Learn to code the MQ Message Property MQI calls

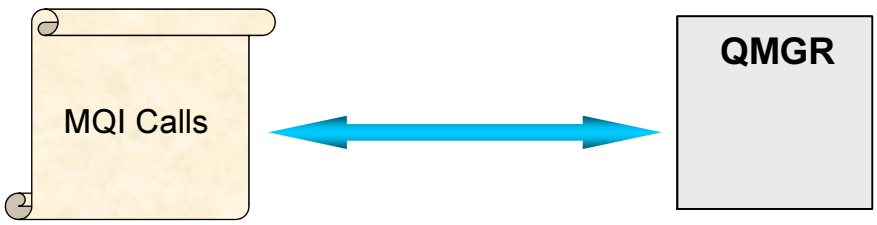
Morag Hughson – morag@mqgem.com

MQGem Software

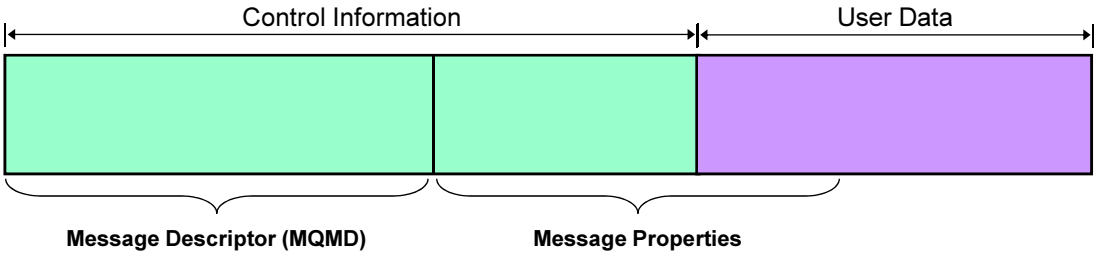
COBOL **C**

Agenda

- **Message Property MQI Concepts**
  - Message Handles
- **Basic MQI walkthrough**
  - With Demonstrations



Message Properties



- **Control information about a message**
  - MQMD fields – pre-defined
  - Message Properties – any value/type required
- **User data – the message body**
  - User-defined format – as today
  - Message Properties – any value/type required

Message Properties - Notes

N O T E S

- Message properties are a concept allowing meta-data or control information to be carried with a message without the need to put it either in a field in the MQMD or build it into the application user-data structure. This control information may be nothing to do with the application, such as tracking information – maybe inserted by an API exit or intermediate serving application – which the end application can ignore, or may be pertinent information that the application uses, perhaps to select messages by.
- Either way, properties are neither part of the user data, nor part of the MQMD. They are carried with the message and can be manipulated by means of a number of new API calls.

Message Handle

- Represents the message
- Retrieved on MQGET
- Can be provided on MQPUT
  - MQPMO.Action
    - MQACTP_NEW
    - MQACTP_FORWARD
    - MQACTP_REPLY
    - MQACTP_REPORT
  - Represents the relationship between two messages
- Create using MQCRTMH
- Delete using MQDLTMH

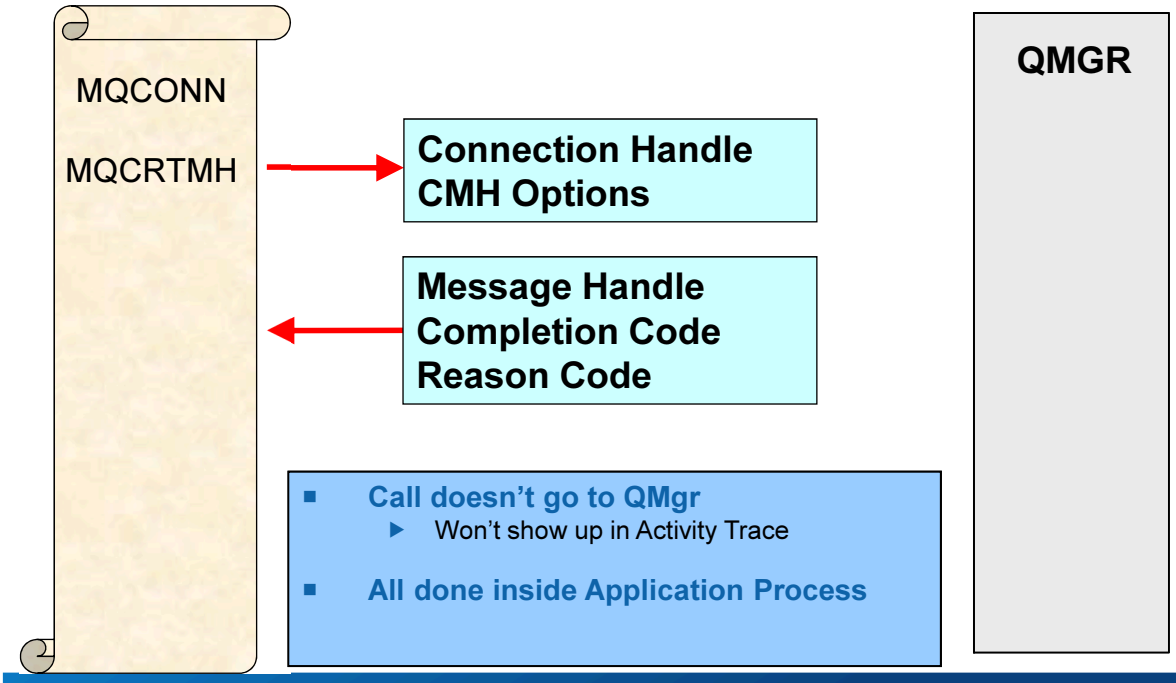
```
MQCRTMH(hConn,
        &cmho,
        &hMsg,
        &CompCode,
        &Reason);

gmo.MsgHandle = hMsg;
MQGET(hConn,
      ....);

pmo.Action = MQACTP_REPLY;
pmo.OriginalMsgHandle = hMsg;
MQPUT(hConn,
      ....);
```


Message Handle - Notes

N O T E S

- Message properties are manipulated via a message handle. When putting or getting a message, a message handle can be associated with the message in order to allow access to the message properties associated with the message.
- This message handle is a handy mechanism to represent a message and additionally allows the ability to tie messages together between MQGET and MQPUT. Without it, there is no way to tell whether the message that was just sent with MQPUT bears any relation to the message previously retrieved with MQGET. There is probably a high likelihood that it is, request/reply being a common model, but no certainty.
- If the message handle representing the message retrieved using MQGET is passed into a subsequent MQPUT, with an Action that says MQACTP_REPLY, it is now absolutely clear what the relationship is between these two messages and any message properties that are important for a reply type relationship can be automatically copied over.
- Before using a message handle, say on an MQGET, you must first create it using the MQCRTMH verb. When you are finished using a message handle, you should delete it using the MQDLTMH verb.

Create Message Handle



Create Msg Handle

- MQHMSG is *NOT* the same as MQHOBJ
  - 64-bit number
  - Do not use types interchangeably
- MQCMHO options
  - Validation
- hConn can be unassociated
  - For passing message properties between queue managers

```
MQCRTMH( hConn,
         &cmho,
         &hMsg,
         &CompCode,
         &Reason);

if (CompCode == MQCC_FAILED)
{
  /* Do some error processing */
}
```

```
MQCMHO    cmho = {MQCMHO_DEFAULT};
MQHMSG    hMsg = MQHM_UNUSABLE_HOBJ;

cmho.Options |= MQCMHO_VALIDATE;
```

MQCRTMH Tips

- **If you need to pass a message between queue managers (hConns) use MQHC_UNASSOCIATED_HCONN**
  - You must have a valid connection to a queue manager created on the same thread prior to doing this.
  - If not, KC says the call will fail with MQRC_HCONN_ERROR
  - By inspection, call fails with MQRC_CONNECTION_BROKEN

MQCRTMH Tips – Notes

N

O

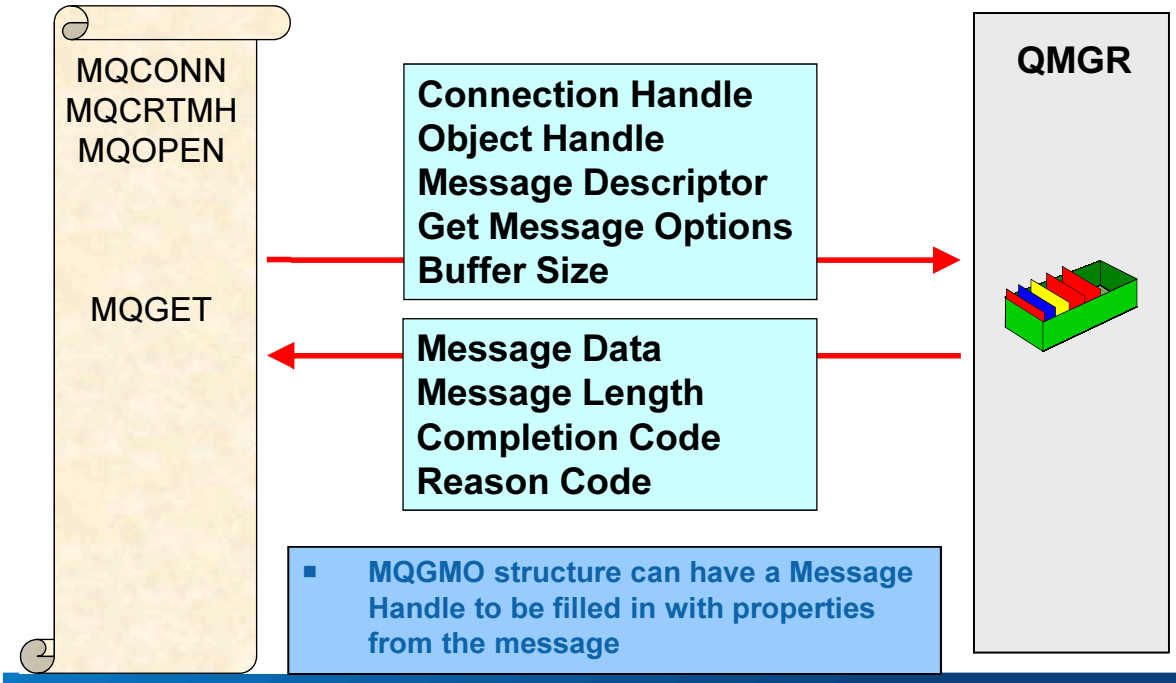
T

E

S

- For the majority of your applications, the message handle that you need to create will be for use with one connection. You will make an MQCONN(X) call and be return a connection handle. You will then use that connection handle to make the MQCRTMH call.
- For occasional niche applications, you may need to use a message handle when retrieving a message on one connection, and then put a message referring to that same message handle onto another connection. In these cases, which are not likely to happen as frequently, you can make the MQCRTMH call using a constant for the unassociated connection handle.
- If you do this, you must still have made a connection to something prior to calling MQCRTMH (to load MQI entry points into the process), and you must delete the message handle with MQDLTMH yourself, MQDISC will not tidy this up for you.

MQGET using Message Handle



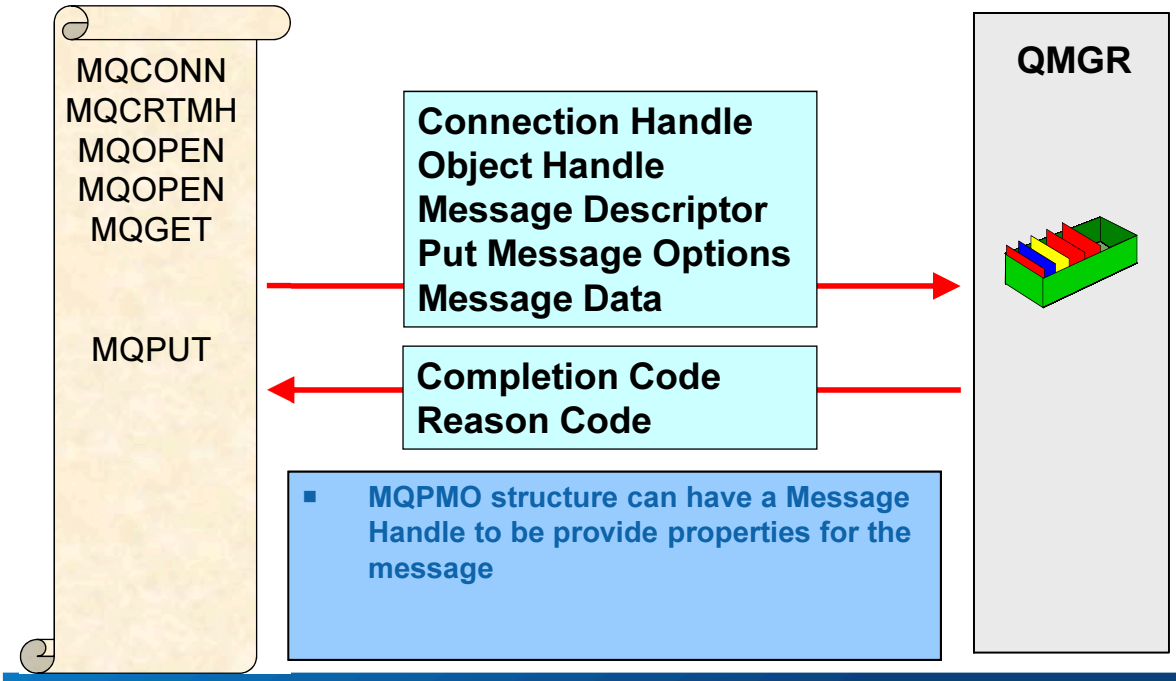
Get with Msg Handle

- MQCRTMH to create handle
- MQGET a message
  - Providing handle to be populated
- MsgHandle field
  - Value returned from MQCRTMH
  - Version 4 of MQGMO

```
MQCRTMH( hConn,
         &cmho,
         &hMsg,
         &CompCode,
         &Reason);

MQGET  ( hConn,
         hObj,
         &md,
         &gmo,
         sizeof(msg),
         msg,
         &msglen,
         &CompCode,
         &Reason);
```

```
MQGMO  gmo     = {MQGMO_DEFAULT};
gmo.Options    = MQGMO_SYNCPOINT_IF_PERSISTENT |
                 MQGMO_PROPERTIES_IN_HANDLE    |
                 MQGMO_FAIL_IF_QUIESCING       |
                 MQGMO_CONVERT;
gmo.Version    = MQGMO_VERSION_4;
gmo.MsgHandle = hMsg;
```

MQPUT using Message Handle



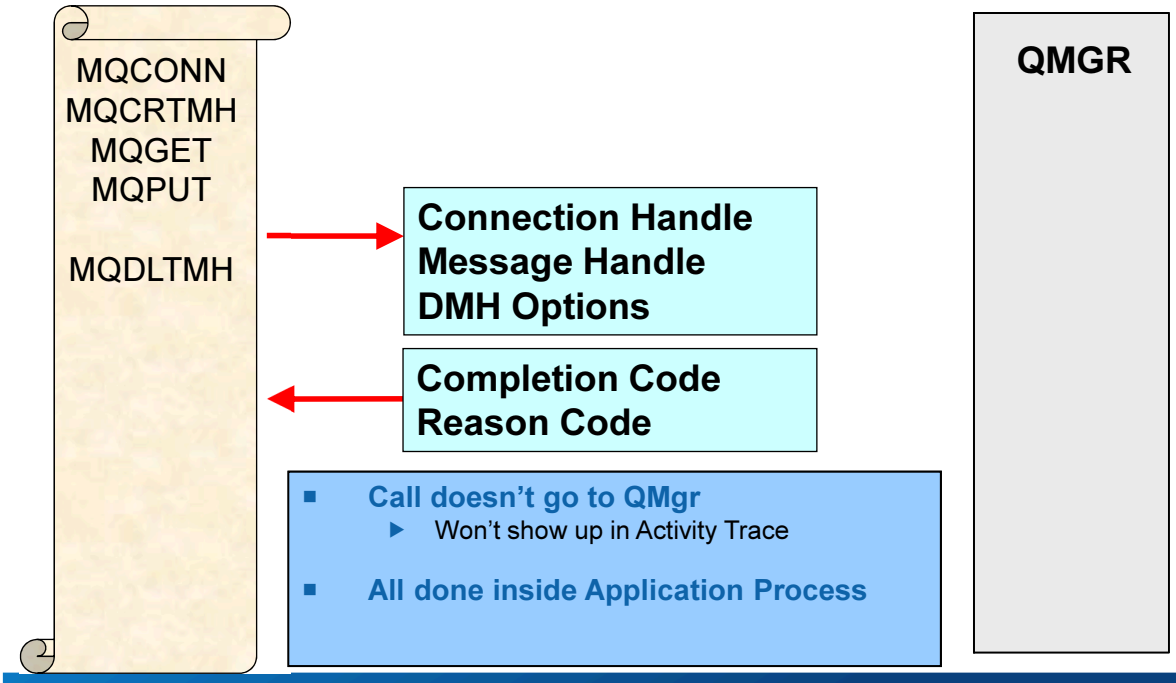
Put with Msg Handle

- MQPUT a message
  - Passing msg handle from Get
  - Indicating action
    - NEW
    - FORWARD
    - REPLY
    - REPORT
- MsgHandle fields
  - Version 3 of MQPMO
  - Value returned from MQGET or MQCRTMH

```
MQPUT ( hConn,
        hObj,
        &md,
        &pmo,
        strlen(msg),
        msg,
        &CompCode,
        &Reason);
```

```
MQPMO   pmo      = {MQPMO_DEFAULT};

pmo.Version             = MQPMO_VERSION_3;
pmo.OriginalMsgHandle = gmo.MsgHandle;
pmo.NewMsgHandle        = MQHM_NONE;
pmo.Action              = MQACTP_REPLY
```

Delete Message Handle



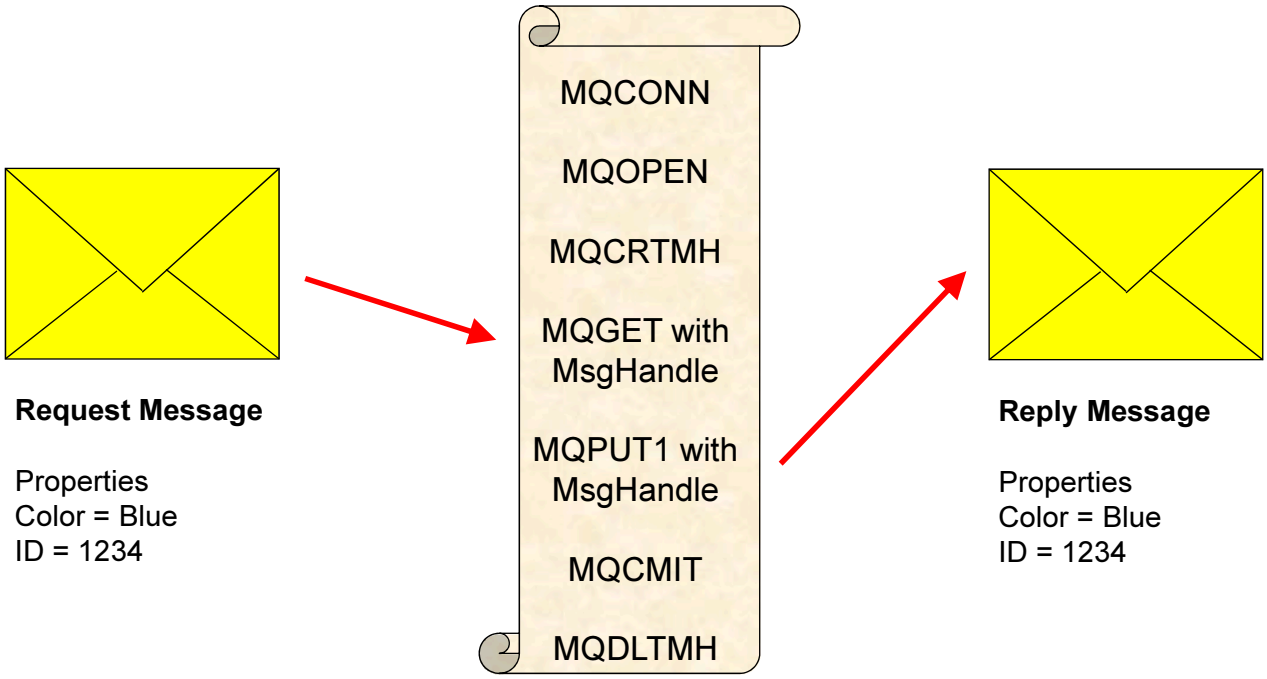
Delete Msg Handle

- Nothing interesting in Delete Message Handle Options
  - MQDMHO_NONE
  - Future proofed
- If Message Handle was created using an unassociated hConn it MUST be explicitly deleted

```
MQDLTMH( hConn,
         &hMsg,
         &dmho,
         &CompCode,
         &Reason);
```

```
MQDMHO    dmho = {MQDMHO_DEFAULT};
```

Demonstration



Demonstration – Notes

N O T E S

- In our first demonstration we will show an application that does not interpret, read, alter or in any way look at message properties of a message, be able to pass on the message properties on the message to the reply that it creates with ease.
- We start with a message with several message properties on it. We will use a simple browse sample to display what is there.
  `amqsbcg Q1 QM1 1`
- Now, using the API Exerciser to demonstrate, we will MQCONN, MQOPEN a queue for INPUT, MQCRTMH, passing the Message Handle into MQGET we retrieve the message. Looking in the buffer of the message retrieved we see no sign of Message Properties or MQRFH2 headers. We can process the message as normal without being affected by the message properties.
- Now we MQPUT1 the reply message, passing the Message Handle into the MQPUT1 and indicating that this is a MQACTP_REPLY message. The MQPUT1 and MQGET were done in syncpoint, so finally we'll MQCMIT.
- Now we end the demonstration by using a simple browse sample to display the reply message and see that the various message properties from the request message have been carried over to the reply.

Message Properties

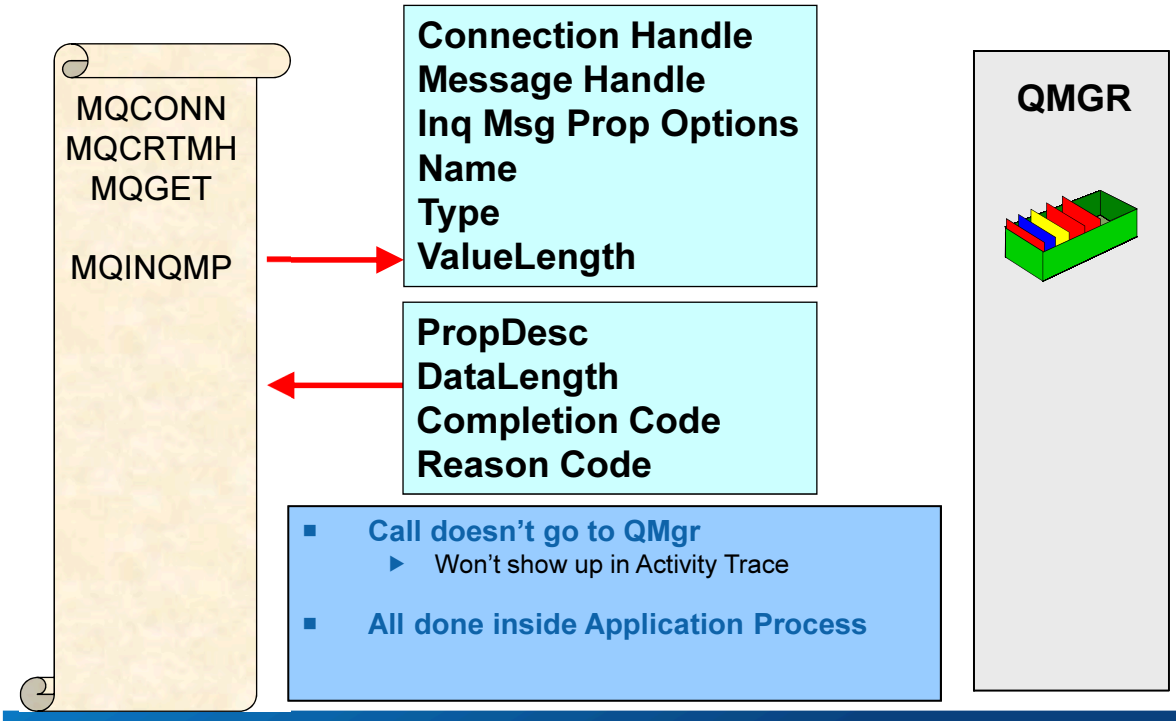
- **Verbs to manipulate**
  - MQSETMP
  - MQINQMP
  - MQDLTMP
    - All take a message handle
- **Property types**
  - MQTYPE_BOOLEAN
  - MQTYPE_BYTE_STRING
  - MQTYPE_INT8 / 16 / 32 / 64
  - MQTYPE_FLOAT32 / 64
  - MQTYPE_STRING
  - MQTYPE_NULL
- **Compatibility with MQRFH2**
  - Pre-V7 JMS User properties
  - API exits, MQMHBUF, MQBUFMH
  - Apps
    - MQGMO_PROPERTIES_FORCE_RFH2
    - Queue attribute

Message Properties - Notes

N O T E S

- Having retrieved your message handle, you can then use it to manipulate the message properties associated with the message.
- You can set a message properties on a message using the MQSETMP verb, and inquire it using the MQINQMP verb. If you need to remove a message property from a message handle, there is an MQDLTMP verb.
- When setting a message property, you must provide its name, value and type. The types are shown on the page. When inquiring a message property you are given its type on return, or you can request it is converted into another type if required. When deleting a message property you simply provide the property name.
- Additionally there are two other message property related API calls, MQMHBUF, and MQBUFMH. These will convert the message properties related to the message into an MQRFH2 header. These calls may be useful in an API exit that was previously written to manipulate MQRFH2s – perhaps for JMS User properties in a prior release. Any applications that require an MQRFH2 for JMS User properties (as in previous releases) can request this with the option MQGMO_PROPERTIES_FORCE_MQRFH2 – or control it by means of an attribute on the queue being used.

Inquire Message Property



- Call doesn't go to QMgr
  - Won't show up in Activity Trace
- All done inside Application Process

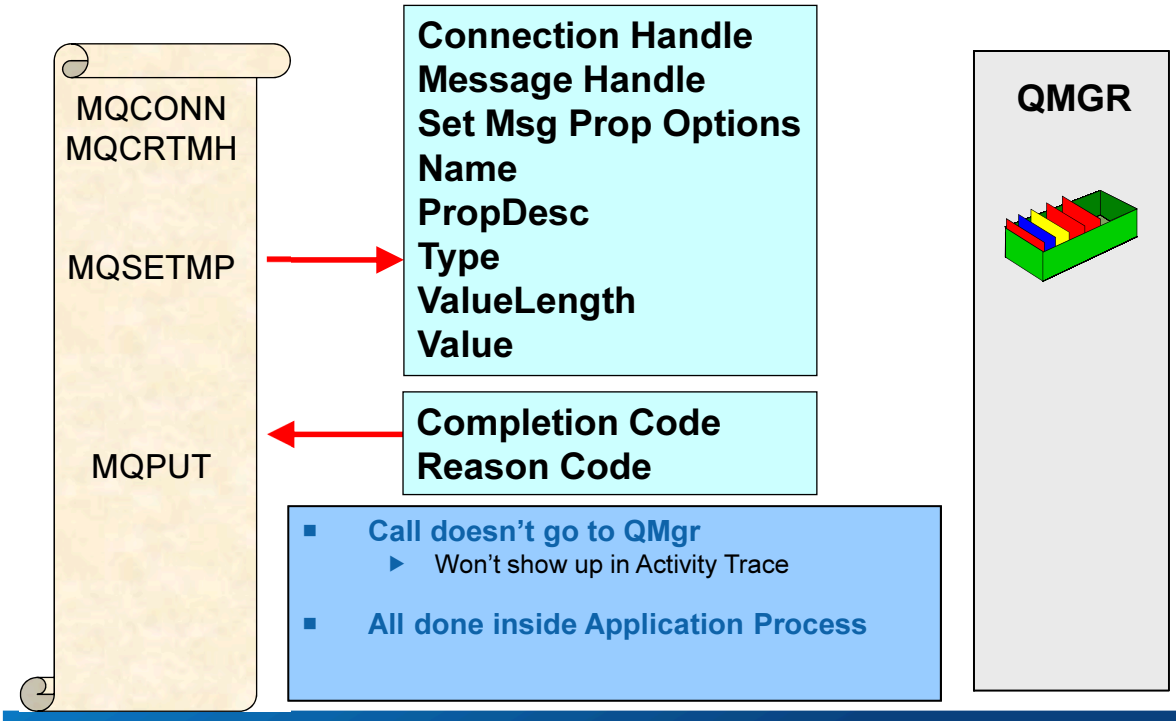
Inquire Msg Property

- Two constants are supplied for the Name parameter ('C' only)
  - MQPROP_INQUIRE_ALL => %
  - MQPROP_INQUIRE_USR => usr.%
  - Or you can build your own MQCHARV string
- If inquiring all properties, provide buffer for property name to be output
- Rather like browsing a queue
  - MQIMPO_INQ_FIRST and MQIMPO_INQ_NEXT
  - Provide a buffer and its length for the return -ed Value and get a data length on output

```
impo.Options = MQIMPO_INQ_NEXT;
impo.ReturnedName.VSPtr  = RetName;
impo.ReturnedName.VSBufSize
                = sizeof(RetName) -1;

MQINQMP( hConn,
         hMsg,
        &impo,
        &Name,
        &PropDesc,
        &Type,
        sizeof(Value),
        Value,
        &DataLength,
        &CompCode,
        &Reason);
```

```
MQIMPO  impo     = {MQIMPO_DEFAULT};
MQPD    PropDesc = {MQPD_DEFAULT};
MQCHARV Name     = {MQPROP_INQUIRE_ALL};
```

Set Message Property



- Call doesn't go to QMgr
  - Won't show up in Activity Trace
- All done inside Application Process

Set Msg Property

- MQSMPO Options
  - Related to the position of the message property with relation to others already present
- Property Descriptor
  - Set to indicate when properties should be copied over to new messages

```
Type           = MQTYPE_STRING;
Name.VSPtr     = "Color";
Name.VSLength  = strlen(Name.VSPtr);
Value          = "Blue";
ValueLen       = strlen(Value);

MQSETMP( hConn,
         hMsg,
        &smpo,
        &Name,
        &PropDesc,
         Type,
         ValueLen,
         Value,
        &DataLength,
        &CompCode,
        &Reason);
```

```
MQSMPO   smpo     = {MQSMPO_DEFAULT};
MQPD     PropDesc = {MQPD_DEFAULT};
MQCHARV  Name     = {MQCHARV_DEFAULT};
```

Property Descriptor

- **Controls when message properties are copied from one message to another.**
- **Context**
  - MQPD_USER_CONTEXT
  - Property will be copied when MQOO_SAVE_ALL_CONTEXT and MQPMO_PASS_ALL_CONTEXT is used
- **CopyOptions**

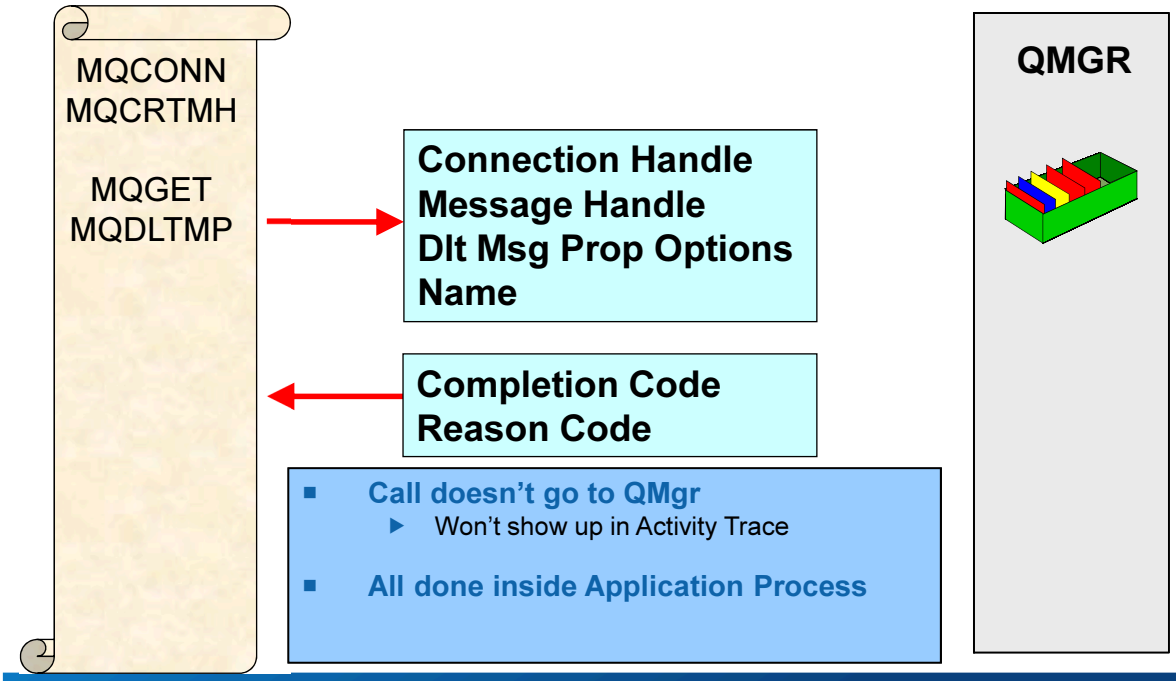
MQPMO.Action	CopyOption	MQCOPY_DEFAULT
MQACTP_NEW	N/A	
MQACTP_FORWARD	MQCOPY_FORWARD	✓
MQACTP_REPLY	MQCOPY_REPLY	
MQACTP_REPORT	MQCOPY_REPORT	✓
	MQCOPY_PUBLISH	✓

Property Descriptor – Notes

N O T E S

- The property descriptor (MQPD) provides some additional information about the property being set or inquired beyond it's Name, Type and Value which are what you have seen so far.
- These various fields describe how and when the property should be copied (the demonstration you saw earlier).
- Two of the fields in the MQPD control two different ways in which a property might be copied from one message to another.
- **Context**
- The context field in the MQPD can indicate that this property is to be treated as part of the context fields of the message, and copied across from one message to another when Save All Context and Pass All Context are used.
- **CopyOptions**
- The copy options indicate whether this property will be copied over depending on the Action indicated by the application.
- DEFAULT does not include REPLY, so for our earlier demonstration, the propertied were explicitly set to be copied on a reply message.
- There is also an ALL and NONE copy option with obvious meanings.

Delete Message Property



Delete Msg Property

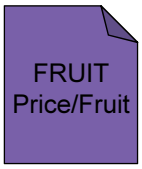
- MQDMPO Options
  - Related to the position of the message property with relation to others already present

```
MQDLTMP( hConn,
         hMsg,
        &dmpo,
        &Name,
        &CompCode,
        &Reason);
```

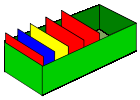
```
MQDMPO   dmpo      = {MQDMPO_DEFAULT};
MQCHARV  Name      = {MQCHARV_DEFAULT};

Name.VSPtr     = "Color";
Name.VSLength  = strlen(Name.VSPtr);
```

Selection of messages

- **MQSUB**
  - Subscribing to specific publications on a topic



- **MQOPEN**
  - Getting message from a queue



```
SubDesc.SelectionString.VSPtr = "Origin = 'Florida'";
SubDesc.SelectionString.VSLength = MQVS_NULL_TERMINATED

ObjDesc.SelectionString.VSPtr = "Colour = 'Blue'";
ObjDesc.SelectionString.VSLength = MQVS_NULL_TERMINATED;
```


Selection of messages - Notes

N O T E S

- Message properties can also be used to selectively consume messages. In a subscribing application you can make a subscription for messages on a specific topic, but additionally only those message on that specific topic which match certain criteria. For example, if you were subscribing on the price of oranges, you might only actually be interested in those where the message property 'Origin' had the value 'Florida'. Doing this means that other messages that do not match what you require are never even put to the subscription destination queue so you do not need to discard those messages that you don't want.

- You can also do selection of messages at MQOPEN time if a point-to-point application wishes to pick out only certain messages. This can be very advantageous for a network connected client application where the saving in network usage is important. Beware deep queues though – MQ is not a database and does not have arbitrary indices for direct access to any message with any arbitrary selection criteria.

Selection of MQMD fields

- **Report Messages**

```
Root.MQMD.MsgType = 4
```

- **Expiry Reports**

```
Root.MQMD.MsgType = 4 AND Root.MQMD.Feedback = 258
```

- **Messages put by a particular user ID**

```
Root.MQMD.UserIdentifier = 'Morag        '
```

- **Messages soon to expire**

```
Root.MQMD.Expiry > 0 AND Root.MQMD.Expiry < 36000
```

- **Specific Correlation ID !!!**

```
Root.MQMD.CorrelId =
"0x414D51434C5558503131312020202020397D6D5976D75E2C"
```


Selection of MQMD fields – Notes

N O T E S

- While MQMD fields are not message properties, it is possible to treat them as if they were. You can omit the MQMD parameter on an MQGET and/or an MQPUT and instead set all the MQMD fields as if they were message properties. I don't see this used often, and it is outside the scope of this introductory presentation.
- However there is one way the interpretation of MQMD fields as message properties can be very useful. That is when you want to select on MQMD fields.
- On the previous page we saw a couple of examples of selection by message property. The difference when selecting on an MQMD field is simply in the syntax of the name. For those of you that have used Message Broker or IIB, the syntax should feel familiar.
- All MQMD fields can be addressed with the syntax:-
  `Root.MQMD.<C header field spelling of field>`
- Several examples of different types of fields on the page. Note that string fields are blank padded, and the spaces must be included (or wildcarded). Not suggesting you'd ever want to use a selector for Correl ID but it acts as an example for a byte string field.

Contents of Message Properties

- Anything you want!
- What you need to select on perhaps?
- Take care with data that should be protected.
- AMS encryption does not apply to message properties

Contents of Message Properties – Notes

N O T E S

- What goes into your Message Properties is entirely up to you. The set of types should be enough for any requirement you can come up with, and of course, if not, most things can be rendered as a string.
- One thing that may guide you is if you need to select messages based on something. That data may be suitable to add as a message property as well as or instead of that information being in the main body of the message.
- Be aware that any data that is added as a message property is not protected by AMS encryption when at rest. The main body of the message is, but the message properties are not – in order to allow selection upon them.
- So don't put data that shouldn't be exposed into a message property.

MQ Technical Conference v2.0.1.7

Summary

- **Several new MQI verbs**
  - MQCRTMH
  - MQDLTMH
  - MQINQMP
  - MQSETMP
  - MQDLTMP
- **Verb structure akin to other familiar MQI verbs**
- **Much easier than parsing an MQRFH2 header!**

Questions & Answers

Morag Hughson – morag@mqgem.com

MQGem Software