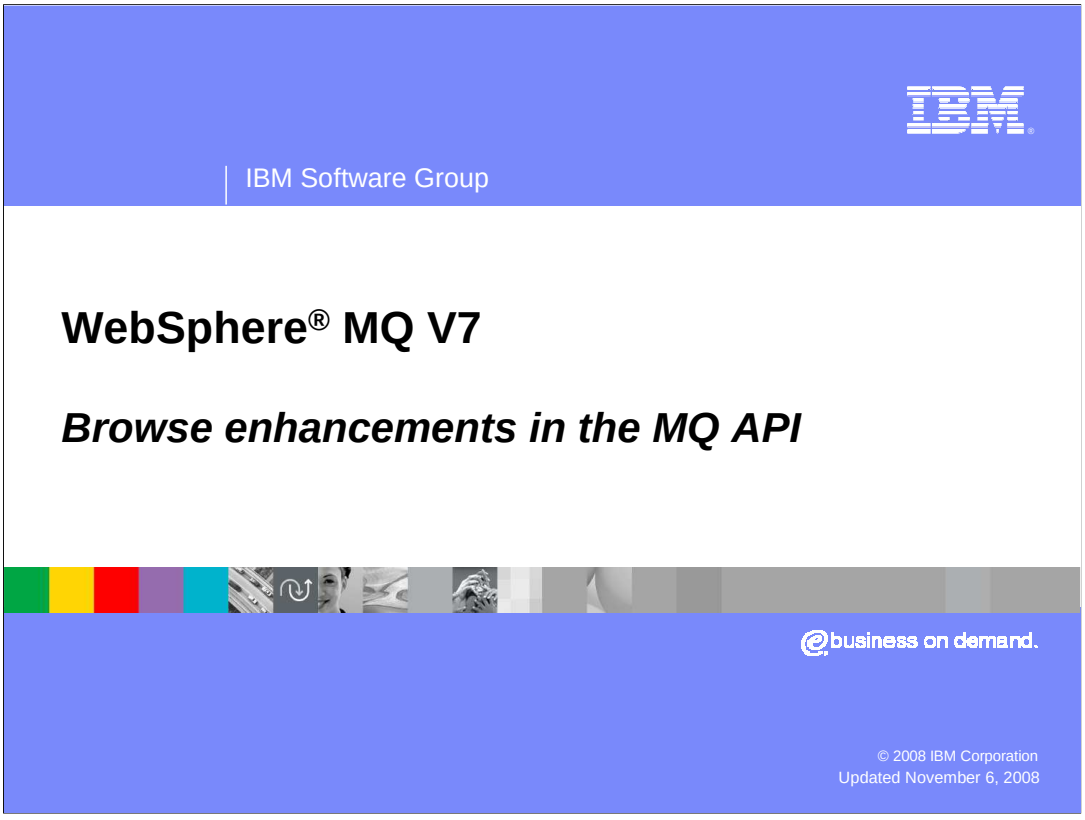
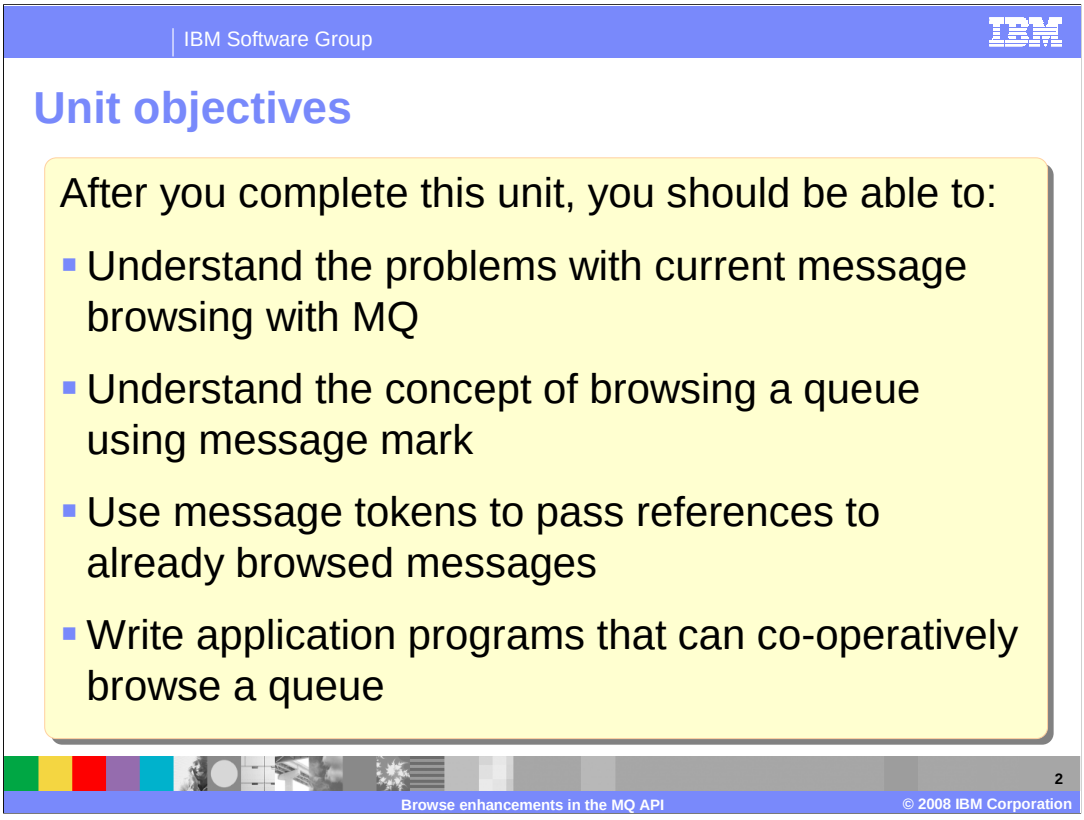



This presentation covers browse enhancements in the MQ API.




After you complete this unit you should have some understanding of the problems associated with browsing messages in MQ Version 6 and earlier. Understand the concepts of marking messages as already having been browsed and be able to use the message mark features introduced in MQ7. Use the message token to pass references that allow applications to efficiently read a specific message. And to write application programs that can co-operatively browse a queue.

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.

## Message browsing in MQ V6

- Browsing is non-destructive form of MQGET
- Browsing a queue uses
  - Cursor
  - MQGMO_BROWSE_FIRST
  - MQGMO_BROWSE_NEXT
- Problems with
  - Priority Inserts
  - Rollbacks




First to clarify what message browsing means in MQ terms. A browsing application is an application reading the contents of a message without removing it from the queue – a non-destructive get. Commonly this is used to read through a queue until a particular, or suitable, message is found in which case it is processed.

A particularly important example of this is a “dispatcher” process which looks through a queue for items of work that are ready to be run.

Think of a queue of payments to be made, the queue is browsed looking for payments whose payment date and time has passed; those messages are processed either by the dispatcher task itself or by some other “worker” thread.

In MQ V6 browsing a queue is carried out using a cursor together with get message options called MQGMO_BROWSE_FIRST and MQGMO_BROWSE_NEXT.

The Cursor is specific to your application and is used to mark the point in the queue where your next message will be read from.

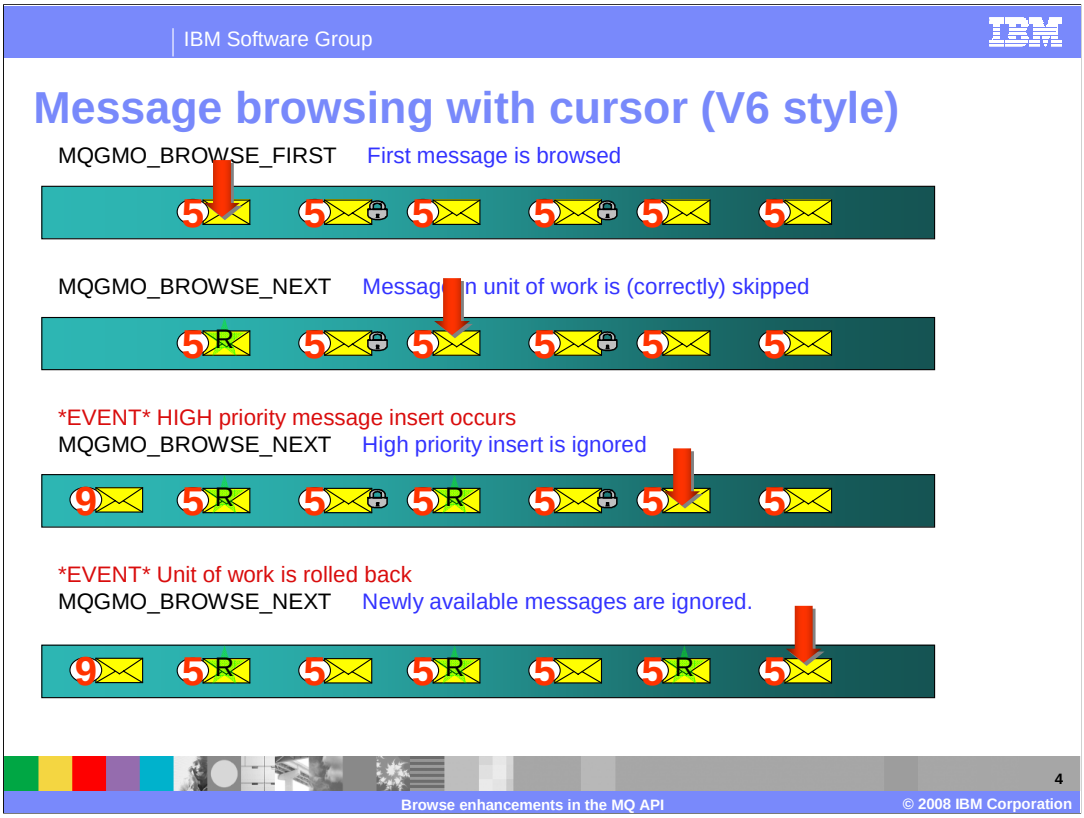
MQGMO_BROWSE_FIRST returns the first message in the queue and sets the cursor to that message.

MQGMO_BROWSE_NEXT returns next message and advances the cursor position in the queue.

If no activity occurs on the queue, this browse will return all the available messages in turn ending when no further messages are available.

However two major problems do occur.

Both relate to messages not being processed in what might be considered the “correct” sequence. The first relates to high priority messages inserted at the “front” of the queue, the second to messages that are “returned” to the queue as a result of a Unit of Work being rolled back.




This slide illustrates the two problems with browsing using a cursor.

The four illustrations of a queue show how the messages are browsed and the cursor moves.

Start with a queue of messages all having the same priority of five. Of the six messages on the queue two have been got by some other application and are therefore not available to be browsed. These two messages are shown as locked with the padlock symbol.

The first MQGET is made with MQGMO_BROWSE_FIRST option, and the first available record is read and the cursor set.

A MQGET with the MQGMO_BROWSE_NEXT returns the next available message, correctly skipping over the locked message that is part of some uncommitted unit of work.

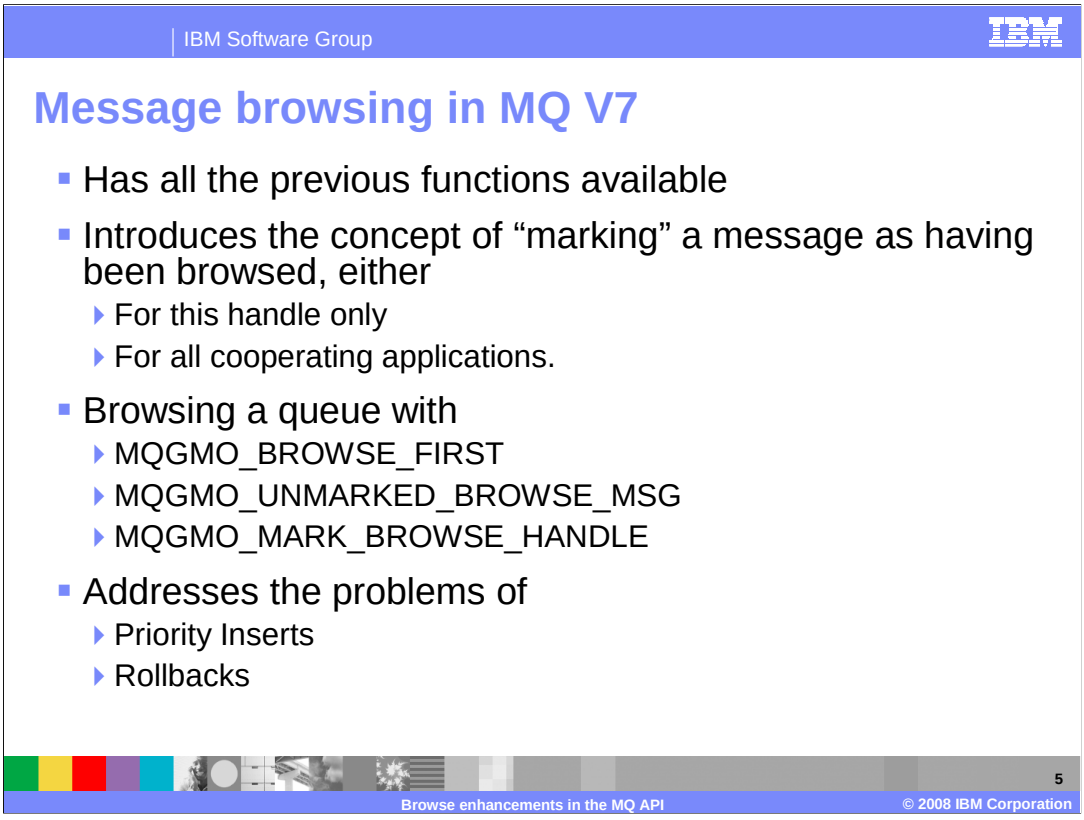
Next some other task adds a high priority (priority nine in fact) message to the queue. But this queue is added at the front (left in the slide) of the queue, this is not in the direction the cursor is moving and it is ignored, you continue by returning the record indicated in the third picture on the slide.

Now the Unit of work that had locked the two messages that have been skipped is rolled back, releasing the messages to be browsed. But once again because the cursor has passed this point the messages are ignored and not returned.

At this point you can see that although the objective was to browse the queue from the front to end (left to right) the current state has unread messages to the left of the cursor which will not be processed until you reach the end of the queue and restart the process.

To be clear this is not a product defect, it is a well documented consequence of using browse with cursor in this manner.

WebSphere MQ7 provides functionality to overcome these problems.




First thing to stress about message browsing in MQ7 is that all the previous browsing options continue to be available. No applications need to be changed but new features are available.

The concept of marking a message as having been browsed is introduced.

When a message is browsed by an application it can be “marked” to say it has already been seen. This “mark” can be at the handle level – applying to this application only, where each application browsing the queue would have its own set of messages that it had marked, or the cooperative mark can be set, this is a set of marks that are visible to all applications using cooperative marking.

Browsing the queue by repeatedly using a sequence of MQGET calls each of which use the combination of MQGMO_BROWSE_FIRST, MQGMO_UNMARKED_BROWSE_MSG and MQGMO_MARK_BROWSE_HANDLE will each time return the “first unmarked message and mark it”.


This slide illustrates what happens to the queue in the previous example where, instead of using browse with cursor, this time each of the MQGET calls specifies the options:

MQGMO_BROWSE_FIRST – saying in each case start at the front of the queue,

MQGMO_UNMARKED_BROWSE_MSG – saying only return messages that are unmarked,

MQGMO_MARK_BROWSE_HANDLE – saying mark each message returned as having been browsed by this application.

Start with the same queue of messages all having the same priority of five. Of the six messages on the queue two have been read by some other application and are therefore not available to be browsed. These two messages are shown as locked with the padlock symbol.

The first MQGET is made and the first available record is read, the cursor set and the message is “marked”, shown later by the M symbol.

A MQGET with the same options is now made. The cursor position is ignored; the next available unmarked record is returned and marked, correctly skipping over the locked message that is part of some uncommitted unit of work.

Now, once more, some other task adds a high priority (nine) message to the queue.

A read with the same options is now made. The cursor position is ignored; the newly added record is returned and marked.

Now the Unit of work that had locked the two messages that have been skipped is rolled back, releasing the messages to be browsed.

A read with the same options is now made. The cursor position is ignored; the newly restored record is returned and marked.

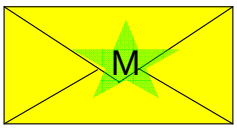
In this case it is clear that the first available (left most) available unreturned message is returned after every call.

This is the situation that you want; no messages have been omitted.

The next slide will show the options available.

## Browse with mark options

- MQGMO_UNMARKED_BROWSE_MSG
- MQGMO_MARK_BROWSE_HANDLE
- MQGMO_UNMARK_BROWSE_HANDLE




- Messages stay marked until
  - The object handle is closed
  - The message is unmarked for this handle by a call to MQGET using the option MQGMO_UNMARK_BROWSE_HANDLE
  - The message is returned from a call to destructive MQGET This is true even if the MQGET is subsequently rolled-back
  - The message expires




This slide shows the options that are used when an application is marking messages when marks are relevant to that application only.

The three options are all Get Message options that are supplied on the MQGET Call.

MQGMO_UNMARKED_BROWSE_MSG means that any marked messages are to be skipped over and not returned. If no unmarked messages are available then the reason code MQRC_NO_MSG_AVAILABLE is returned.

MQGMO_MARK_BROWSE_HANDLE is an instruction to MARK the returned message, this mark is visible to applications using this HANDLE only.

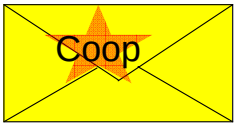
MQGMO_UNMARK_BROWSE_HANDLE is how a MARK can be removed programmatically.

This leads to the consideration of how long messages do indeed stay marked.

The answer is, until the handle to the queue used to mark them is closed, or the mark is explicitly removed, or the message is read destructively or until the message expires.

# Cooperative browse with mark options

- Open Option MQOO_CO_OP must be used.
- Intent to read only unmarked messages is still
  - MQGMO_UNMARKED_BROWSE_MSG
- Marking options are
  - MQGMO_MARK_BROWSE_CO_OP
  - MQGMO_UNMARK_BROWSE_CO_OP




- Closing, or termination of an application does not remove marks.
  - New Queue Manager attribute
    - **ALTER QMGR MARKINT(*integer* | NOLIMIT)**
  - Time out after which time if no application has destructively got the message it is returned to the unmarked pool for reprocessing.




This slide shows the options that are used when an application is marking messages and when the marks are to be shared by all cooperating applications.

In order to take part in cooperative marking an application must open the queue with the open option MQOO_CO_OP. All the applications that use this open option can then share a common set of marks. A message marked cooperatively by one is seen as marked by all.

On the MQGET call the options that set (or indeed unset) the mark must be changed as indicated to MQGMO_MARK_BROWSE_CO_OP.

Unlike the non cooperative case the closing or failure of an application does not cause the messages to be unmarked. This leads to a potential problem where an application marks a message as a signal of its intent to process the message but for some reason fails before successful processing can be carried out. This could lead to a marked message sitting on the queue and never processed.

A new queue manager attribute is introduced in MQ7 to assist in this situation. This is the Mark Interval, and can be set by the command

ALTER QMGR MARKINT (integer). What this specifies is a time interval (in milliseconds) after which a marked message should be unmarked if no application has processed the message by destructively reading it. So effectively the MARK operation has a timeout of this time and is automatically reset at the end.

## Message tokens in WebSphere MQ V7

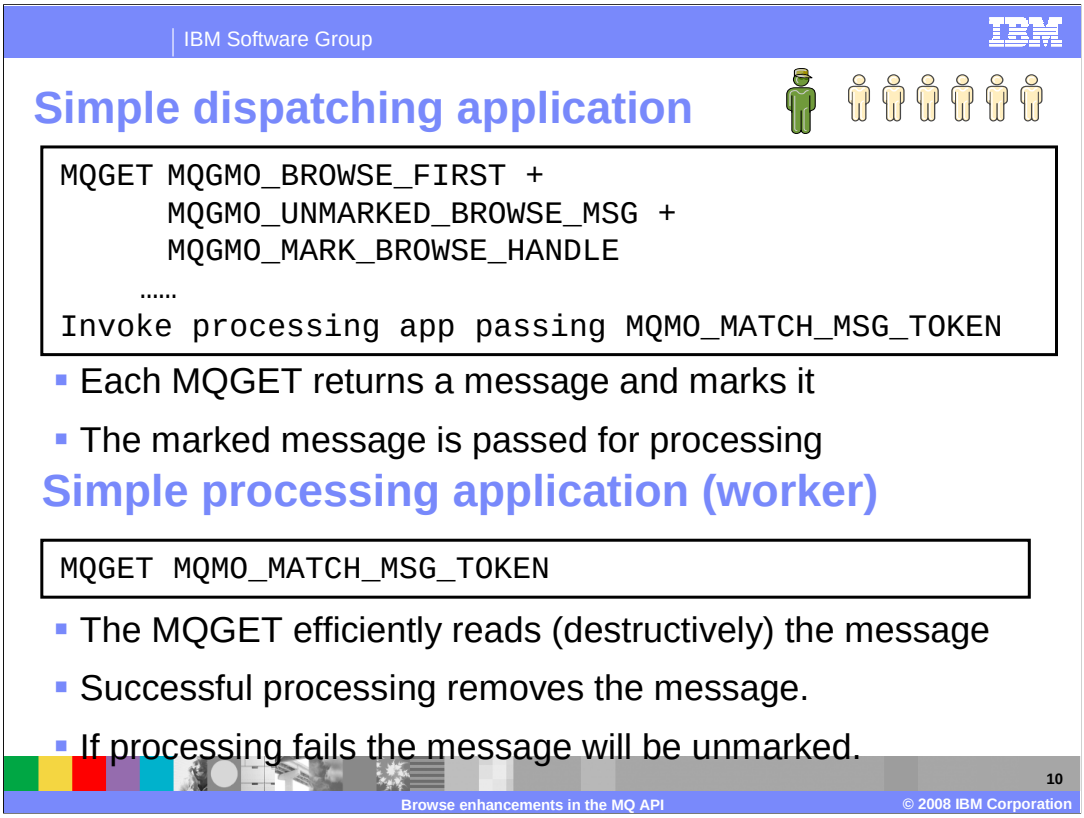
- Field in MQGMO (version 3)
  - Previously available only on z/OS®
  - 16 byte identifier, unique within that queue
- Can be used as a match option for MQGET
  - MQMO_MATCH_MSG_TOKEN
- Created by the queue manager when a message is placed on a queue
  - Does not follow the message from queue to queue
  - Does not survive a queue manager restart




This slide introduces another new topic the message token. The message token is a field in the MQ Get Message Options. It was previously only available on MQ for z/OS but is available on all platforms in version seven. It is a unique identifier (like a fingerprint) for a message that is assigned when the message is created (by MQPUT) and provides a very efficient mechanism for reading a particular message from a queue when used with the MQMO_MATCH_MSG_TOKEN option.

The message token is returned to the program when the message is put or got; in particular it is returned when a message is browsed. This makes it a useful way in which the identity of a message may be passed to another application for it to efficiently access a particular message. The value of the message token is specific to a particular message, on a particular queue and only for the lifetime of a queue manager start. That is to say that the tokens may change if the queue manager is restarted, or if a message is copied to another queue.

Message Token together with the new browse options provides the tools to build a simple and efficient dispatching or work queue processing applications.

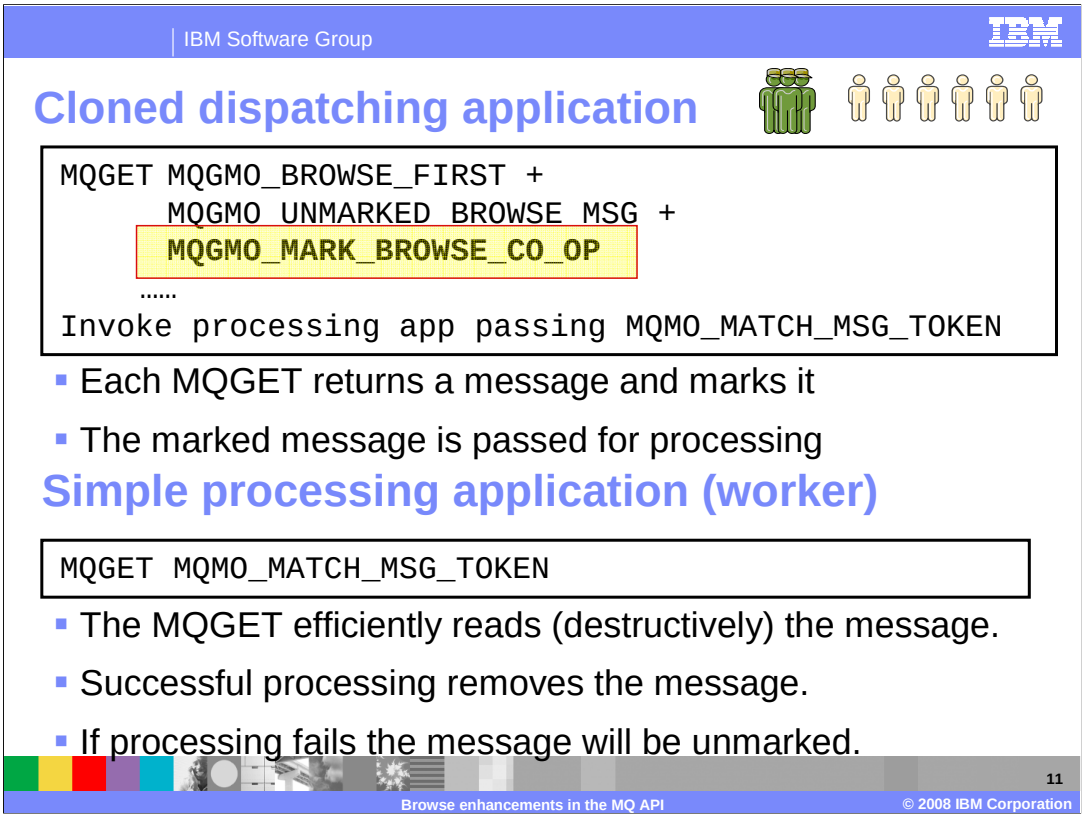



Here is a single dispatcher application using the new mechanism to browse the work queue. At every read it will take the highest priority or oldest message first. Because only a single instance of the dispatcher exists it uses non cooperative marking.

The dispatcher then examines the message and passes it to the appropriate worker task to be processed. It passes the message token in order to provide the worker task an efficient way of accessing the message.

The worker thread performs a destructive get and processes the message. Its normal operation will probably be to consume the message and terminate. The message will then be removed from the queue. Should the worker task fail and back out the message, the mark will be reset automatically and the message will be processed again by the dispatcher.

This simple model works fine, but multiple dispatchers may be needed. For example the dispatching process may not be able to run fast enough to keep up with the rate at which new work items are being added to the queue.




Here multiple identical dispatchers are used.

Because they are using cooperative marking the dispatcher applications need to be modified to open the queue with the cooperative option and to mark the messages with the cooperative option. This is the only required change.

Note that the processing application requires no change at all.

The case where one of the dispatchers fails after marking a message but before invoking the worker process is catered for by the Mark Interval Queue manager property which automatically deselects unread messages.

It will be clear that dispatchers can be added and removed from this configuration as the workload requires.

## Unit summary

Having completed this unit, you should be able to:

- Understand the problems with current message browsing with MQ
- Understand the concept of browsing a queue using message mark
- Use message tokens to pass references to already browsed messages
- Write application programs that can co-operatively browse a queue




In summary, this presentation has covered the problems associated with browsing queues in earlier releases of MQ, introduced the concepts and mechanisms used to browse and mark messages, shown how message token can be used to pass a reference to a message and how applications can cooperate to collectively browse a queue.

# 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_340_wmqv7_API_4_BrowseMark.ppt

This module is also available in PDF format at: ../iea_340_wmqv7_API_4_BrowseMark.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 z/OS

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.