protect messages, queues, and related data, whether in storage or in transmission. MQ Everyplace provides for authentication, data encryption, non-repudiation and compression. MQ Everyplace automatically manages the characteristics of communication channels used between queue managers. Both data and serialized Java objects can be sent in a message.

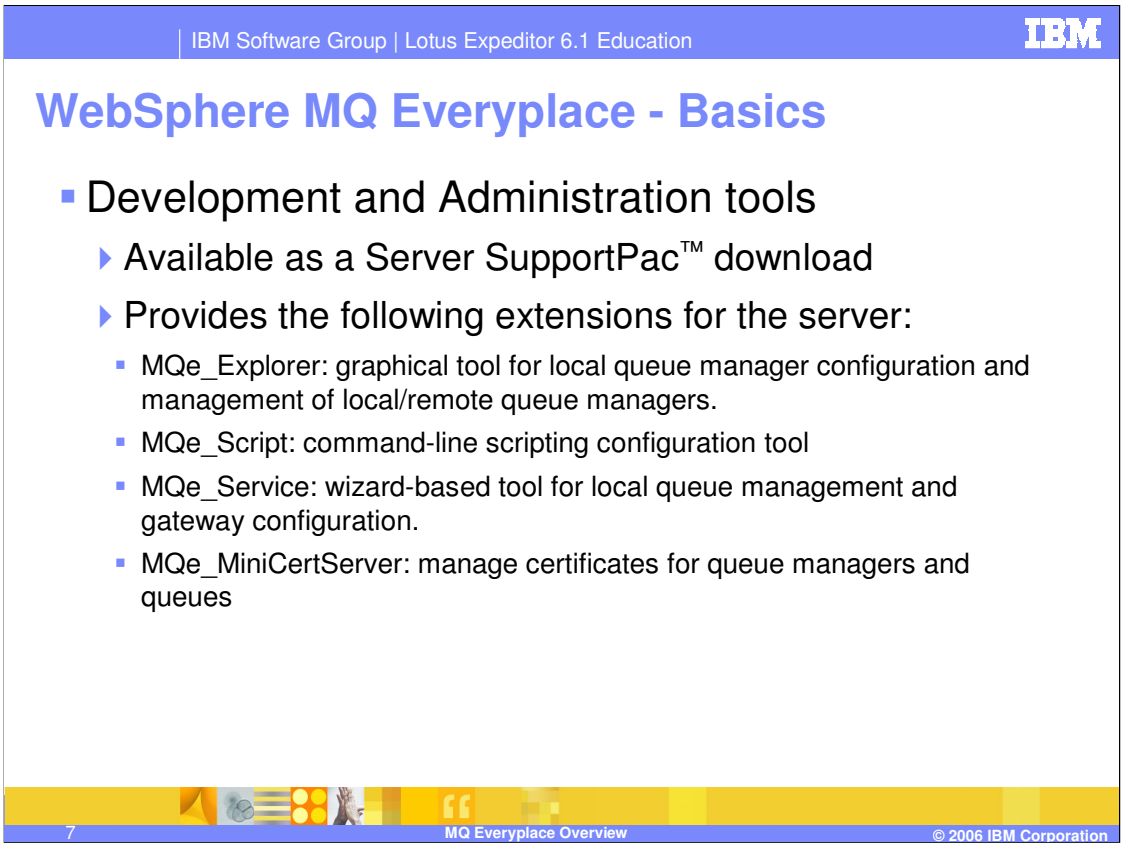
MQ Everyplace serves as a gateway to IBM's enterprise messaging offerings which include WebSphere MQ, WebSphere Business Integration Message Broker and WebSphere Business Integration Event Broker.

WebSphere MQ Everyplace - Basics

- Development Kit
  - Java™ interface (includes JMS support)
    - Client
    - Server
    - Gateway
  - C interface
    - Client only



The WebSphere MQe Development Kit is a development environment for writing messaging and queuing applications based on Java and C. The Java libraries provide access to all MQe client, server and gateway functionality. This includes support for Java Messaging Service APIs for point-to-point messaging. The native C API libraries provide a subset of the MQe functionality and are limited to client functions. A C Binding library provides C API access to the majority of the Java API functions that are not supported by the native C interface.



Tools are available to assist with developing and administering MQe applications.

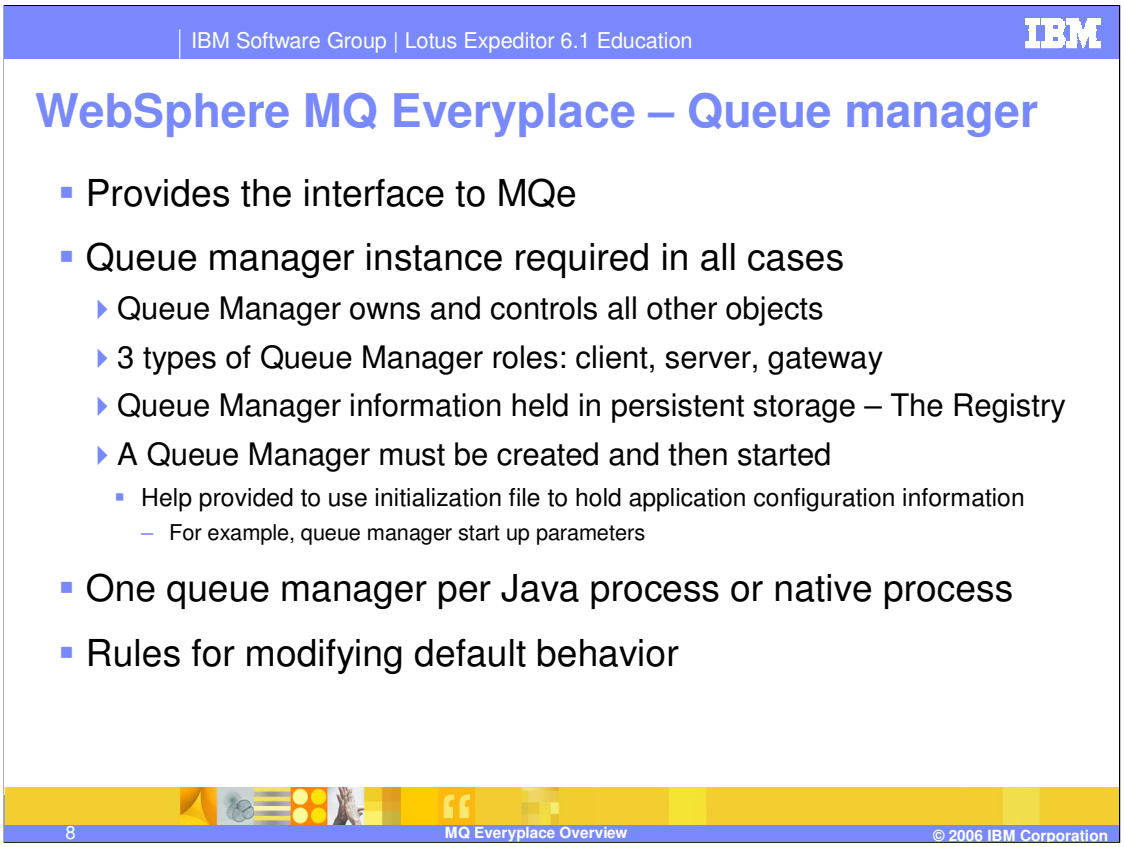
The MQe_Explorer provides a graphical user interface for the management of an MQe network and its interconnection with MQ. It allows MQe queue managers and their associated objects, such as queues, connections, and bridges, to be locally or remotely configured. MQe_Explorer also provides a simple way of creating local queue managers, which can then be further configured to meet the needs of applications. It also offers a launch and debug environment for MQe applications.

MQe_Script is a command-line based tool for MQe and is platform independent. It allows MQe queue managers and their associated objects, such as queues, connections, listeners, and bridge objects, to be locally or remotely configured. Test messages can be sent to the queues to validate the operation of the network. Like the MQe_Explorer, MQe_Script provides a simple way of creating local queue managers, which you can then configure and extend for use by your application.

MQe_Service is a wizard-based tool for MQe local queue manager creation and operation. It enables the automated set up of MQe gateway and MQ queue managers, where messages are required to pass between MQe and MQ networks.

MQe_MiniCertServer is a tool for the issue and renewal of WLTS certificates to queue managers and queues, which are used for certificate-based authentication.

These tools are not installed with Lotus Expeditor. You can download these tools from the WebSphere MQ Everyplace support site: http://www.ibm.com/software/integration/wmqe/support/.



The MQe Queue manager provides a central point of access to a messaging and queuing network for MQe applications . A queue manager owns and controls MQe messages, queues, and connections. It allows applications to access messages and queues.

MQe identifies three distinct roles for queue managers in addition to the basic queue manager functionality:

- **Client,** defined as a queue manager that supplies messages to, or gets messages from, a server
- **Server,** defined as a queue manager that provides services to many attached client queue managers
- **Gateway**, defined as a server queue manager that also has the capability to exchange messages with WebSphere MQ base messaging queue managers

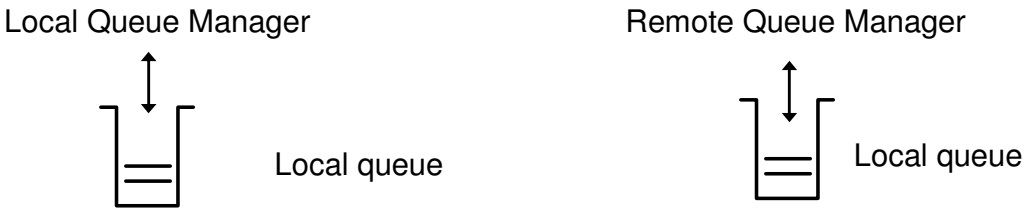
The **registry** is the main store for queue manager-related information. Note that this is not the Windows Registry. The type of storage for the registry (for example, the file system) is specified by an adapter. The default adapter is a file system adapter.

An instance of a Queue Manager must be created and then started before it can be used. Examples are provided that show how to supply the queue manager startup information from an initialization file or property file.

You may run only one queue manager per Java process, or with the case when using the native C interface, only one queue manager per process. Care should be taken not to start a queue manager twice. Two processes handling the same queues will have indeterminate results.

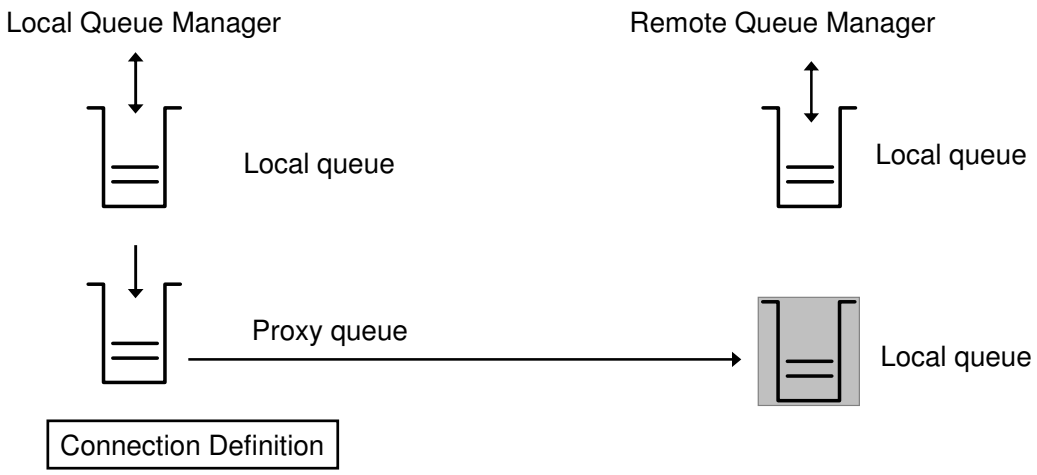
The MQe provides an extendable rules-based behavior. MQe uses rules - which are essentially user exits - to allow applications to monitor and modify the behavior of some of its major components, including queue managers.

Queue Manager roles - Basic



The role a queue manger plays in the MQe network is completely dependent upon the classes that are instantiated when the queue manager is configured or started. If a queue manager only has local queues configured, then it may not send messages or receive messages across a network. This is considered a basic queue manager.

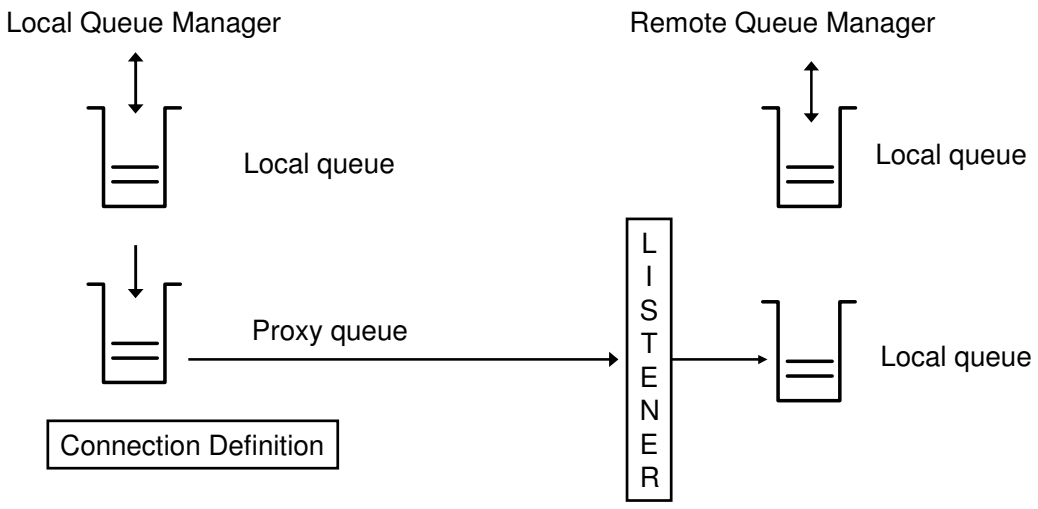
Queue manager roles – Client



The role of the basic queue manager may be extended to that of a client queue manager by configuring two objects from the MQ Everyplace toolkit. The first is a connection definition which provides the network information to contact a remote queue manager. This will include such information as the network address, the

port the remote queue manager is listening on, the communications adapter to use, and the name of the remote queue manager. To actually send a message to a remote queue manager, a proxy queue is also required. There are two types of proxy queue: synchronous and asynchronous. These will be discussed later.

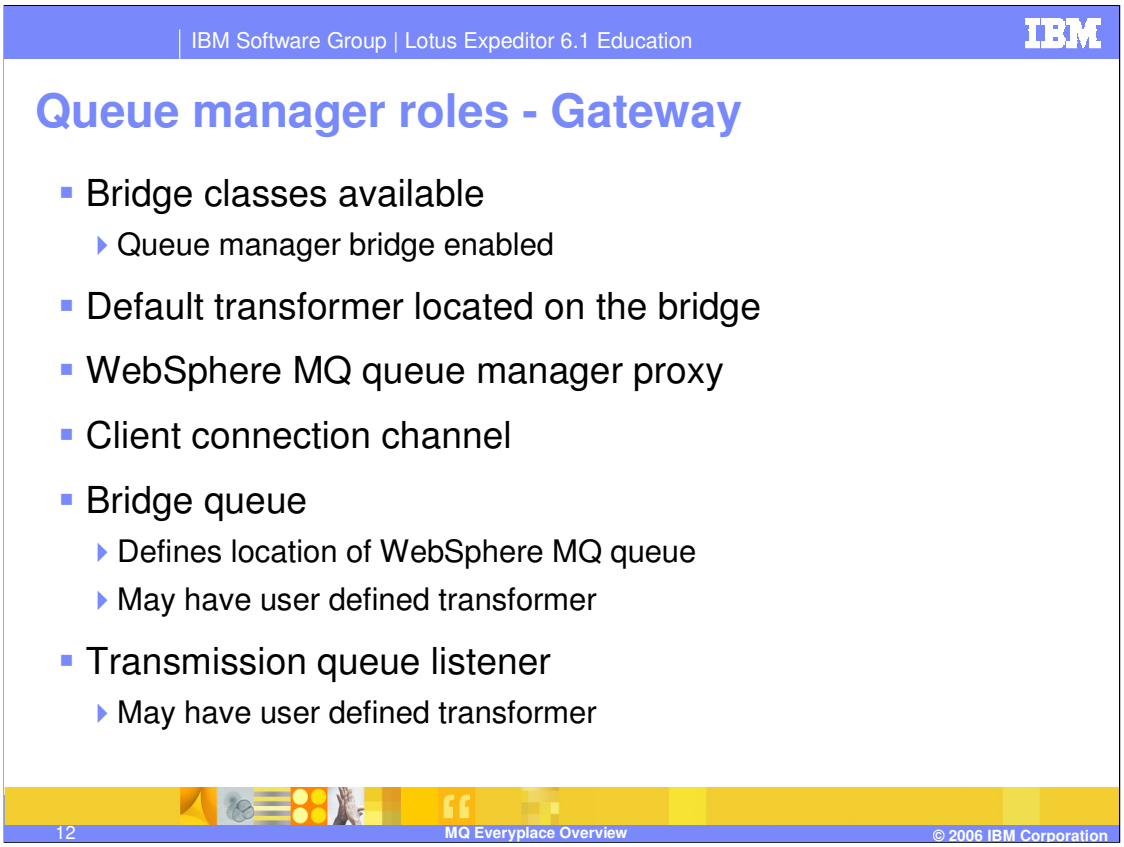
Queue manager roles – Server



In the server role, the Queue manager listens for messages being delivered on a local queue from client queue managers. Typically a Server Queue manager communicates with many client queue managers.

The role of the basic queue manager may be extended to that of a **server queue manager** by configuring a listener object from the MQ Everyplace toolkit. The listener waits on a port known to the clients using the same communications adapter as the clients. This enables the server queue manager to receive messages or requests for messages. In addition to local queues, a server queue manager may also have a store queue, which may hold messages for client queue managers.

The use of a store queue is discussed in detail later in the presentation.



In the **gateway** role, a server queue manager exchanges messages with a WebSphere MQ-based queue manager. A server queue manager must utilize the MQ bridge software to exchanges messages with MQ. This bridge uses the MQ Java client to interface to one or more MQ queue managers. Only one bridge is allowed per Java process.

A WebSphere MQ **queue manager proxy** definition is required for each MQ queue manager that communicates with MQe. Each of these definitions can have one or more associated **Client connection channels** defined, where each represents a connection to a single MQ queue manager.

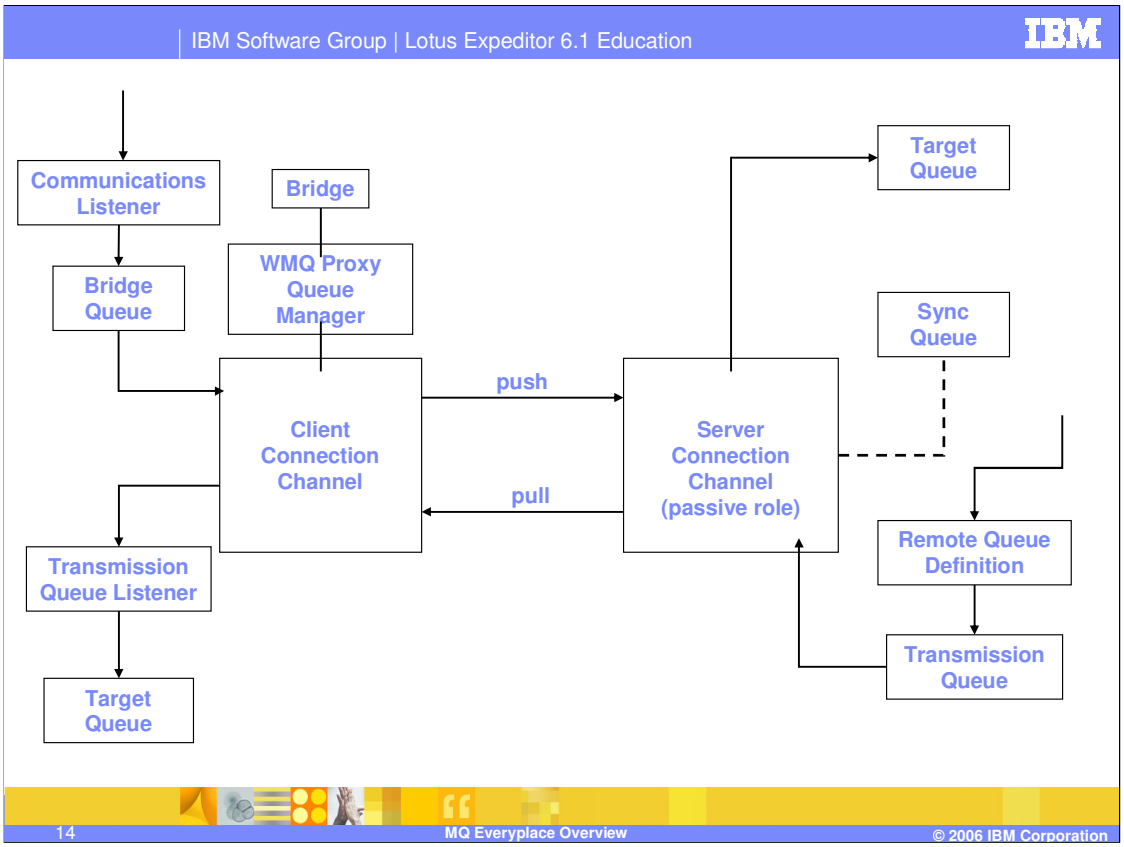
A **bridge queue** is a special form of remote queue, describing a queue on an MQ remote queue manager. Bridge queues put or get from the MQ queue they reference. In Java™ only, it uses a transformer to perform necessary data or message reformatting as each message is exchanged between the MQe and MQ systems. The definition of the bridge queue includes the location of the remote queue manager on an MQ system and any transforms that need to be performed on the MQe message before being sent to MQ.

The **Transmission queue listener** pulls messages from MQ to MQe. This resource may be defined with a transformer class for transforming messages coming from MQ to MQe.

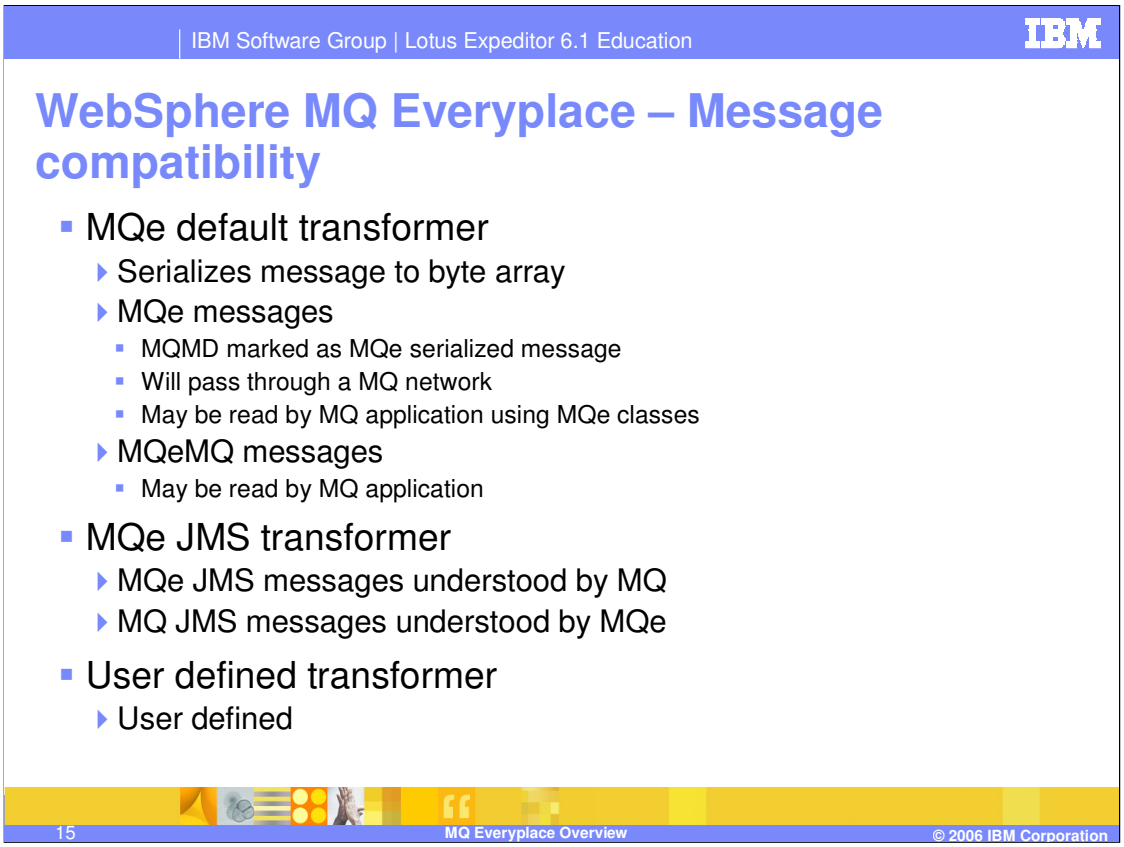
Extending the WebSphere MQ Network

- Java classes
  - Shipped as part of WebSphere MQ
- Java server connections
  - For use by the client connection in WebSphere MQ Everyplace
- Sync queue
  - One for each server connection
- Transmission queue
  - Does not hold connection information
- Remote queue definitions
  - May be remote queue manager definitions
- Local queues

When creating a gateway queue manager, it is also necessary to create objects on WebSphere MQ. The Java classes are required. For each client connection channel defined on the MQe bridge, a server connection is required. For each server connection, a sync queue is required. A transmission queue with the name of the MQe queue manager is required. This queue should not hold any connection information. Remote queue definitions are required for the queues on the MQe queue manager that will be receiving MQ messages. Also, local queues are required to receive the messages from MQe.



Once all the MQe and MQ objects have been created the topology will look something like this.



MQe messages destined for MQ pass through the bridge and are converted into an MQ format, using either a default transformer or one specific to the target queue.

A default transformer will take a MQeMsgObject, serialize it and create the necessary header information so the message may pass through the MQ network. This message will not be understood by a MQ application. To allow a MQ application to understand MQe message, it is necessary to use the MQeMQMsgObject. In addition to these standard messages, there is a JMS transformer which allows JMS messages to be passed between MQe and MQ applications. If none of the supplied transformers provide the level of support required by the application, a user written transformer can be used.

This MQeMQMsgObject class and the default transformer behavior mean that:

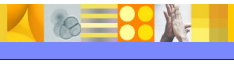
An MQe message can travel across an MQ network to an MQe network without change.

An MQ message can travel across an MQe network to an MQ network without change.

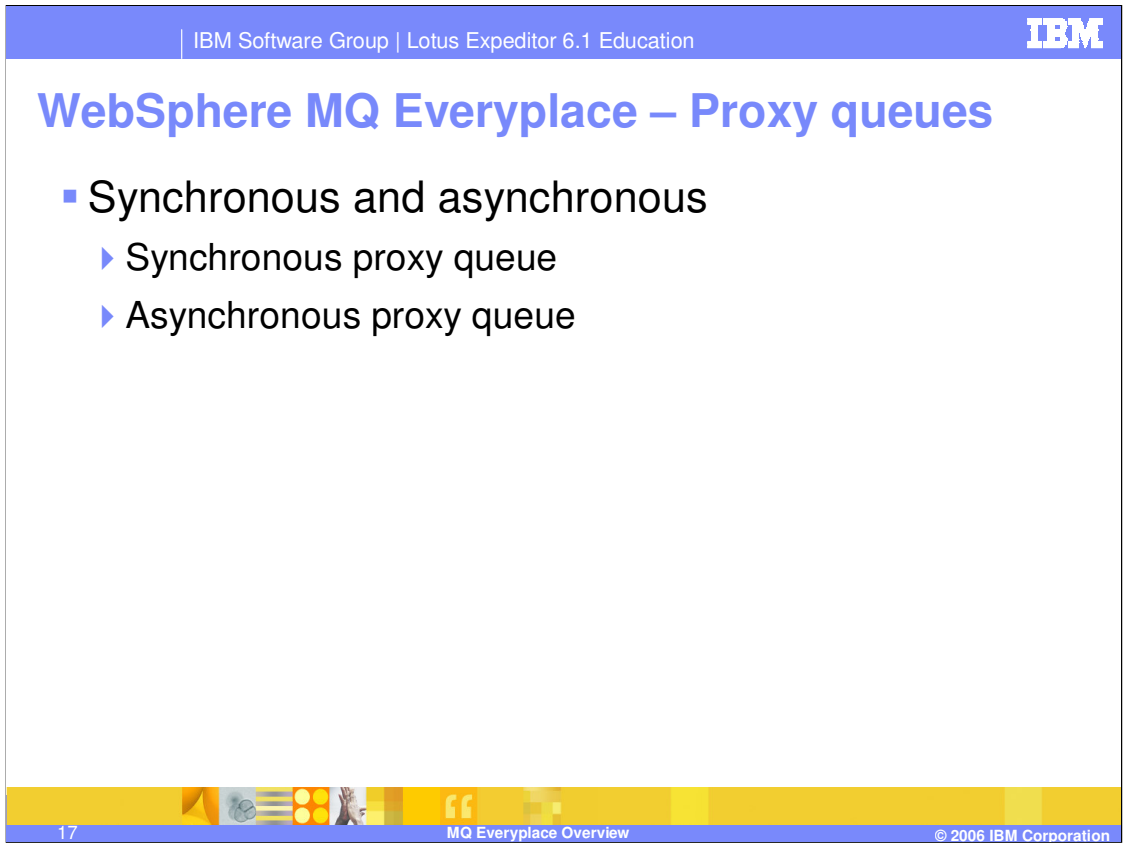
An MQe application can drive any existing MQ application without the MQ application being changed.

Section

WebSphere MQ Everyplace - Queues



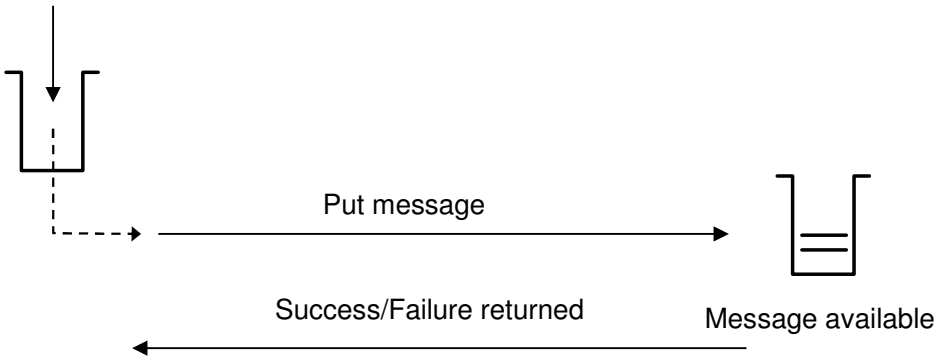
Next, let's discuss MQe queues.



The application interface to MQe is provided by the queue manager. Although queues are required in order to route messages and hold messages in persistent storage, they do not form part of the MQe API. Proxy queues provide a definition of a local queue on a remote queue manager and provide the mechanism for pushing messages from a client queue manager to a server queue manager. A **synchronous proxy queue** is purely a definition; the message is immediately transmitted across the network. Therefore if an error occurs either with the network or at the remote queue manager, an exception is immediately returned to the application.

An **asynchronous proxy queue** immediately puts the message to permanent storage. The message is then sent across the network by a background thread.

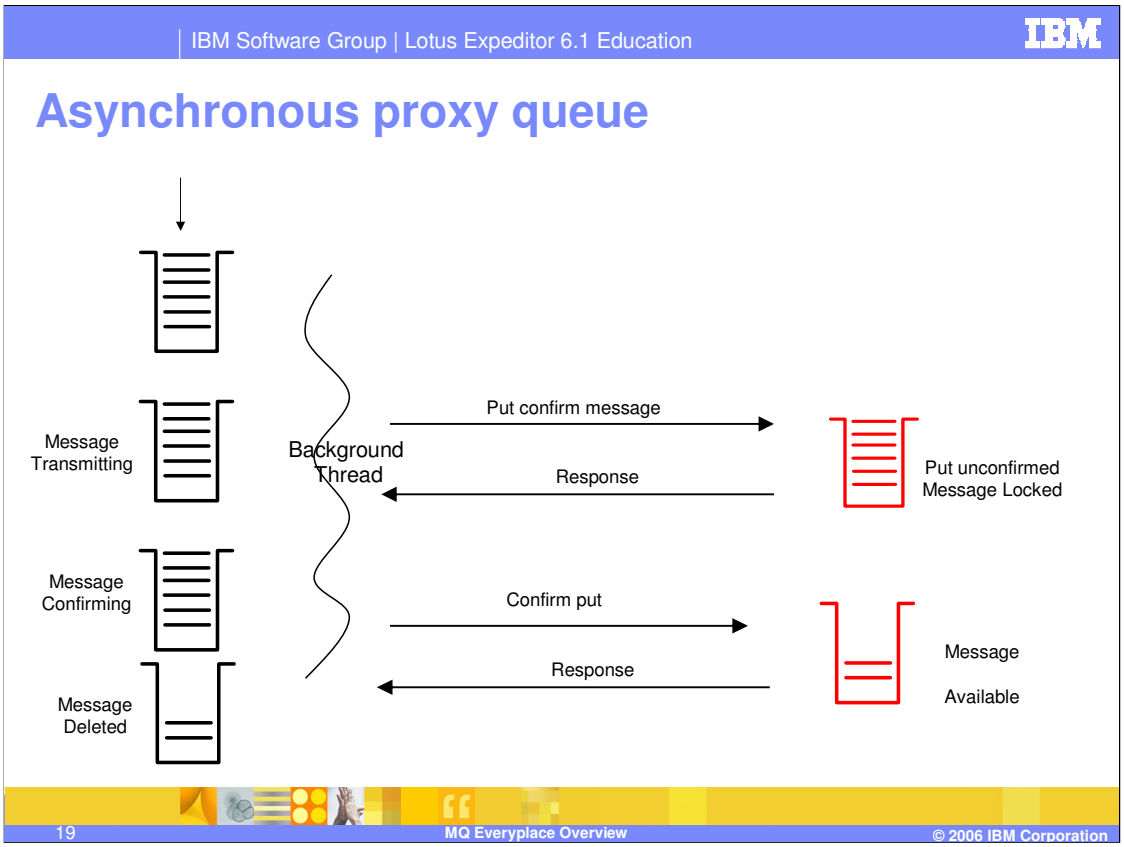
Synchronous proxy queue



The synchronous proxy queue has a simple flow:

- An API is called on the queue manager to put the message to the synchronous proxy queue
- A Message is immediately sent across the wire
- If the message is successfully put onto the remote queue, it is immediately available
- Success or failure is returned to application

A synchronous queue should be used when the application needs to know immediately if the message has been successfully put to the remote queue.



The first action followed by an **asynchronous proxy queue** is to place the message in permanent storage. If this is successful the method returns successfully to the application, otherwise an exception is thrown. Once the message is in persistence storage on the queue, it is the responsibility of the trigger transmission thread to ensure the message is correctly sent to the remote queue. Due to the nature of MQe, the fact that no assumption can be made on the presence of a network, the trigger transmission thread never throws an exception.

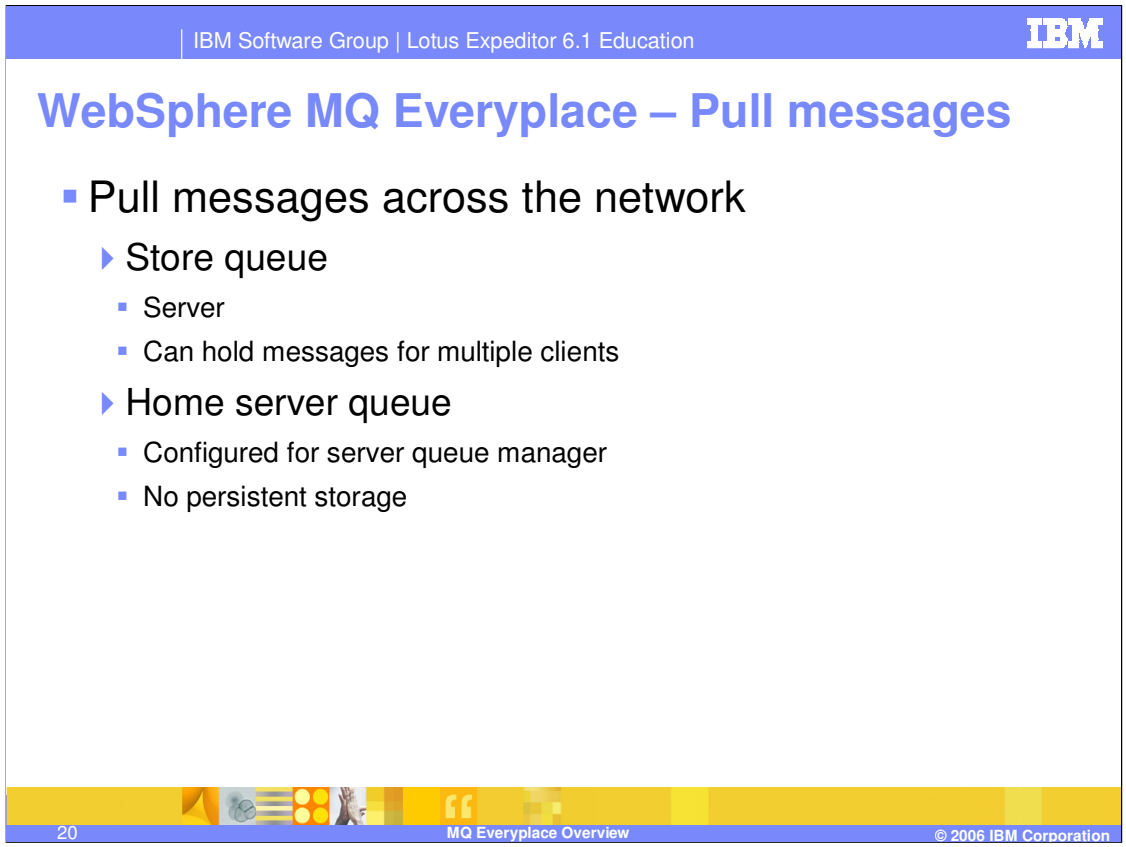
The following protocol of put with confirm ensures the client does not delete a message from an asynchronous proxy queue until it has confirmation that the message has successfully been put onto the remote queue.

• The message on the asynchronous proxy queue is changed to a state of *transmitting*, the message is sent across the network. If this is successful the message is put on the remote queue and locked in the put *unconfirmed* state.

• A response is returned to the client with success or failure. If this fails no exception is thrown, the thread will move on to the next queue.

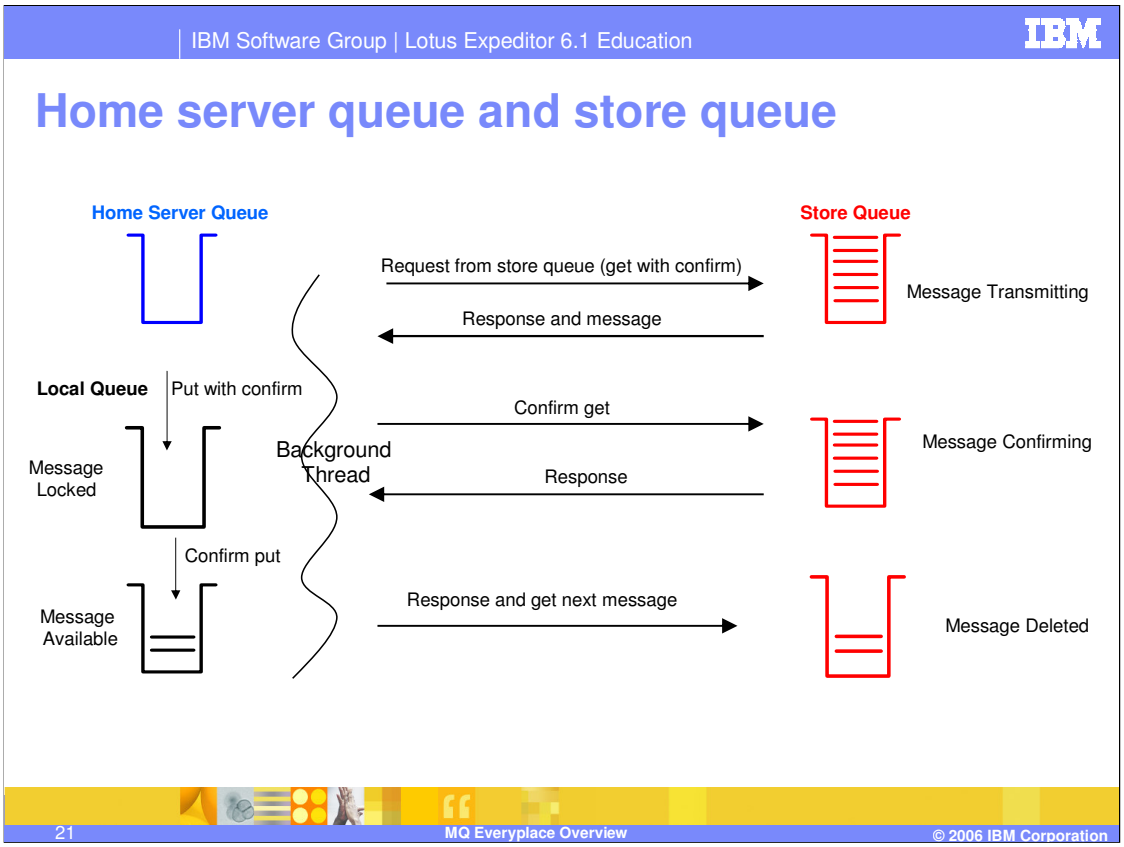
• On successfully sending the message, the state of the message on the asynchronous proxy queue is altered to *committing*. The client sends a put confirm to the remote queue. If this is successful the state of the message is changed to *confirmed* and the message is unlocked and it may now be accessed by an application.

• A response is sent by the server to the client to say the put confirm was successful. If this is received the message is removed from the asynchronous remote queue. If the response is not received by the client, maybe because the network has gone down, the message on the asynchronous remote queue is held in committing state. When the trigger transmission thread goes around the queues again, the message will be confirmed again. In the instance of a response being lost, this does not cause a problem as MQe recognizes that the message has already been confirmed and a success response is sent back to the client. This way the



The combination of a store queue and home server queue provide the mechanism for a client queue manager to pull messages from the server. A store queue holds messages for multiple queue managers defined on that queue and is located on a server queue manager.

When a request is made to see if a message is available, the store queue iterates over all the messages until one is found for the queue manager making the request. A home server queue holds information about a particular server queue manager, and on a call to trigger transmission, will request messages for the client queue manager from the store queue.



Before looking at how the home server queue and store queue provide pull functionality, it is important to note that the time interval on the home server queue should not be used – you should set the time interval to zero. This time interval is used for a background thread specific to the home server queue. The trouble is that if anything unforeseen happens with this thread, no indication is sent to the user application. Instead a specific call should be made to trigger transmission, perhaps using a rule on the queue manager.

The home server queue does not actually hold any messages in persistent storage. It puts them to the actual local queue for which the message is destined.

The background thread will do the following:

• Make a request to the remote queue manager store queue to see if there are any messages for the client queue manager (note the queue manager name is used, and not a specific queue name).

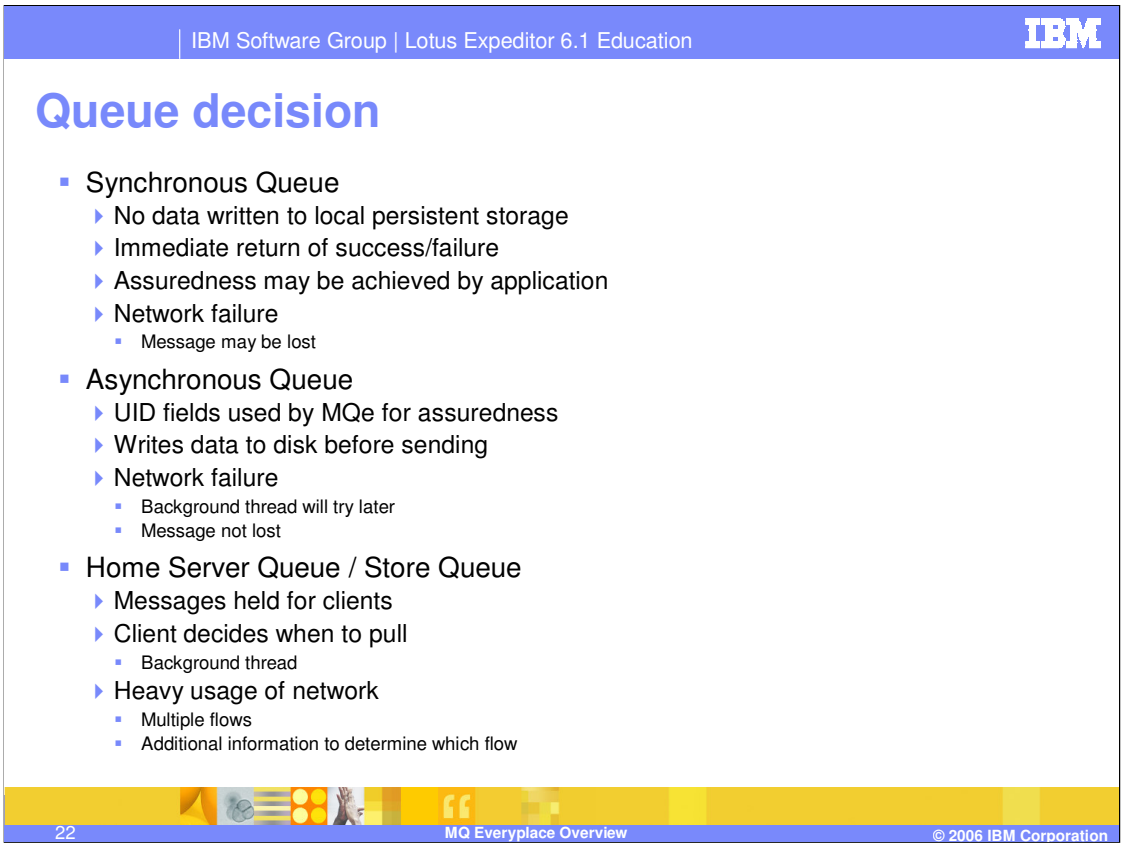
• If there are messages the message state is changed to *transmitting*. A response is sent to the client indicating there is a message, along with the message.

• The home server queue then puts the message to the local queue. The message is locked with the state of *put unconfirmed*.

• A confirm get is then sent to the store queue which alters the message state to *confirming*.

• If successful the response is sent back to the client and a put confirm is made on the message in the local queue. If successful, the message is then unlocked and becomes available to applications.

• A response is sent back to the store queue and the message is deleted. A request for a message is sent at the same time.



Deciding on which type of queue to use depends on the requirements of the application. Various approaches can be taken on how to move messages between clients and servers.

Synchronous Queues have the following characteristics:

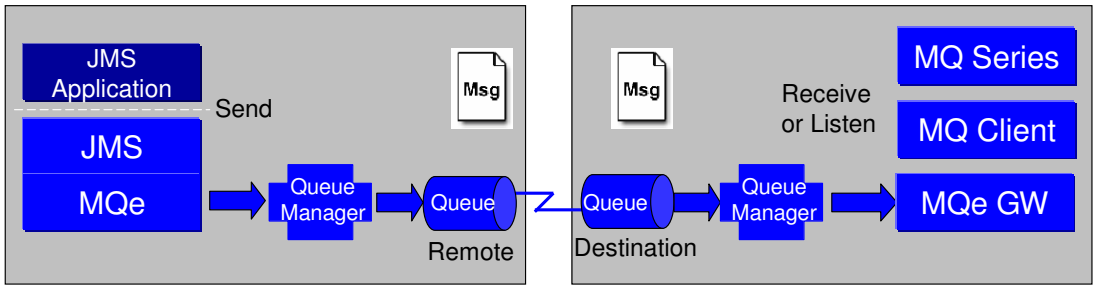
- No data is written to local persistent storage
- Immediate return of success or failure
- Assuredness of delivery may be achieved by the application
- A network failure may result in a lost message

Asynchronous Queues have the following characteristics:

- UID fields used by MQe for assuredness
- Data is written to disk before sending
- In the event of a network failure, a background thread will try later. The message is not lost.

For the Home Server Queue and Store Queue, messages are held for clients and clients decide when to pull using a background thread. This is characteristic of heavy network usage, containing multiple flows and handshakes.

MQ integration messaging end-to-end



- JMS with MQ Everyplace implements Point-to-Point (PTP) Messaging
- A Message Producer can send messages to a Message Consumer as shown
- If a network connection is not available, then messages can be queued locally for subsequent delivery when the connection becomes available

Point-to-point (or PTP) systems are about working with queues of messages. They are point-to-point in that a client sends a message to a specific queue.

The JMS PTP model defines how a client works with queues: how it finds them, how it sends messages to them, and how it receives messages from them.

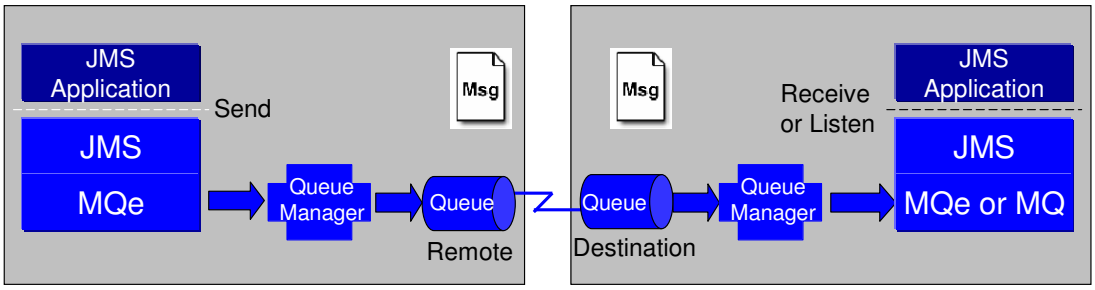
The use of JMS as the API to write MQ Everyplace applications has a number of benefits, because JMS is an open standard:

- It provides the protection of investment, both in skills and application code
- It is widely available for people skilled in JMS application programming
- It provides the ability to write messaging applications that are independent of the JMS implementations

An application can synchronously receive messages from a queue or asynchronously listen and be notified of messages in a queue.

End-to-end JMS integration messaging

(with MQ or WebSphere Business Integrator)

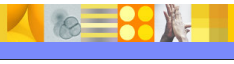


- JMS with MQ Everyplace implements Point-to-Point (PTP) Messaging
- A Message Producer can send messages to a Message Consumer as shown
- If a network connection is not available, then messages can be queued locally for subsequent delivery when the connection becomes available
- **Customers must either purchase WebSphere MQ or WebSphere Business Integration Brokers to meet WebSphere Application Server messaging licensing guidelines**

MQe JMS applications can communicate with other JMS applications built on top of MQe or MQ. The same characteristics apply as on the previous chart describing JMS MQe to MQ application interaction. Although the WebSphere Application Server ships with JMS capability, licensing terms prevent you from building an end-to-end messaging flow between a Lotus Expeditor application and a JMS application built on top of WebSphere Application Servers JMS capabilities. Instead, you must either purchase WebSphere MQ or a WebSphere Business Integration Broker product to meet the WebSphere Application Server messaging licensing guidelines.

MQ Everyplace security

- MQ Everyplace provides security under 3 different categories:
  - Local security
  - Message-level security
  - Queue-based security



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

MQe provides security under three different categories:

Local security, which protects message-related data at a local level

Message-level security, which protects messages between the initiating and receiving MQe application

Queue-based security, which protects messages between the initiating queue manager and the target queue.

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

The MQe security features of all three categories protect message data using an attribute, for example MQeAttribute. Depending on the category, the attribute is applied either externally or internally. Each attribute can contain the following:

Authenticator, which provides additional controls to prevent access to the local data by unauthorized users

Cryptor, which controls the strength of protection required

Compressor, which optimizes the size of the protected data

Key, which controls access by requesting a password

Target entity name, which requests the target queue name

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

MQ Everyplace trouble shooting

- Lotus Expeditor information center contains several sections on trouble shooting:
  - Problem Determination
  - Common Problems
  - Tracing and Logging
  - MQ Everyplace Diagnostic tool

Refer to the Lotus Expeditor information center for information on trouble shooting problems with Lotus Expeditor and MQ Everyplace. In addition, the server SupportPac tools provide serviceability tools and problem determination information.

Refer to the MQ Everyplace User Guide for information about the trace window which provides a means of debugging MQe applications, MQe_Explorer and MQe. See the Troubleshooting section for lists common errors and solutions.

Trademarks, copyrights, and disclaimers

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

Everyplace IBM Lotus SupportPac WebSphere

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

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

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

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.

This concludes the presentation.