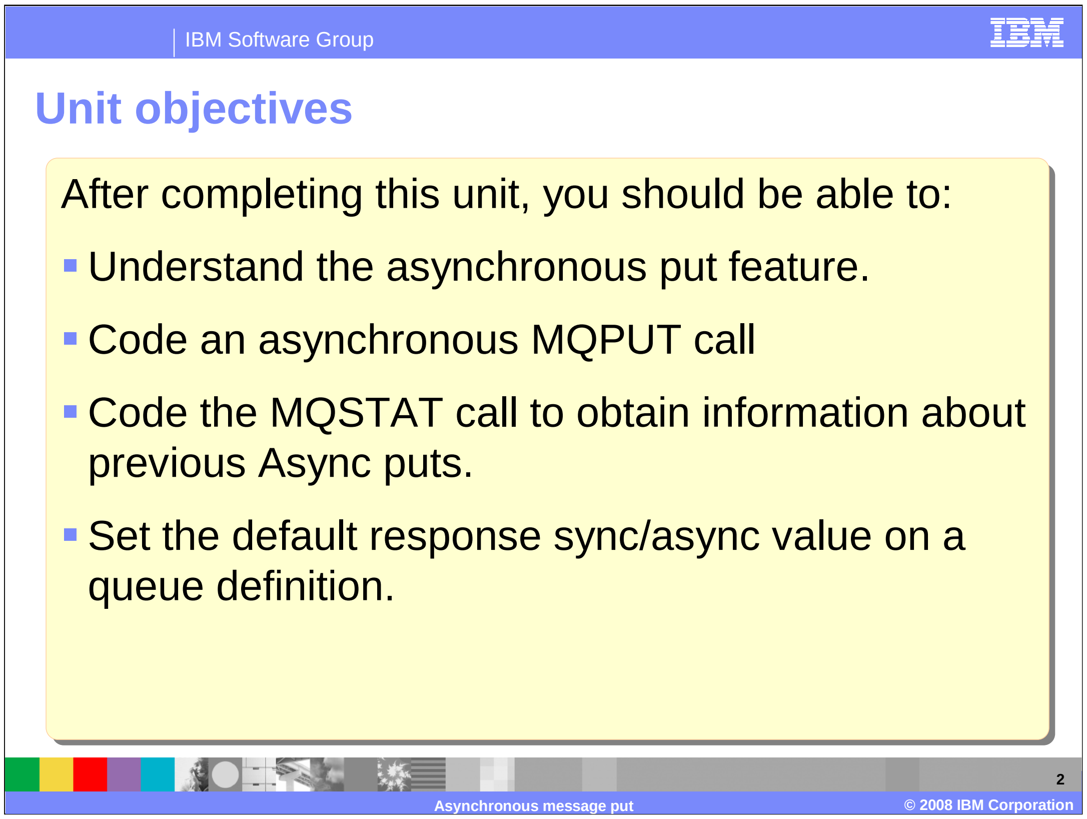
IBM Software Group

WebSphere® MQ V7.0

Client enhancements: Asynchronous message put

business on demand.

This unit covers the major enhancement available to TCP client connected applications putting messages to MQ. In some circumstances significant throughput improvement can be gained.



Unit objectives

After completing this unit, you should be able to:

- Understand the asynchronous put feature.
- Code an asynchronous MQPUT call
- Code the MQSTAT call to obtain information about previous Async puts.
- Set the default response sync/async value on a queue definition.

After you complete this unit, you should understand the Asynchronous PUT feature of MQ version 7.

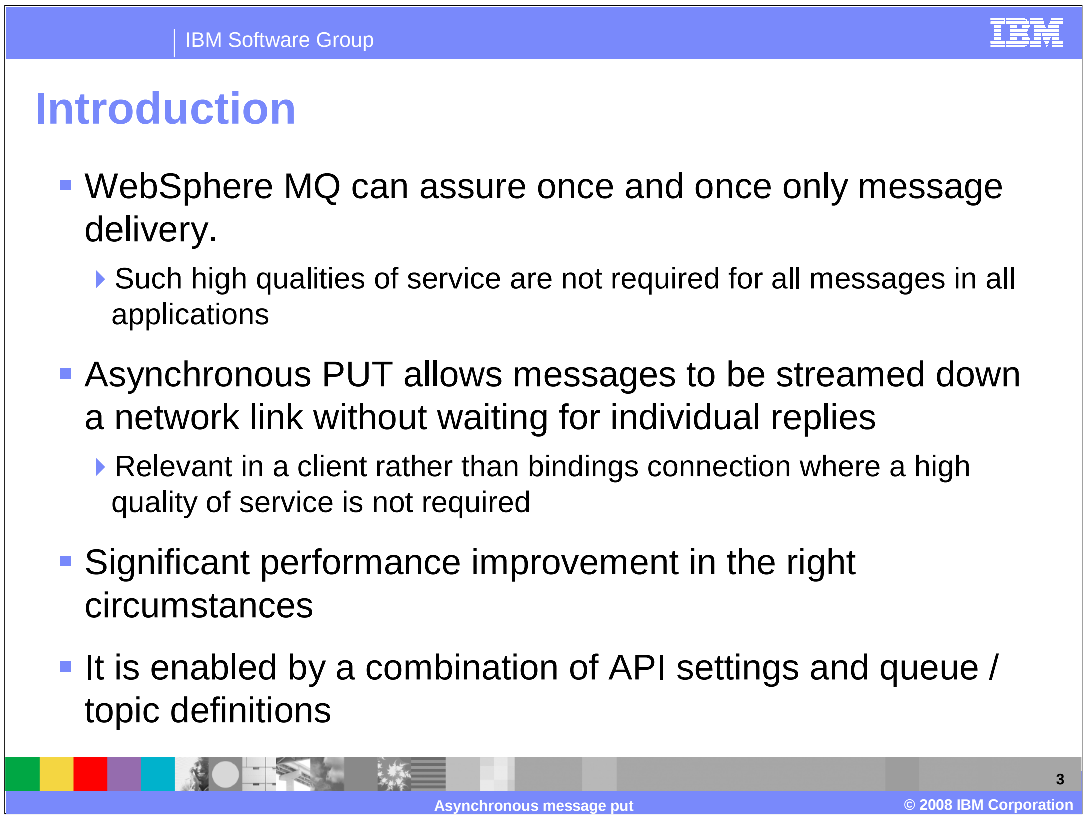
Understand the options required to code an MQPUT using the asynchronous put feature.

Use the new MQSTAT call to determine more information about failing asynchronous puts.

Use the administration interfaces to set the default asynchronous put setting for queues.

You will also understand the potential benefits in using these features.

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.



Introduction

- WebSphere MQ can assure once and once only message delivery.
  - Such high qualities of service are not required for all messages in all applications
- Asynchronous PUT allows messages to be streamed down a network link without waiting for individual replies
  - Relevant in a client rather than bindings connection where a high quality of service is not required
- Significant performance improvement in the right circumstances
- It is enabled by a combination of API settings and queue / topic definitions

One of the strengths of WebSphere MQ is, and remains, its ability to assure once and once only message delivery.

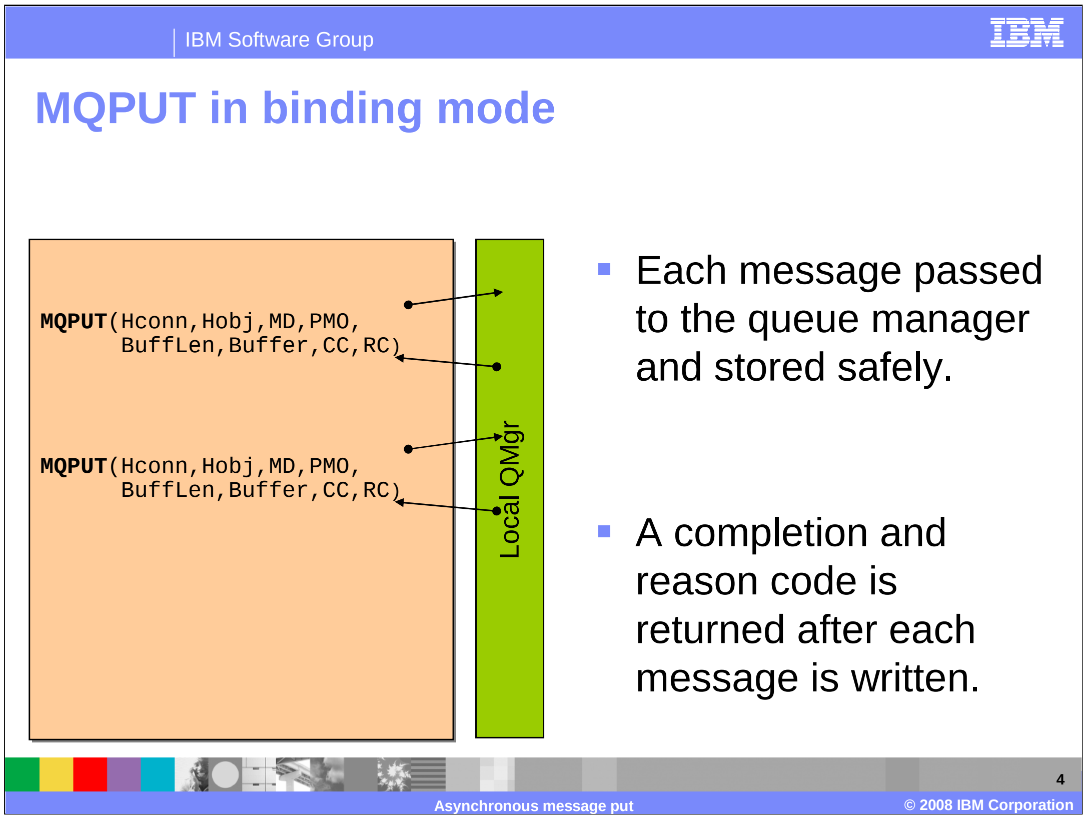
Such high qualities of service are not required for all messages in all applications.

Asynchronous response from PUT or “Fire and Forget” is a feature that allows messages to be streamed down a network link without waiting for individual replies.

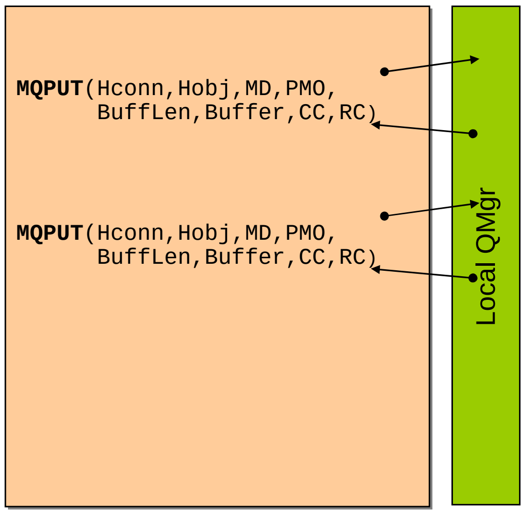
The feature is relevant in a client rather than bindings connection where a high quality of service is not required.

In the right circumstances significant performance improvement should be seen.

It is enabled by a combination of API settings and queue / topic definitions.



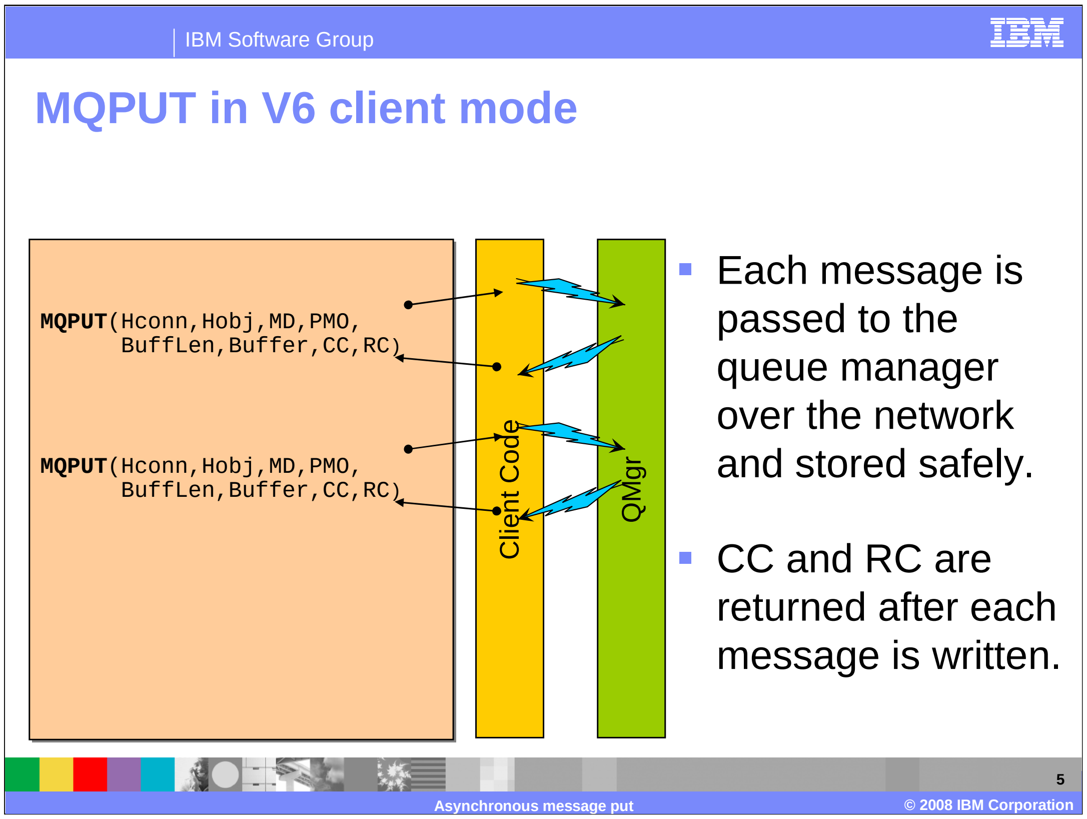
MQPUT in binding mode



- Each message passed to the queue manager and stored safely.
- A completion and reason code is returned after each message is written.

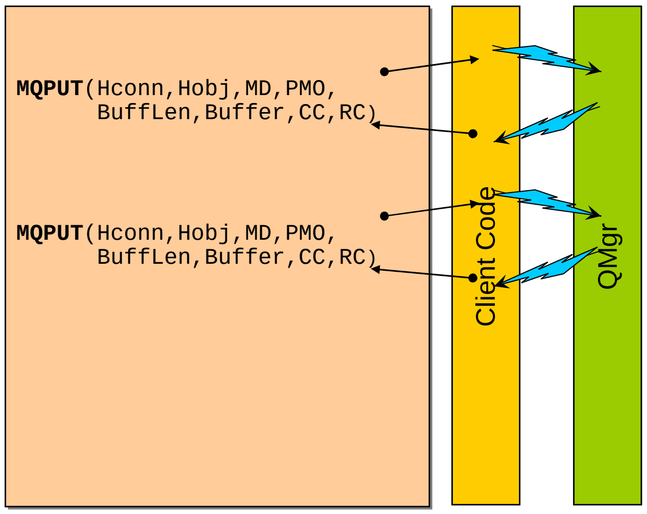
Consider a typical MQ application that is performing a sequence of MQPUT operations. If this application is connected to a queue manager on a local machine in “binding” mode then this slide illustrates what happens.

For each MQPUT operation: The application program begins the MQPUT Call; the request is passed to the queue manager which stores the message away safely on some persistent medium; at which point zero completion and reason codes and other data such as the messageID are passed back to the application.



IBM Software Group

MQPUT in V6 client mode



```
MQPUT(Hconn,Hobj,MD,PMO,
      BuffLen,Buffer,CC,RC)

MQPUT(Hconn,Hobj,MD,PMO,
      BuffLen,Buffer,CC,RC)
```

- Each message is passed to the queue manager over the network and stored safely.
- CC and RC are returned after each message is written.

5

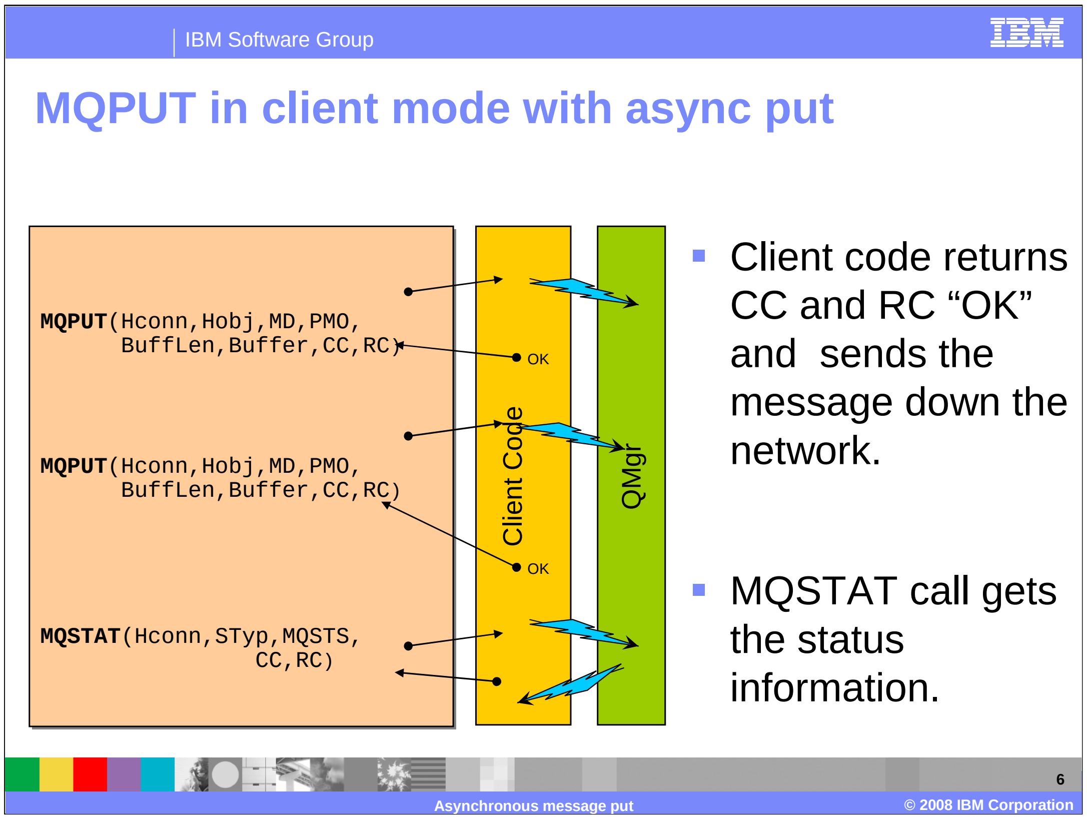
Asynchronous message put

© 2008 IBM Corporation

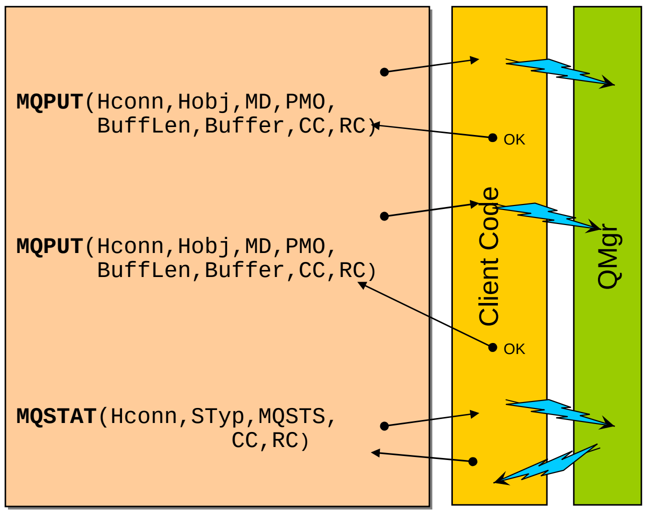
In a version six (or earlier) client application connected over a TCP communications link very much the same thing happens with additional steps.

For each MQPUT operations: The application program begins the MQPUT Call; the request is passed to the MQ client code; the MQ client code passes the request over the communication link using TCP/IP protocols to the queue manager which stores the message away safely on some persistent medium; at which point the zero completion and reason codes are passed backup the communication link to the MQ client code; the MQ client code then passes the call response to the application.

In this case the application code thread is held up blocked while the message is flowed to the queue manager, the message is stored and the response flowed back.



MQPUT in client mode with async put



- Client code returns CC and RC “OK” and sends the message down the network.
- MQSTAT call gets the status information.

In a version 7 client application, connected over a TCP communications link to a version 7 queue manager, something rather different can occur when the asynchronous put feature is in use.

For each MQPUT operations, the application program begins the MQPUT Call and the request is passed to the MQ client code. Now the MQ client code makes an optimistic assumption and responds to the application with a condition and reason code of zero. The application then proceeds while the client code delivers the message to the queue manager on another thread.

If the application issues another suitable MQPUT it too gets a rapid and optimistic condition code of zero and that message is delivered to the queue manager by the client code asynchronously.

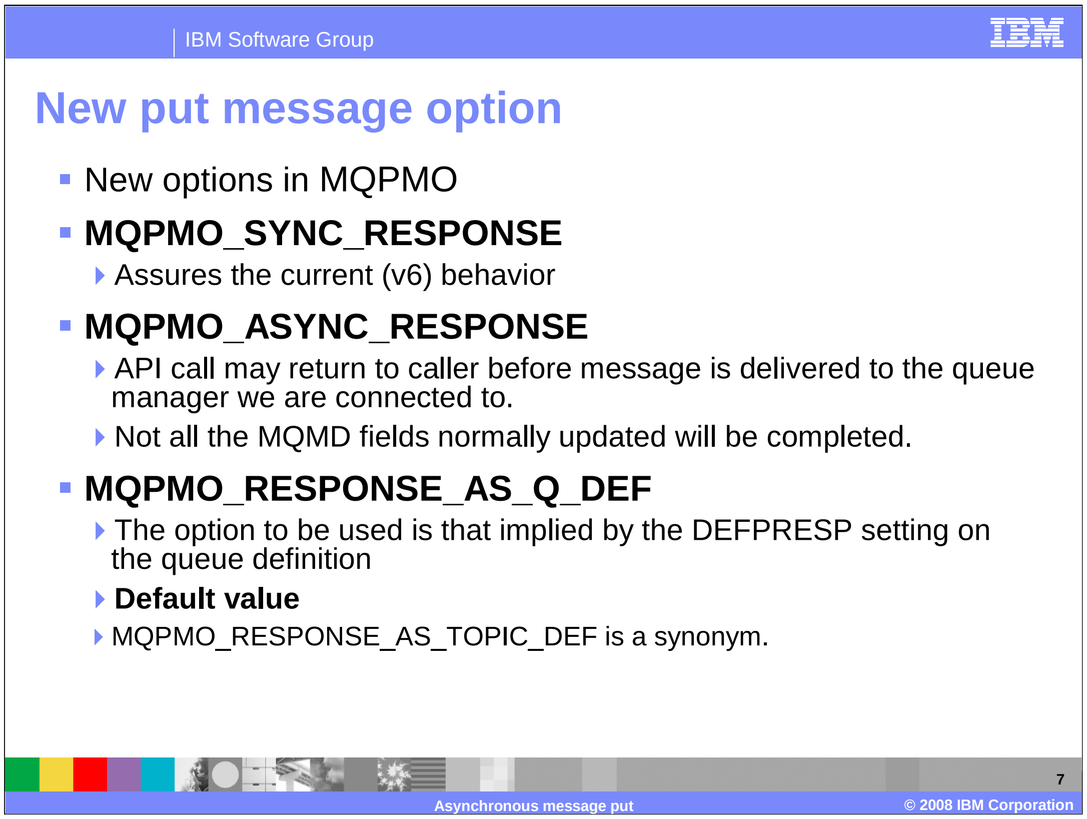
In this way the application is able to issue a sequence of MQPUT requests without having to wait for responses to be returned from the queue manager.

But what happens when all the messages have been sent?

If the messages were being put as part of a unit of work then a sync point will be taken. The sync point will only succeed if ALL the put messages have been successfully stored on the queue manager. So the effect of using this feature for MQPUTs within a sync point is to speed up the processing of successful units of work but to delay the discovery of a failure until the sync point, rather than the failing MQPUT. For many applications this will be a good trade, after all most applications are designed for the puts to work 99.99% of the time.

If the messages are being put outside a unit of work then if the application “cares” about the success of the MQPUTs special code must be written to find out if the requests succeeded.

This is the new MQSTAT call; a later slide shows its syntax.


IBM Software Group

New put message option

- New options in MQPMO
- **MQPMO_SYNC_RESPONSE**
  - Assures the current (v6) behavior
- **MQPMO_ASYNC_RESPONSE**
  - API call may return to caller before message is delivered to the queue manager we are connected to.
  - Not all the MQMD fields normally updated will be completed.
- **MQPMO_RESPONSE_AS_Q_DEF**
  - The option to be used is that implied by the DEFPRESP setting on the queue definition
  - **Default value**
  - MQPMO_RESPONSE_AS_TOPIC_DEF is a synonym.

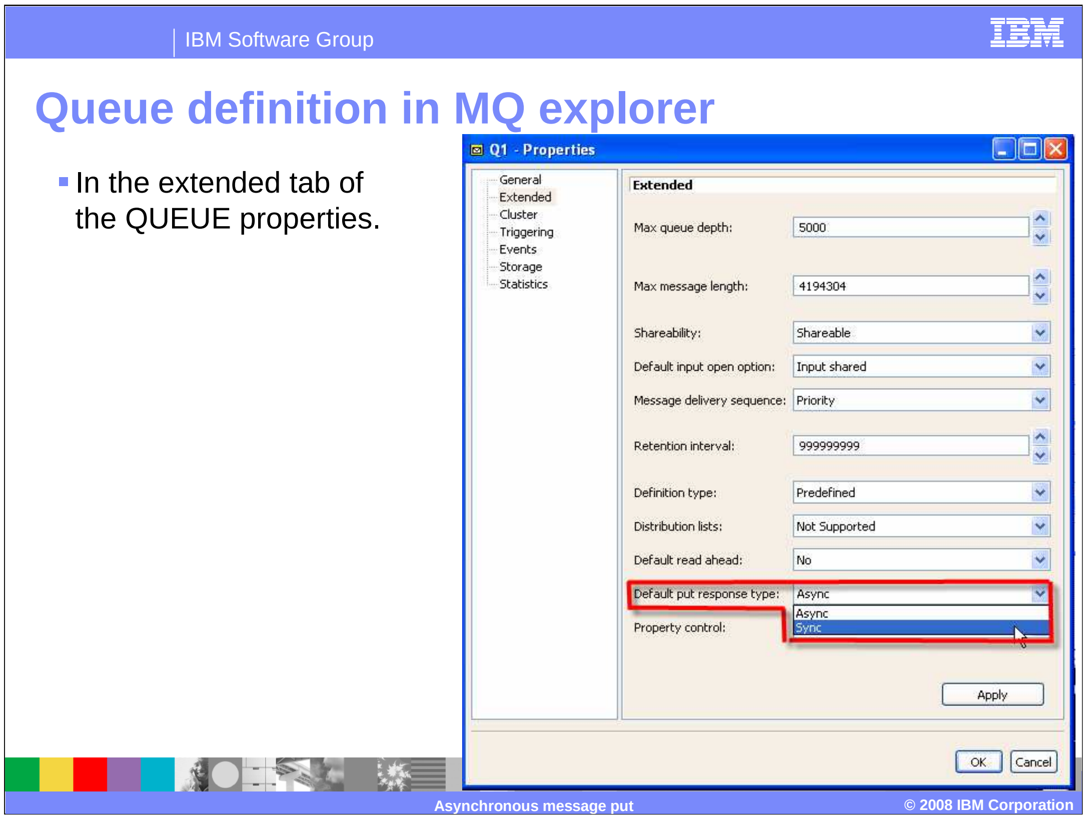
7

Asynchronous message put © 2008 IBM Corporation

Whether or not messages are being put with asynchronous responses is determined by a combination of new options on the MQPUT call and new attributes of a queue definition.

The new options are specified in the put message options (MQPMO) block of an MQPUT call. Possible values are MQPMO_SYNC_RESPONSE; this setting assures the version 6 behavior. It would be specified in an application that had to have immediate response from the put. This immediate response would be essential to the logic of the application. MQPMU_ASYNC_RESPONSE would be specified if this application does not require the synchronous response. The application must not depend on all the normally completed fields in the Message descriptor (MQMD) being completed.

The information center gives details of which fields are completed, these would normally include messageID. The default value, and the one used by all existing applications, is MQPMO_RESPONSE_AS_Q_DEF. This means that the decision is left the administrators of the system and the behavior will be that specified by the DEFPRESP attribute. It should be mentioned that asynchronous response from MQPUT can also apply when the MQPUTS are to a TOPIC not a QUEUE, which is when messages are being published to a topic rather that appended to a queue.


IBM Software Group

Queue definition in MQ explorer

- In the extended tab of the QUEUE properties.

Q1 - Properties

General
Extended
Cluster
Triggering
Events
Storage
Statistics

Extended

Max queue depth:	5000
Max message length:	4194304
Shareability:	Shareable
Default input open option:	Input shared
Message delivery sequence:	Priority
Retention interval:	999999999
Definition type:	Predefined
Distribution lists:	Not Supported
Default read ahead:	No
Default put response type:	Async
Property control:	

Async
Sync

Apply

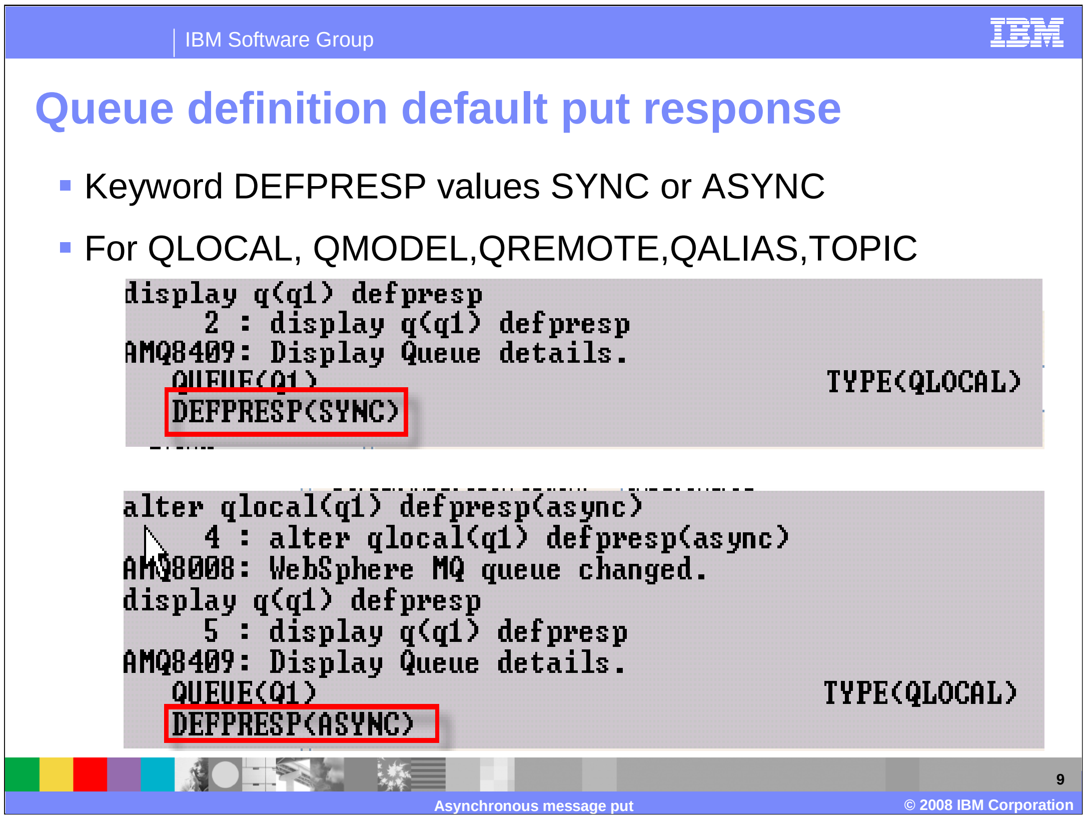
OK Cancel

Asynchronous message put

© 2008 IBM Corporation

MQ explorer or RUNMQSC (or equivalent PCF commands) can be used to define the default behavior of queues and topics.

In MQ explorer the “Default put response type” property can be found on the extended tab.



IBM Software Group

Queue definition default put response

- Keyword DEFPRESP values SYNC or ASYNC
- For QLOCAL, QMODEL,QREMOTE,QALIAS,TOPIC

```
display q(q1) defpresp
     2 : display q(q1) defpresp
AMQ8409: Display Queue details.
   QUEUE(Q1)                                    TYPE(QLOCAL)
   DEFPRESP(SYNC)
```

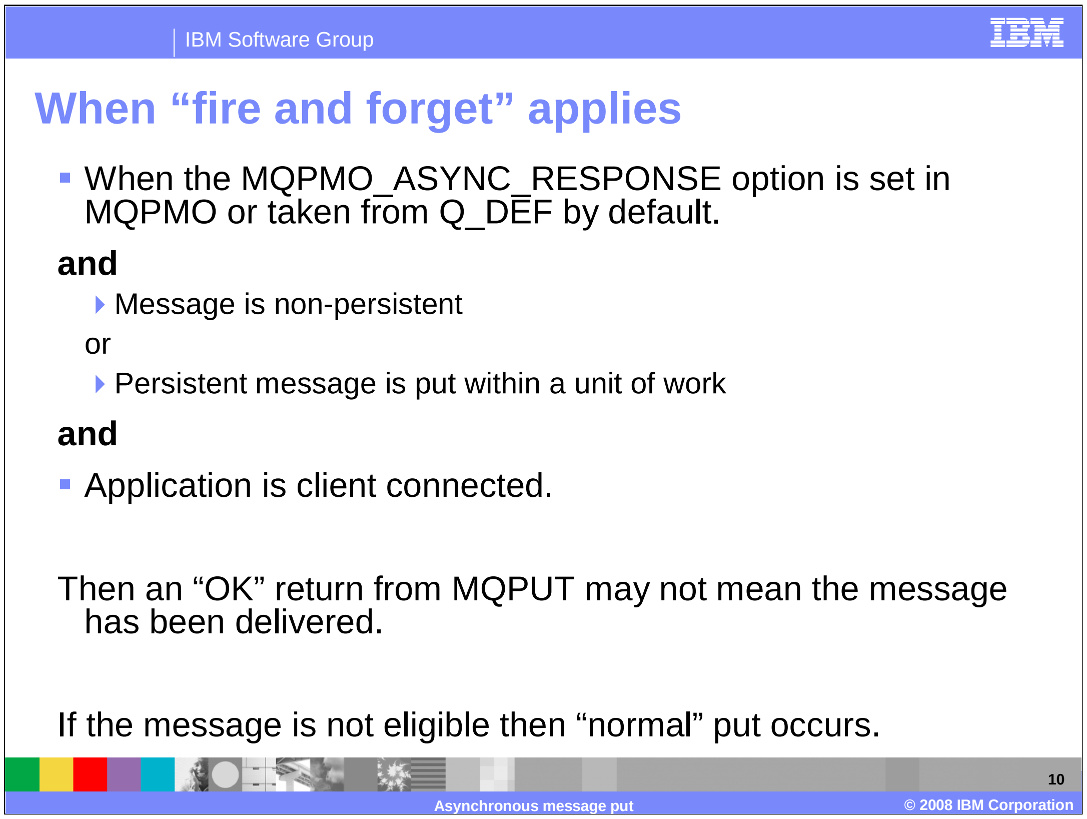
```
alter qlocal(q1) defpresp(async)
     4 : alter qlocal(q1) defpresp(async)
AMQ8008: WebSphere MQ queue changed.
display q(q1) defpresp
     5 : display q(q1) defpresp
AMQ8409: Display Queue details.
   QUEUE(Q1)                                    TYPE(QLOCAL)
   DEFPRESP(ASYNC)
```

9

Asynchronous message put © 2008 IBM Corporation

In RUNMQSC the keyword is “DEFPRESP” and takes values “SYNC” or “ASYNC”.

This can apply to QLOCAL, QMODEL, QREMOTE, QALIAS or TOPIC objects.



When “fire and forget” applies

- When the MQPMO_ASYNC_RESPONSE option is set in MQPMO or taken from Q_DEF by default.

and

- Message is non-persistent

or

- Persistent message is put within a unit of work

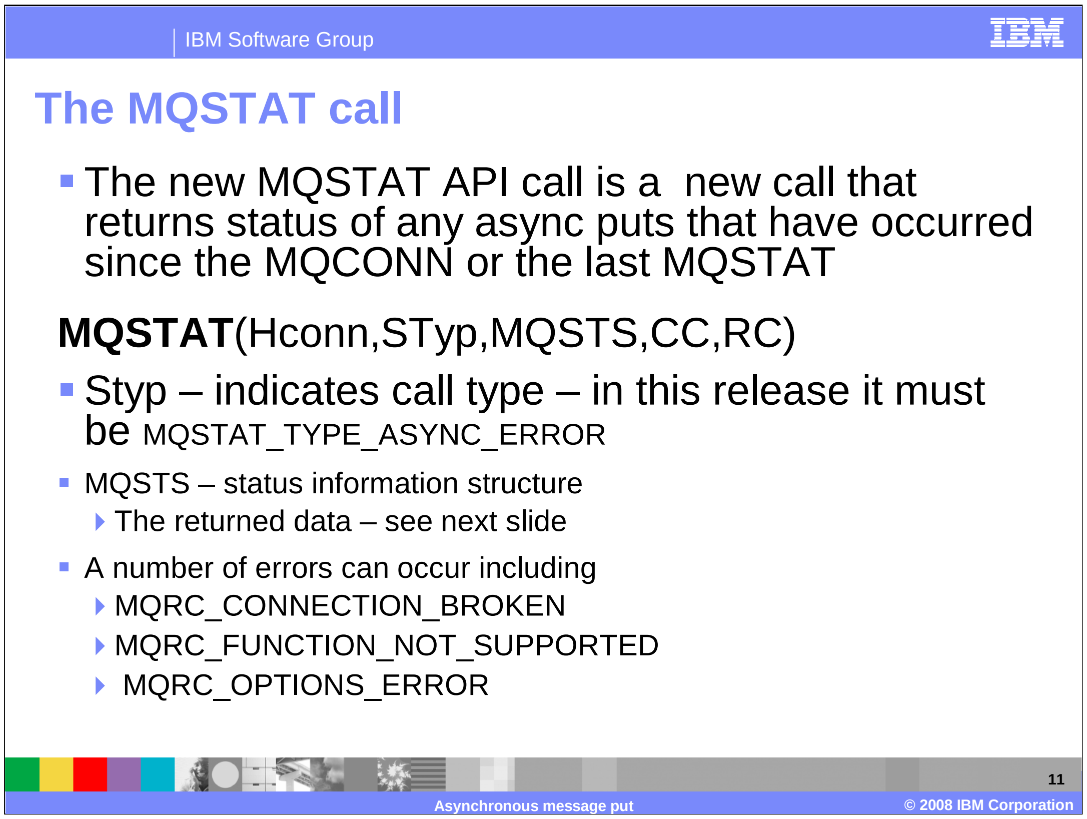
and

- Application is client connected.

Then an “OK” return from MQPUT may not mean the message has been delivered.

If the message is not eligible then “normal” put occurs.

To qualify for fire and forget put delivery, a message must meet a set of qualification criteria. First, the application MQPMO option field specifies MQPMO_ASYNC_RESPONSE or the application MQPMO options field specifies MQPMO_RESPONSE_AS_Q_DEF (or its synonym MQPMO_RESPONSE_AS_TOPIC_DEF.) The objects DEFPRESP attribute is set to ASYNC and the message is being put within a unit of work or the messages MQMD persistence is set to MQPER_NOT_PERSISTENT and the application is client connected.


IBM Software Group

The MQSTAT call

- The new MQSTAT API call is a new call that returns status of any async puts that have occurred since the MQCONN or the last MQSTAT

MQSTAT(Hconn,STyp,MQSTS,CC,RC)

- Styp – indicates call type – in this release it must be MQSTAT_TYPE_ASYNC_ERROR
- MQSTS – status information structure
  - The returned data – see next slide
- A number of errors can occur including
  - MQRC_CONNECTION_BROKEN
  - MQRC_FUNCTION_NOT_SUPPORTED
  - MQRC_OPTIONS_ERROR

11

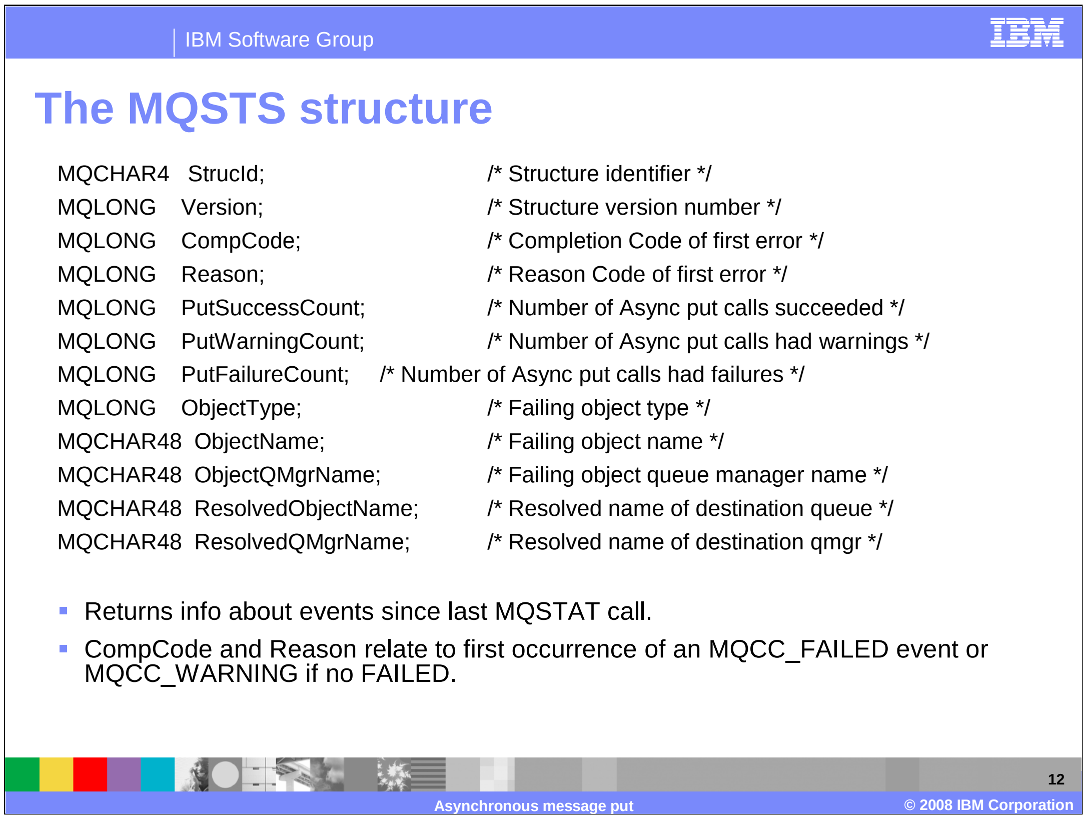
Asynchronous message put © 2008 IBM Corporation

One final topic is the new MQSTAT call.

This new call can be used to return information about asynchronous response puts.

In this release it supports only one call type “MQSTAT_TYPE_ASYNC_ERROR” the call returns a structure described on the next slide that gives some information about the results of all the asynchronous response MQPUT calls since the application began or the last MQSTAT call.

The MQSTAT call can itself fail for various reasons – one of which is that it is no longer possible to communicate with the queue manager to retrieve the required data.


IBM Software Group

The MQSTS structure

```
MQCHAR4   StrucId;                /* Structure identifier */
MQLONG    Version;                /* Structure version number */
MQLONG    CompCode;               /* Completion Code of first error */
MQLONG    Reason;                 /* Reason Code of first error */
MQLONG    PutSuccessCount;        /* Number of Async put calls succeeded */
MQLONG    PutWarningCount;        /* Number of Async put calls had warnings */
MQLONG    PutFailureCount;   /* Number of Async put calls had failures */
MQLONG    ObjectType;             /* Failing object type */
MQCHAR48  ObjectName;             /* Failing object name */
MQCHAR48  ObjectQMgrName;         /* Failing object queue manager name */
MQCHAR48  ResolvedObjectName;     /* Resolved name of destination queue */
MQCHAR48  ResolvedQMgrName;       /* Resolved name of destination qmgr */
```

- Returns info about events since last MQSTAT call.
- CompCode and Reason relate to first occurrence of an MQCC_FAILED event or MQCC_WARNING if no FAILED.

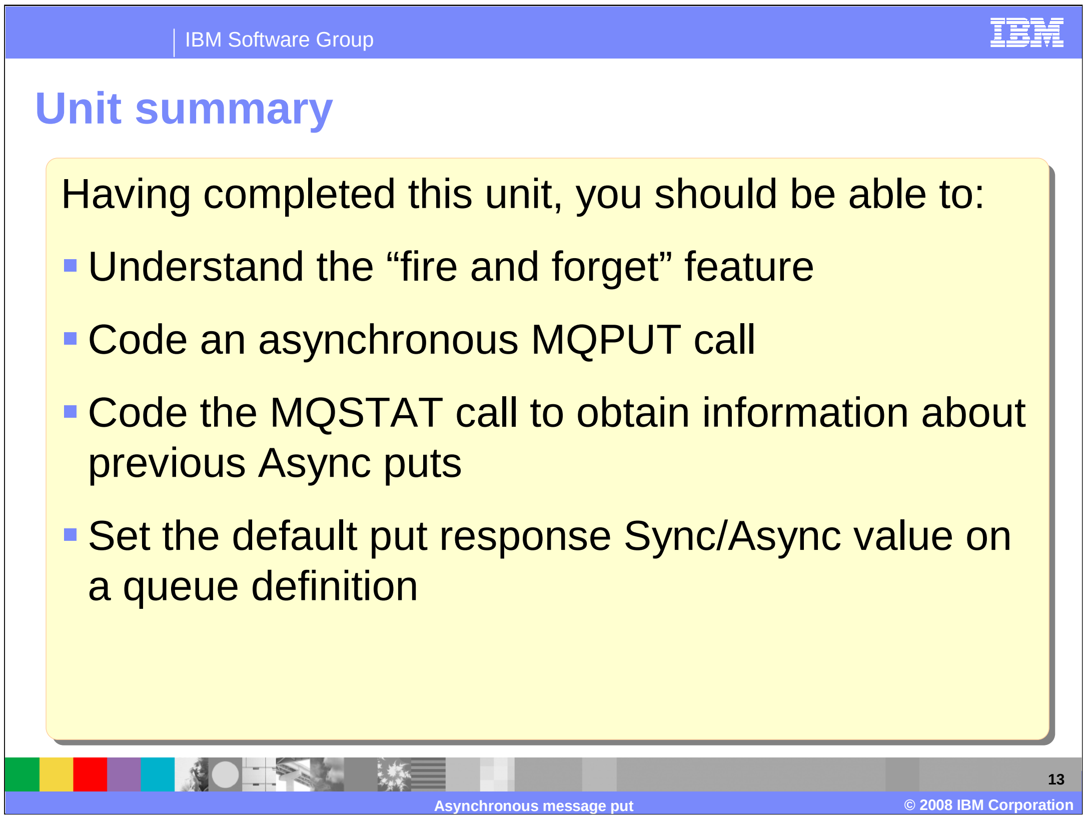
12

Asynchronous message put © 2008 IBM Corporation

This structure returns information about previous asynchronous response MQPUT and MQPUT1 operation.

The CompletionCode, ReasonCode, ObjetType, ObjectName, ObjectQMgrName, ResolvedObjectName and ResolvedQMgrName, reported to the first put to result in an MQCC_FAILED code. Unless no calls resulted in MQCC_FAILED, in which case the first call resulting MQCC_WARNING is reported. Unless of course all calls have succeeded in which case OK is reported and no values are returned in the failing fields.

The three count fields record the number of calls that succeeded, failed or resulted in warnings.


IBM Software Group

Unit summary

Having completed this unit, you should be able to:

- Understand the “fire and forget” feature
- Code an asynchronous MQPUT call
- Code the MQSTAT call to obtain information about previous Async puts
- Set the default put response Sync/Async value on a queue definition

13

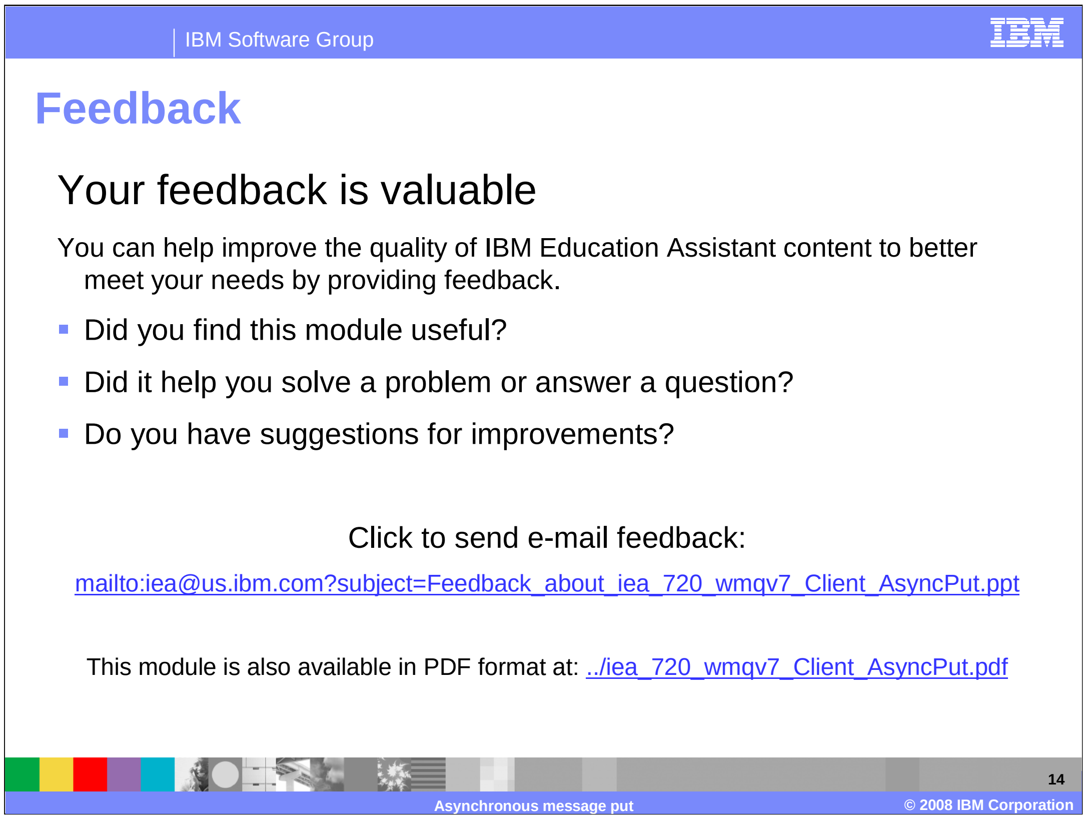
Asynchronous message put

© 2008 IBM Corporation

Having completed this unit, you should understand the Asynchronous PUT feature of MQ version 7 and how this can increase the throughput of some client connected applications.

You should understand the options required to code an MQPUT using the asynchronous put feature, use the new MQSTAT call to determine more information about failing asynchronous puts and use the administrative interfaces to set the default asynchronous put setting for queues.

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



Feedback

Your feedback is valuable

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

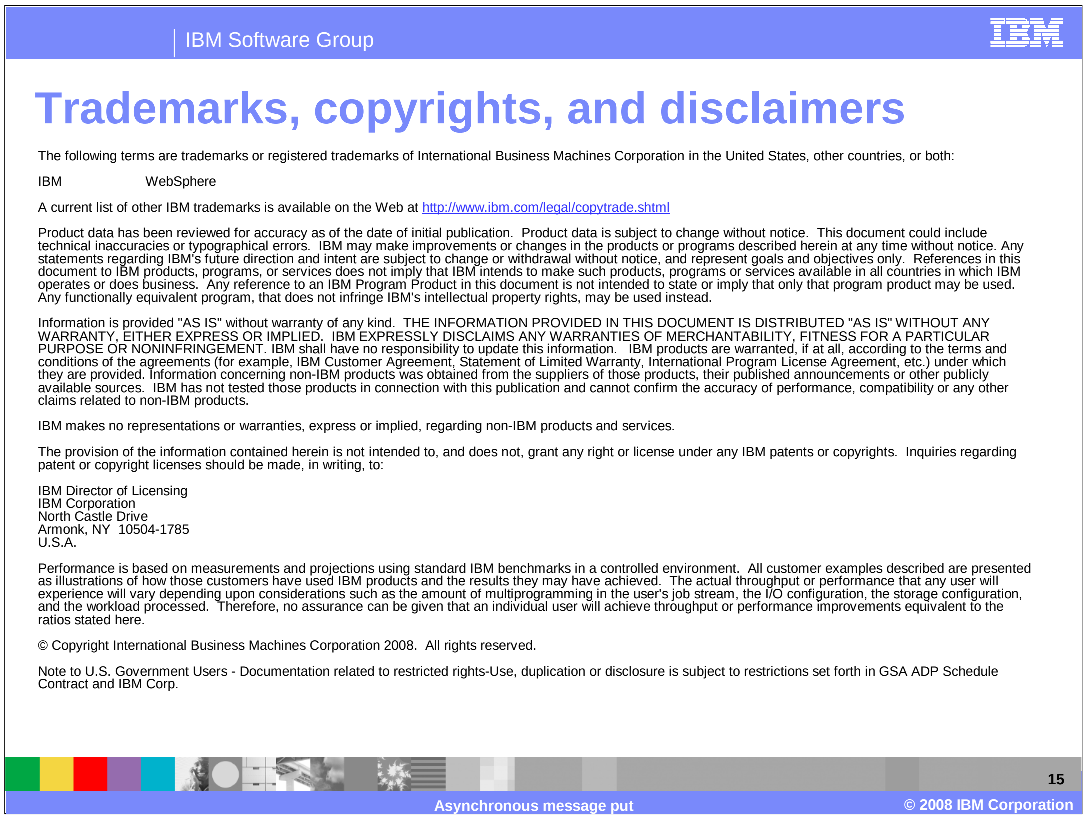
- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send e-mail feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_iea_720_wmqv7_Client_AsyncPut.ppt

This module is also available in PDF format at: ../iea_720_wmqv7_Client_AsyncPut.pdf

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



Trademarks, copyrights, and disclaimers