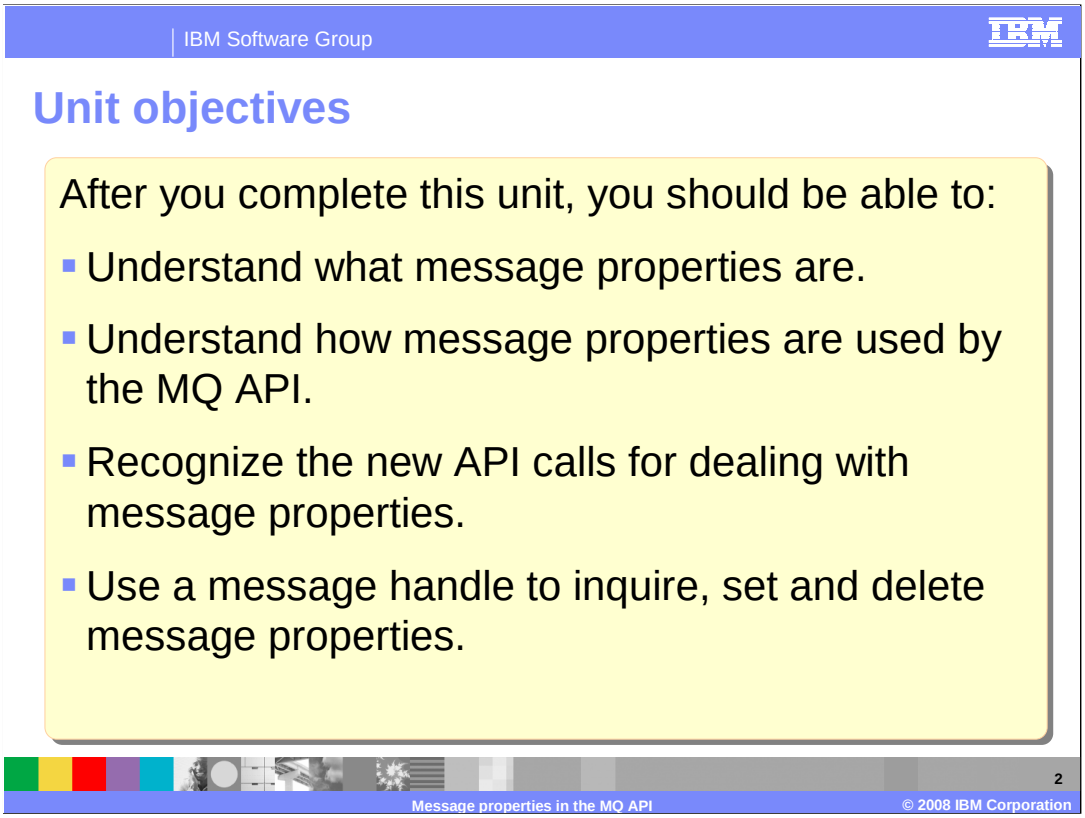
This unit covers the MQ API (or MQI) enhancement in WebSphere MQ version 7.0 covering the new message properties features added in this release.

This unit assumes a reasonable understanding of the existing WebSphere MQ API for putting and getting messages to and from a queue.



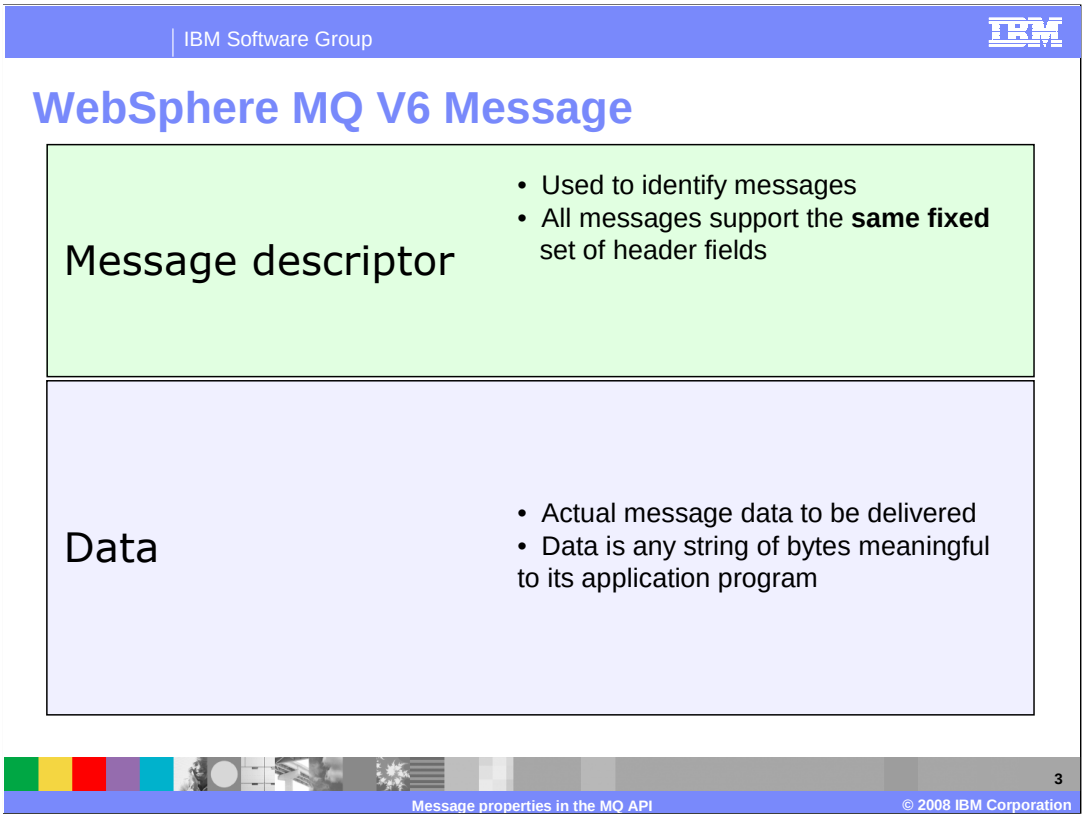
After you complete this unit, you should have some understanding of what message properties are and why they have been introduced.

Understand the concepts of message handles and the types of message property available.

With the assistance of the information center you should be able to code MQ7 applications that create message handles and use these handles to inquire and set message properties.

Additionally you should understand how the information in message properties is passed to older version queue managers.

This unit does not attempt to cover the full range of syntax and options available, for which you should refer to the product information center.



There is a type of application that it needs to add information to a message, this information is used by some applications during the lifetime of the message but ignored by other applications. Examples might be routing information, accounting and billing information and a whole range of metadata not directly concerned with the message payload. Many vendor products might want to “enrich” messages in this way.

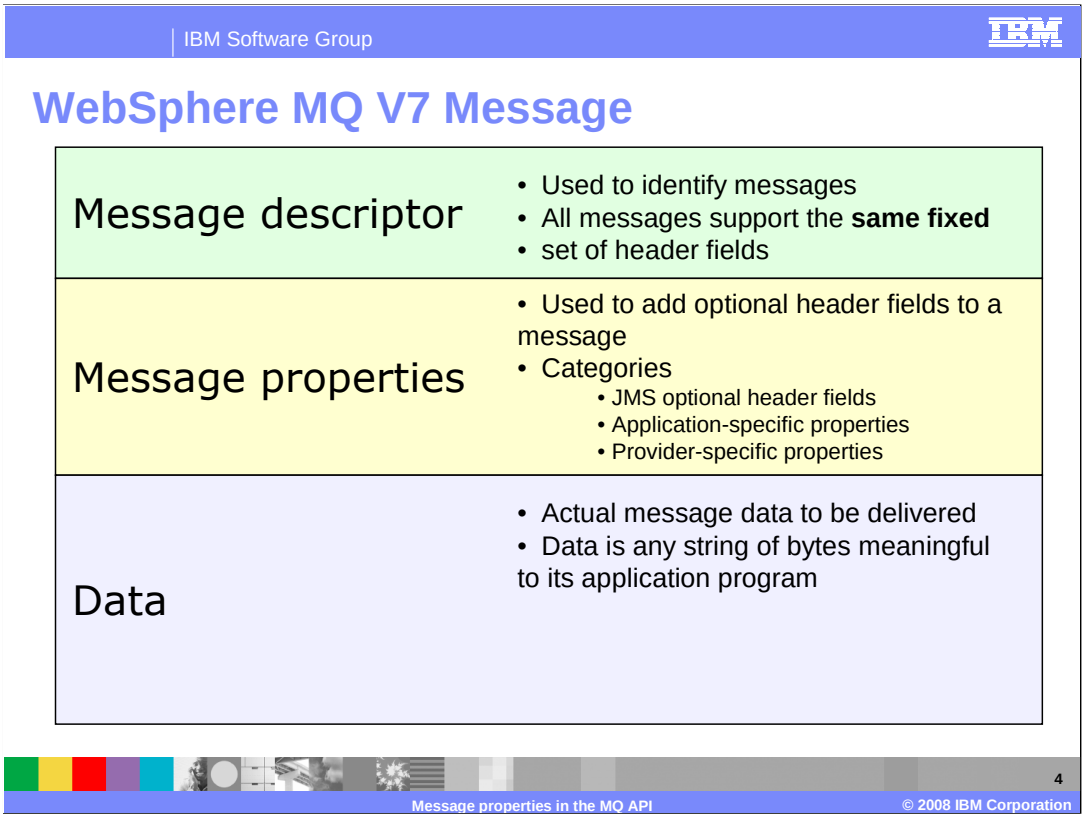
WebSphere MQ version six and earlier messages have two parts, the Message Descriptor (or Header) and the message data (or payload). The Message Descriptor is a fixed set of fields and so there is only one place that applications can add such information – the message payload.

This means that all applications reading the message must be aware that the data is present and know how to ignore it if they don’t need it.

In V7 applications can add properties to messages using new MQI calls. A property is a named piece of data. It is not treated as part of the message payload.

The properties are accessed by means of a message handle passed to MQPUTs and MQGETs in the MQPMO and MQGMO.

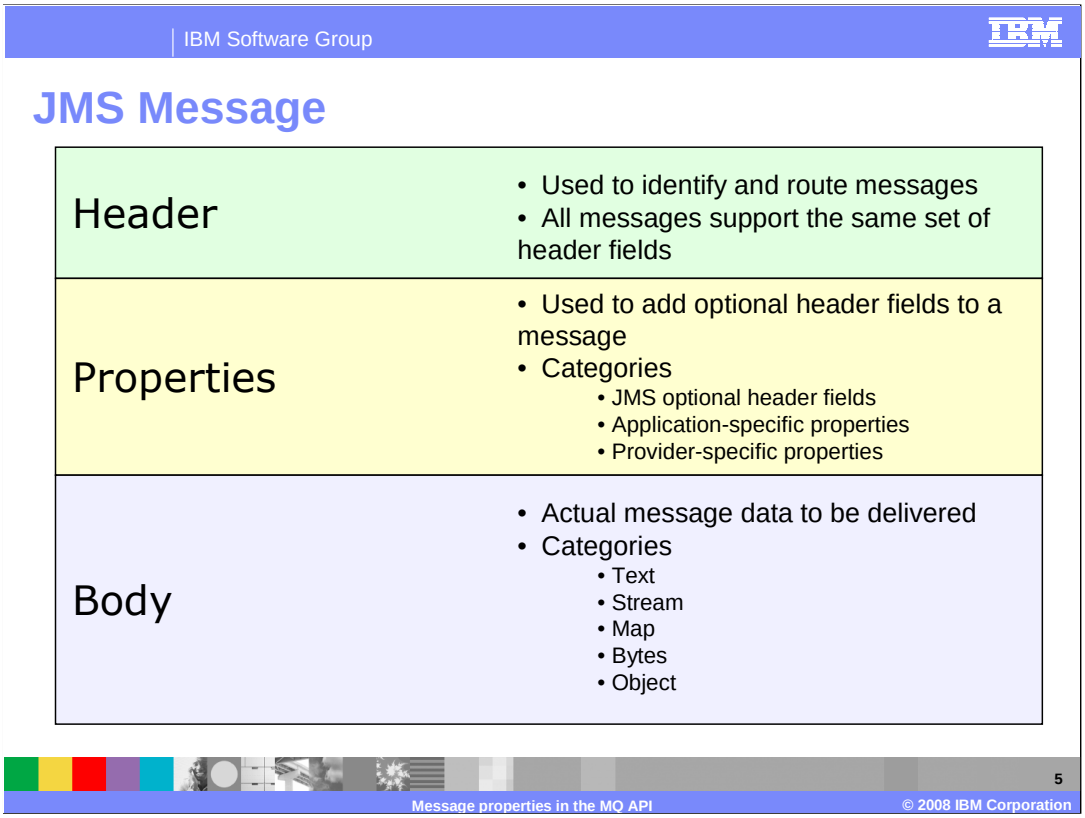
Applications will only be aware of this information if they choose to access it.



In version 7.0 WebSphere MQ introduces another component to the message, in addition to the fixed descriptor and the application data containing the payload a “Message Properties” component is added.

The message properties are a set of name, value pairs. The name-space for the property names is made hierarchical by splitting the name into components separated by dots. Some of those hierarchical categories are reserved for use by specific components of MQ, including the JMS component.

Any name beginning with “Root.” for example.



Looking at the makeup of a JMS message you see that the broad structure of native MQ messages now matches that of the JMS message.

Indeed, the JMS component of MQ now indeed uses the newly introduced message properties in order to carry the JMS properties data. This makes the interaction between JMS and MQ API applications much simpler.

One of the features of the JMS messaging model is the ability to use selectors based on any of a message's properties to select messages to be read. In version 7.0 this is introduced into the native WebSphere MQ API.

So both in the handing of message properties and in the ability to perform message selection WebSphere version 7.0 provides natively the functions required of a JMS provider.

What is a message property?

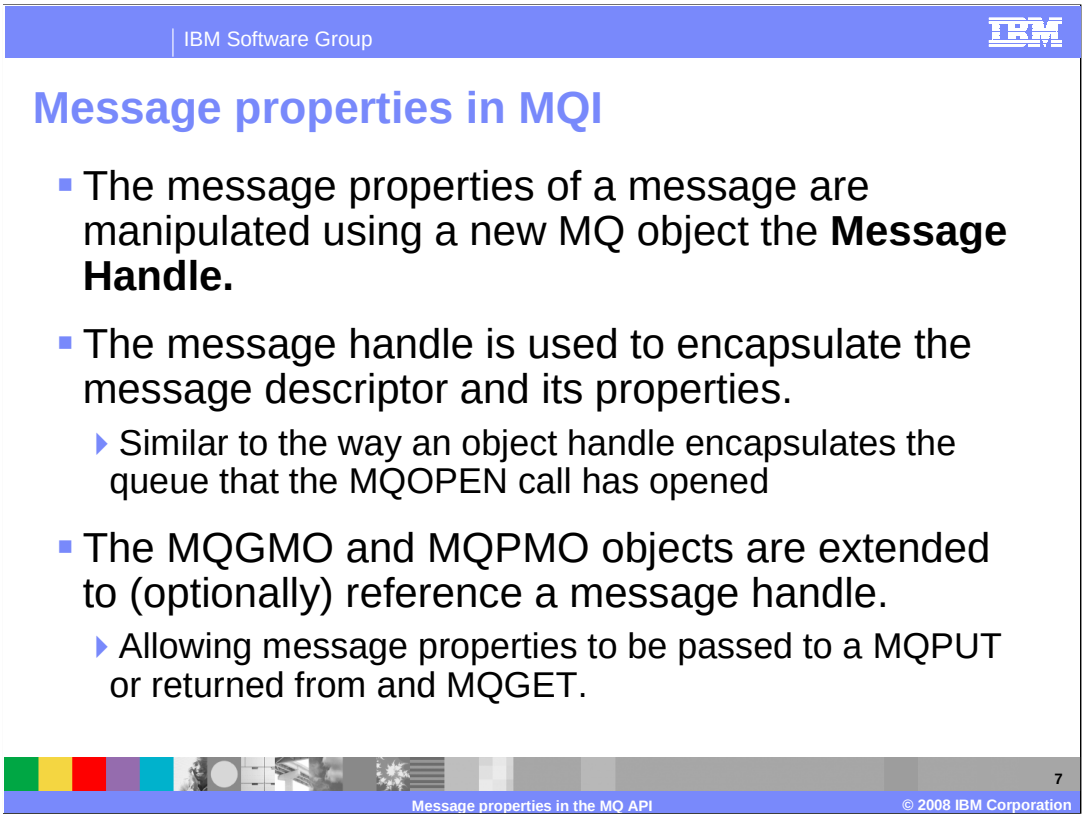
- A message property is a *named* piece of data which is associated with a message.
- Property names are made hierarchical by use of the "."
  - "A.X" and "A.Y" are both part of "A".
  - Applications are recommended to use a recognizable prefix such as domain name – com.any.company……
- The property has a value of a specified type:
  - MQBOOL – a Boolean
  - MQBYTE[ ] – a byte string
  - MQCHAR[ ] – a string
  - MQFLOAT32, MQFLOAT64 – 32/64-bit floating-point numbers
  - MQINT8, MQINT16, MQINT32, MQINT64 – 8/16/32/64-bit integers
- Each message property has an associated property descriptor
  - The property descriptor describes attributes of the property itself.

As mentioned earlier the message properties are name, value pairs.

Looking first at the name. A property name is made up of a hierarchical set of simple names separated by the "dot" or full stop character. Some high level (left hand side) names are reserved by WebSphere MQ. For example "mq.". Some are also likely to be used by other applications so use of a prefix, such as domain name, for your own data is strongly recommended.

Looking now at the values a property can have. In addition to strings, byte arrays and Booleans version 7.0 has added data types to represent floating point numbers and integers of various sizes.

In addition to its name and its value each message property has a descriptor that gives additional information to the queue manager about how to handle the property. The descriptor includes a "CopyOptions" field which indicates whether the property should be copied when messages are being forwarded, replied to or published.

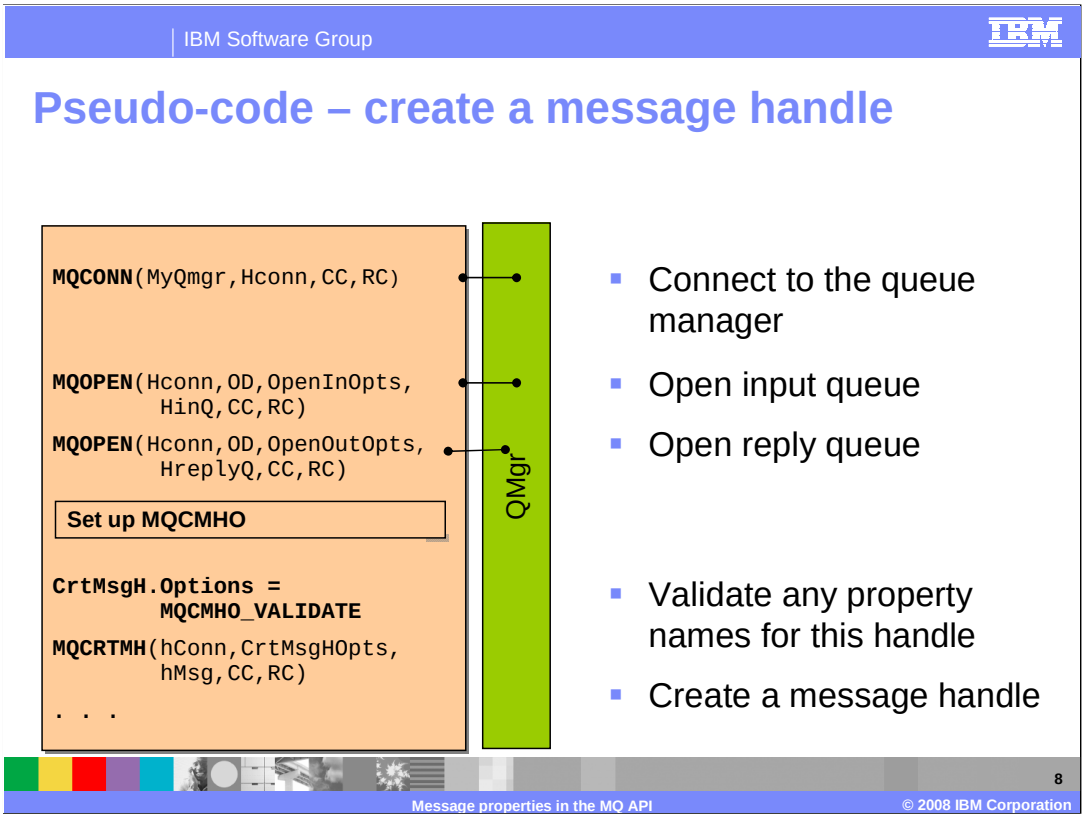


A new MQ object the Message Handle is introduced in version 7.

This message handle encapsulates the set of properties for a message. MQ API calls are provided to create and destroy message handles; to examine, add and update the properties in a message handle.

When using the MQPUT call to place a message on a queue message handles in the Put Message Options are used as the source of the message properties for the message. The same is true when MQPUT is publishing to a topic.

When using a MQGET to retrieve a message then the get message options object can supply a message handle to receive the message properties associated with the message being retrieved.



The next few slides look at the code that might be used in a simple application that is receiving messages and replying to them.

It must be stressed that this is an extremely simplified flow and in no way complete. The information center supplies full information on coding these calls

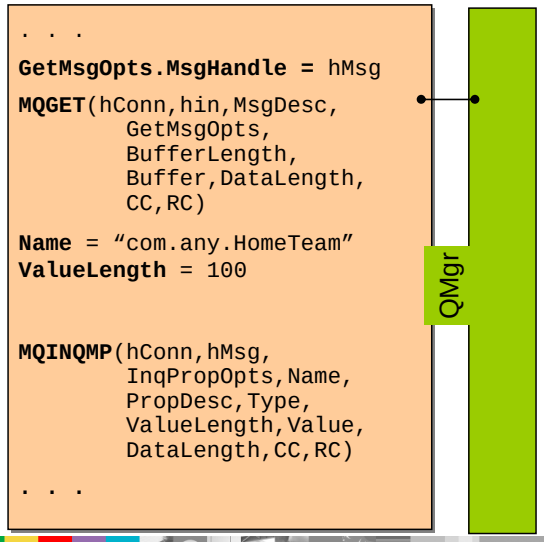
The MQCONN Call connects to the queue manager and returns the connection handle to the queue manager.

Next the input and reply queues are opened.

Now comes the first new element the creation of the Message Handle.

The Create Message Handle option is set to validate the property names, the new MQCRTMH call is made to create and return the new message handle.

Pseudo-code – return and inquire message properties



- Set hMsg in MQGMO to get message with properties
- Inquire for property "HomeTeam", allowing100 bytes for the return area.
- **Note**: MQINQMP call does not pass to Queue Manager.

The first line on this slide shows how to signal to the queue manager that a message handle allowing access to message properties is to be returned from the subsequent MQGET.

A message handle is placed in the Get Message Options (MQGMO) object

On return from the MQGET the message handle contains the encapsulation of all the message properties associated with the returned message.

This example then inquires the value for a particular property "com.any.HomeTeam" to be requested.

The MQINQMP call returns the property

Variations on the MQINQMP call are available that return all properties and allow the caller to iterate around them.

It is important to note that neither the MQCRTMH call, nor the MQINQMP call, require to be passed to the server but are processed inline on the client side. This means that manipulating message handles and message properties can be done without expensive calls to a, possibly remote, queue manager.

Pseudo-code – set properties and MQPUT

```
MQCRTMH(hConn,CrtMsgHOpts,
        hMsgNew,CC,RC). . .

Name = "com.any.ResultType"
Type = MQTYPE_STRING
Value = "DRAW"

MQSETMP(hConn,hMsgNew,
        SetPropOpts,Name,
        PropDesc,Type,
        ValueLength,Value,
        CC,RC)


PutMOpts.OriginalMsgHandle=hMsg
PutMOpts.NewMsgHandle=hMsgNew
PutMsgOpts.Action = REPLY

MQPUT(hConn,hout,MsgDesc,
        PutMsgOpts,
        BuffLength,
        Buffer,CC,RC)
. . .
```

- Create new message handle
- Set new properties to be added to output message
- Pass the original and new message handles and signal that this is a REPLY Message.
- Put the message

This slide illustrates a simple case of setting a property in a reply message.

First a second message handle is created to contain the new properties.

The new string property "com.any.ResultType" is set in the newly created message handle.

The next step is the setting up of the Put Message options. This is a little more complex than in the get case.

When putting a message it is possible to have two sets of message properties involved. This is especially true when forwarding a message or replying to a message.

The first set of properties is for the original message, these are treated as read only by the MQPUT operation and some of the properties are copied to the REPLY or FORWARD message.

The property descriptor of each property together with the ACTION put message option determines if a property is propagated to the new message.

For all the MQ and JMS properties the information center tabulates which are copied and which are not.

In the case illustrated the action indicates that a REPLY is being carried out. The original message handle with the properties returned on the earlier MQGET are passed in the call. These are merged with those in the new message handle which contains the new properties.

The MQPUT operation will:

Examine the Action from the Put message options.

As a REPLY message is indicated it will add all the properties from the original message handle whose descriptors indicate they are propagated to reply messages.

Merge all the properties from the new message handle.

Merge any of the properties generated as a result of the MQPUT operation

Return in the new message handle all the properties of the message just put.

Once more note that the MQSETMP calls do not involve a call to the queue manager bur are performed inline.

And once more recall that this is a very much simplified flow.

API summary

- **MQCRTMH** creates a message handle
- **MQDLTMH destroys** a message handle.
- **MQINQMP** inquires a property from a message handle.
  - Can iterate over properties
- **MQSETMP** sets a property in a message handle
  - Can set application or MQ defined properties.
- **MQDLTMP** deletes a message property from a handle
- **MQMHBUF and MQBUFMH**

Here is a summary of the major API calls related to the message properties.

The MQCRTMH call is used to create message handles for subsequent use in manipulating message properties.

One thing not referred to earlier was the lifetime of a message handle. These message handles will persist until the application disconnects from the connection that the message handles were associated with, when they are deleted implicitly. Alternatively the MQDLTMH call can be used to delete the handle programmatically. One exception is the use of message handles that have no connection affinity, these are created using a special connection handle value of MQHC_UNASSOCIATED_HCONN, in this case the message handle must be explicitly deleted.

The MQINQMP and MQSETMP calls are used to inquire and set the message properties in a message handle as seen earlier. MQINQMP can also iterate through some or all of the properties.

The MQDLTMP call deletes a particular property from a message handle.

The last two calls are included for completeness. MQMHBUF will take a message handle and return a MQRFH2 version of the properties in a buffer, MQBUFMH performs the reverse operation. These functions are likely to be of particular use in API exit routines.

Full details of all these, and other, API calls can be found in the information center.

Compatibility with older versions

- In V6 MQRFH2 headers are used to store MQ JMS properties.
- Messages containing RFH2 headers might be passed to version 7.0 queue manages.
  - This is OK, the queue manager can parse the contents.
  - V7.0 applications can choose how to have the properties presented.
- Messages with message properties might be read by a pre version 7.0 client
  - This is OK the properties can be delivered as MQRFH2 headers
  - Fully partially or not at all – this is a queue attribute.
- Messages with properties might be sent down a channel to a pre version 7.0 queue manager.
  - The properties are passed as MQRFH2 headers
  - Fully partially or not at all – this is a channel attribute

Older queue managers are of course unable to manipulate message properties. However applications might need to access some of the information contained in message properties.

Because the MQ JMS implementation in older MQ versions used MQRFH2 (MQ Rules and Formatting header v2) headers, they are the chosen mechanism to pass any message properties to these older applications.

The problem with MQRFH2 headers is that they are part of the message body and unless an application is expecting the header and knows how to parse it a problem will occur.

When a version 7.0 queue manager receives a message with MQRFH2 headers it is quite able to parse the contents and can deliver the headers to the application programs either in the version 7.0 style. That is in a message handle, or as MQRFH2 headers. This is an application choice and is controlled by get message options – detailed in the next slide – or by queue properties if no relevant get message option is supplied.

If an MQ6 application connect to version 7.0 queue manager and reads messages with message properties then these properties are ignored or passed as MQRFH2 headers depending on the queue definition.

Finally messages with properties might be sent down a channel to an older queue manager, a channel attribute can be used to control the MQRFH2 headers that are sent down the channel.

Property Control Attributes

PROPCTL is a new attribute of a queue or of a channel

- **COMPAT** – if any V6 JMS properties are in a message
  - Then **ALL** properties are returned
  - Else **NO** properties are returned
  - This is the default.
- **NONE** – all properties are removed from messages.
- **ALL** – all properties included in MQRFH2 headers.
- **FORCE** – properties returned in MQRFH2 header only.
  - Properties are not accessible by message handle even if application supplies one.
  - Not for channels

These are the property control attributes that can be specified on a queue.

The reasoning behind the COMPAT default goes like this.

If the message has any V6 JMS attributes then probably the application processing it knows how to parse and process MQRFH2 headers. In that case MQ delivers all the properties to it – the application should be able to find the properties it needs. These JMS properties are the same properties that might also be set by WebSphere Message Broker.

If the message had no properties then perhaps the application cannot parse MQRFH2 headers so do not deliver any.

This value always defines what MQRFH2 headers are supplied to older MQ clients connecting to a version 7.0 queue manager.

Version 7.0 clients can use equivalent get message options values to override the queue definition EXCEPT in the FORCE case where the queue property overrides the get message options.

The COMPAT, NONE and ALL options, but not FORCE, can also be set on Sender, Server, Cluster Sender and Cluster Receiver channels.

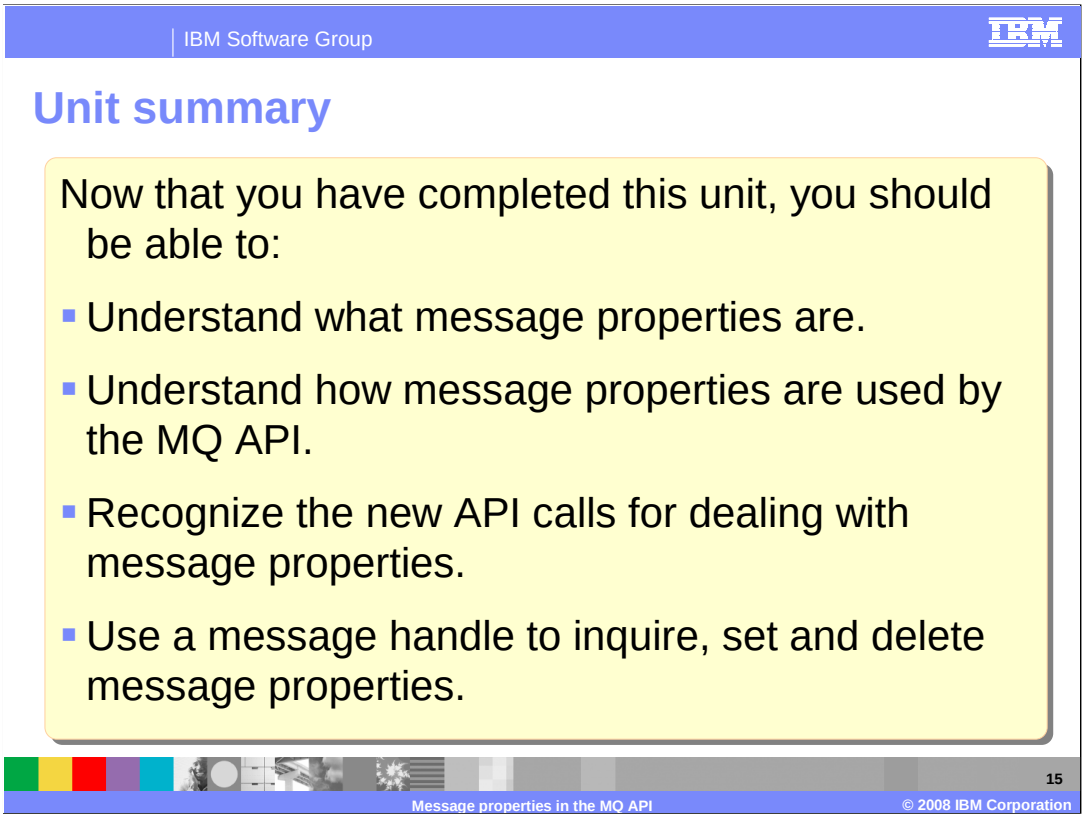
MQMD properties

- Fields in the message descriptor can also be handled as message properties
  - Root.MQMD.<Field> as the message property name, where <Field> is the C language declaration for the MQMD field name
  - For example, Root.MQMD.MsgType
  - StrucId and Version fields are not available as message properties
- This gives JMS applications access to and ability to set MQMD properties

All the fields in the fixed header on MQ messaged, the MQMD, can also be accessed and set as message properties.

The stem Root.MQMD is used for these properties.

A consequence of this is that the JMS API can now be used to access and set all fields in the MQMD. Additionally the entire MQ message payload can be accessed as a byte stream by a JMS application.



This completes the introduction to the new message properties features.

It has covered what message properties are and why they have been introduced. How message handles are created and the types of message property available.

The outline of an application using message properties has been shown. With the assistance of the information center you should be able to code MQ V7 applications that create message handles and use these handles to inquire and set message properties.

This unit did not attempt to cover the full range of syntax and options available, for which you should refer to the product information center.

This is the end of this unit.

Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send e-mail feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_iea_320_wmqv7_API_2_GetSet.ppt

This module is also available in PDF format at: ../iea_320_wmqv7_API_2_GetSet.pdf

You can help improve the quality of IBM Education Assistant content by providing feedback.

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:

IBM WebSphere

A current list of other IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

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.