Application Programming with MQ Verbs

Chris Leonard
IBM UK

Wednesday 6th August 2014
Session 16203

#SHAREorg
Agenda

- MQI Concepts
- MQI Structures & Datatypes
- Basic MQI walkthrough
- Using Properties With Message Handles
- Using the MQI from Object-oriented applications
Languages

• Procedural (MQI)
  – C
  – COBOL
  – Visual Basic
  – RPG
  – PL/1
  – Assembler
  – TAL

• Object-Oriented (Classes)
  – Java
  – JMS/XMS
  – C++
  – .NET languages
  – ActiveX (MQAX)
  – Perl
Interface

- Simple ‘handle’ based interface
  - Returned handle passed to subsequent call
- Each verb returns
  - Completion Code
    - MQCC_OK 0
    - MQCC_WARNING 1
    - MQCC_FAILED 2
  - Reason Code
    - MQRC_xxxxxxxx 2xxx
    - MQRC_NONE 0
- Make sure you check the reason codes!
Data Structures

- Programmers should be familiar with:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQMD</td>
<td>Message Descriptor</td>
<td>Attributes associated with a message</td>
</tr>
<tr>
<td>MQOD</td>
<td>Object Descriptor</td>
<td>Describes what object to open</td>
</tr>
<tr>
<td>MQSD</td>
<td>Subscription Descriptor</td>
<td>Describes what to subscribe to</td>
</tr>
<tr>
<td>MQPMO</td>
<td>Put Message Options</td>
<td>Describes how a message should be put</td>
</tr>
<tr>
<td>MQGMO</td>
<td>Get Message Options</td>
<td>Describes how a message should be got</td>
</tr>
</tbody>
</table>
Data Structure Tips

- Use structure initialisers
  - MQMD md = { MQMD_DEFAULT };
  - Initialise to version 1
- Structures are versioned
  - Set the minimum version you need
    - md.Version = 2;
  - Don’t use current version
    - md.Version = MQMD_CURRENT_VERSION;
- Bear in mind that some structures are input/output
  - May need to reset values for subsequent call
    - Eg. MsgId & CorrelId fields of MQMD on MQGET call
### MQ Elementary Data Types

- The main MQI data types

<table>
<thead>
<tr>
<th>DataType</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQHCONN</td>
<td>4-byte Connection Handle</td>
</tr>
<tr>
<td>MQHOBJ</td>
<td>4-byte Object Handle</td>
</tr>
<tr>
<td>MQLONG</td>
<td>4-byte binary integer</td>
</tr>
<tr>
<td>MQPTR</td>
<td>Pointer</td>
</tr>
<tr>
<td>MQCHARn</td>
<td>A series of “n” bytes containing character data</td>
</tr>
<tr>
<td>MQBYTEEn</td>
<td>A series of “n” bytes containing binary data</td>
</tr>
<tr>
<td>MQCHARV</td>
<td>Variable length string</td>
</tr>
</tbody>
</table>
Connect

- Basic connect
Connect with extended options

- Handle sharing options
- Client channel specification
- FASTPATH connection
- Additional security settings
- Reconnect option
Connecting

- MQCONNX
  - Don’t hardcode QM name
  - Always check reason codes

- Connections options
  - Connection not thread specific
  - Client reconnect

```c
MQHCONN hQm = MQHC_UNUSABLE_HCONN;
MQCHAR48 Qm = "QM1";
MQCNO cno = {MQCNO_DEFAULT};

cno.Options |= MQCNO_HANDLE_SHARE_BLOCK | MQCNO_RECONNECT;

MQCONNX( Qm, &cno, &hQm, &CompCode, &Reason);
if (CompCode == MQCC_FAILED) {
    /* Do some error processing */
    /* Possibly retry */
}
```
MQCONN(X) Tips

• Don’t hardcode Queue Manager names
  – Pass as parameter or configure in INI file
• Best to use MQCONNX
  – Has options structure should it be needed
• Most expensive verb
  – Don’t issue it repeatedly for each request
    • Often problem for OO languages
• If MQI handle need to be used on different threads
  – Use connection options to indicate the MQI handle can be shared
  – Choose to block or reject any calls from another thread when handle is in use
• If reconnecting use exponential back-off with random wait
  – Try to avoid client storms
• Can dynamically load MQ libraries if client or local binding
  – Preferable to shipping two versions of the program
Open a Queue

MQCONNX

MQOPEN

Connection Handle
Open Options
Object Descriptor

Object Handle
Completion Code
Reason Code

QMGR

- Indicate type of open required
  - input, output, inquire etc

- Indicate object name to open
  - Queue name
  - Topic
Open a queue

- MQOPEN a queue
- OpenOptions
  - MQOO_ flags which are required
- MQOD describes a object to open
  - ObjectType
    - MQOT_Q or MQOT_TOPIC
  - ObjectString/ObjectName

```c
MQHOBJ hObj = MQHO_UNUSABLE_HOBJ;
MQOD ObjDesc = {MQOD_DEFAULT};

ObjDesc.ObjectType = MQOT_Q;
strcpy(ObjectDesc.ObjectName, "Q1");
```

```c
OpenOpts = MQOO_OUTPUT
  | MQOO_FAIL_IFQUIESCING;
MQOPEN( hQm,
    &ObjDesc,
    OpenOpts,
    &hObj,
    &CompCode,
    &Reason);
```
# Object Descriptor (MQOD)

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>StrucId</td>
<td>Structure identifier</td>
<td></td>
</tr>
<tr>
<td>Version</td>
<td>Structure version number</td>
<td>1</td>
</tr>
<tr>
<td>ObjectType</td>
<td>Object type</td>
<td></td>
</tr>
<tr>
<td>ObjectName</td>
<td>Object name</td>
<td></td>
</tr>
<tr>
<td>ObjectQMgrName</td>
<td>Object queue manager name</td>
<td></td>
</tr>
<tr>
<td>DynamicQName</td>
<td>Dynamic queue name</td>
<td></td>
</tr>
<tr>
<td>AlternateUserId</td>
<td>Alternate user identifier</td>
<td></td>
</tr>
<tr>
<td>RecsPresent</td>
<td>Number of object records present</td>
<td></td>
</tr>
<tr>
<td>KnownDestCount</td>
<td>Number of local queues opened successfully</td>
<td>2</td>
</tr>
<tr>
<td>UnknownDestCount</td>
<td>Number of remote queues opened successfully</td>
<td></td>
</tr>
<tr>
<td>InvalidDestCount</td>
<td>Number of queues that failed to open</td>
<td></td>
</tr>
<tr>
<td>ObjectRecOffset</td>
<td>Offset of first object record from start of MQOD</td>
<td></td>
</tr>
<tr>
<td>ResponseRecOffset</td>
<td>Offset of first response record from start of MQOD</td>
<td></td>
</tr>
<tr>
<td>ObjectRecPtr</td>
<td>Address of first object record</td>
<td></td>
</tr>
<tr>
<td>ResponseRecPtr</td>
<td>Address of first response record</td>
<td></td>
</tr>
<tr>
<td>AlternateSecurityId</td>
<td>Alternate security identifier</td>
<td></td>
</tr>
<tr>
<td>ResolvedQName</td>
<td>Resolved queue name</td>
<td>3</td>
</tr>
<tr>
<td>ResolvedQMgrName</td>
<td>Resolved queue manager name</td>
<td></td>
</tr>
<tr>
<td>ObjectString</td>
<td>Long object name</td>
<td></td>
</tr>
<tr>
<td>SelectionString</td>
<td>Selection string</td>
<td>4</td>
</tr>
<tr>
<td>ResObjectString</td>
<td>Resolved long object name</td>
<td></td>
</tr>
<tr>
<td>ResolvedType</td>
<td>Resolved object type</td>
<td></td>
</tr>
</tbody>
</table>
Open Options

- `#define MQOO_BIND_AS_Q_DEF` 0x00000000
- `#define MQOO_READ_AHEAD_AS_Q_DEF` 0x00000000
- `#define MQOO_INPUT_AS_Q_DEF` 0x00000001
- `#define MQOO_INPUT_SHARED` 0x00000002
- `#define MQOO_INPUT_EXCLUSIVE` 0x00000004
- `#define MQOO_BROWSE` 0x00000008
- `#define MQOO_OUTPUT` 0x00000010
- `#define MQOO_INQUIRE` 0x00000020
- `#define MQOO_SET` 0x00000040
- `#define MQOO_SAVE_ALL_CONTEXT` 0x00000080
- `#define MQOO_PASS_IDENTITY_CONTEXT` 0x00000100
- `#define MQOO_PASS_ALL_CONTEXT` 0x00000200
- `#define MQOO_SET_IDENTITY_CONTEXT` 0x00000400
- `#define MQOO_SET_ALL_CONTEXT` 0x00000800
- `#define MQOO_ALTERNATE_USER_AUTHORITY` 0x00001000
- `#define MQOO_FAIL_IF_QUIESCING` 0x00002000
- `#define MQOO_BIND_ON_OPEN` 0x00004000
- `#define MQOO_BIND_NOT_FIXED` 0x00008000
- `#define MQOO_CO_OP` 0x00020000
- `#define MQOO_NO_READ_AHEAD` 0x00080000
- `#define MQOO_READ_AHEAD` 0x01000000

**Options can be ‘ORed’ together as required**
MQOPEN Tips

• Try not to hardcode queue/topic names
• Try not to open queues exclusively
  – Will reduce options for workload balancing
• Use MQPUT1 if only opening queue to put one message
• Consider queue cache for common used queues
  – MQOPEN is relatively expensive – load and security check
• Use read ahead for performance gain
  – If client and non-persistent messaging
• If opening model reply queues
  – Be aware of how many instances of queues you may be creating
    • Particularly large numbers of clients.
  – May be better to share reply queue
Put a message

MQCONNX
MQOPEN

MQPUT

Connection Handle
Object Handle
Message Descriptor
Put Message Options
Message Data

Completion Code
Reason Code

• Updates structure
  • Message Descriptor
  • Put Message Options

QMGR
Putting Application

- MQOPEN a queue
- MQPUT a message
  - Simple Hello World message
  - Set message format to string
  - Put outside of syncpoint

```c
MQMD md = {MQMD_DEFAULT};
MQPMO pmo = {MQPMO_DEFAULT};
char msg = "Hello World!";

memcpy(md.Format, MQFMT_STRING, MQ_FORMAT_LENGTH);

pmo.Options = MQPMO_NO_SYNCPOINT;
```
### Put Options

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQPMO_SYNCPOINT</td>
<td>0x00000002</td>
</tr>
<tr>
<td>MQPMO_NO_SYNCPOINT</td>
<td>0x00000004</td>
</tr>
<tr>
<td>MQPMO_DEFAULT_CONTEXT</td>
<td>0x00000020</td>
</tr>
<tr>
<td>MQPMO_NEW_MSG_ID</td>
<td>0x00000040</td>
</tr>
<tr>
<td>MQPMO_NEW_CORREL_ID</td>
<td>0x00000080</td>
</tr>
<tr>
<td>MQPMO_PASS_IDENTITY_CONTEXT</td>
<td>0x00000100</td>
</tr>
<tr>
<td>MQPMO_PASS_ALL_CONTEXT</td>
<td>0x00000200</td>
</tr>
<tr>
<td>MQPMO_SET_IDENTITY_CONTEXT</td>
<td>0x00000400</td>
</tr>
<tr>
<td>MQPMO_SET_ALL_CONTEXT</td>
<td>0x00000800</td>
</tr>
<tr>
<td>MQPMO_ALTERNATE_USER_AUTHORITY</td>
<td>0x00001000</td>
</tr>
<tr>
<td>MQPMO_FAIL_IF_QUIESCING</td>
<td>0x00002000</td>
</tr>
<tr>
<td>MQPMO_NO_CONTEXT</td>
<td>0x00004000</td>
</tr>
<tr>
<td>MQPMO_LOGICAL_ORDER</td>
<td>0x00008000</td>
</tr>
<tr>
<td>MQPMO_ASYNC_RESPONSE</td>
<td>0x00010000</td>
</tr>
<tr>
<td>MQPMO_SYNC_RESPONSE</td>
<td>0x00020000</td>
</tr>
<tr>
<td>MQPMO_RESOLVE_LOCAL_Q</td>
<td>0x00040000</td>
</tr>
<tr>
<td>MQPMO_WARN_IF_NO_SUBS_MATCHED</td>
<td>0x00080000</td>
</tr>
<tr>
<td>MQPMO_RETAIN</td>
<td>0x00200000</td>
</tr>
<tr>
<td>MQPMO_MD_FOR_OUTPUT_ONLY</td>
<td>0x00800000</td>
</tr>
<tr>
<td>MQPMO_SCOPE_QMGR</td>
<td>0x04000000</td>
</tr>
<tr>
<td>MQPMO_SUPPRESS_REPLYTO</td>
<td>0x08000000</td>
</tr>
<tr>
<td>MQPMO_NOT_OWN_SUBS</td>
<td>0x10000000</td>
</tr>
<tr>
<td>MQPMO_RESPONSE_AS_Q_DEF</td>
<td>0x00000000</td>
</tr>
<tr>
<td>MQPMO_RESPONSE_AS_TOPIC_DEF</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

- Options can be ‘ORed’ together as required
MQPUT Tips

- Always use explicit syncpoint setting
  - Defaults are not the same on z/OS and Distributed
  - Generally
    - Syncpoint when persistent
    - No syncpoint when non-persistent

- Try not to use extreme message sizes
  - QM optimized for message 4K – 1MB

- Consider async put response for performance gain
  - If on client and sending many non-persistent messages
Get a message

Connection Handle
Object Handle
Message Descriptor
Get Message Options
Buffer Size

Message Data
Message Length
Completion Code
Reason Code

Updates structure
- Message Descriptor
- Get Message Options
Getting Application

- MQOPEN a queue
  - Syncpoint if persistent
  - Always ask for convert
  - Wait for message
    - up to one minute

```c
MQMD md = {MQMD_DEFAULT};
MQGMO gmo = {MQGMO_DEFAULT};
gmo.Options = MQGMO_SYNCPOINT_IF_PERSISTENT |
MQGMO_CONVERT |
MQGMO_WAIT |
MQGMO_FAIL_IF_QUIESCING;
gmo.WaitInterval = 60 * 1000;

OpnOpts = MQOO_INPUT_SHARED |
MQOO_FAIL_IF_QUIESCING;
MQOPEN( hConn,
    &od,
    OpnOpts,
    &hObj,
    &CompCode,
    &Reason);

MQGET ( hConn,
    hObj,
    &md,
    &gmo,
    sizeof(msg),
    msg,
    &msglen,
    &CompCode,
    &Reason);
```
# Get Message Options (MQGMO)

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>StrucId</td>
<td>Structure identifier</td>
<td></td>
</tr>
<tr>
<td>Version</td>
<td>Structure version number</td>
<td>1</td>
</tr>
<tr>
<td>Options</td>
<td>Options that control the action of MQGET</td>
<td></td>
</tr>
<tr>
<td>WaitInterval</td>
<td>Wait Interval</td>
<td></td>
</tr>
<tr>
<td>Signal1</td>
<td>Signal</td>
<td></td>
</tr>
<tr>
<td>Signal2</td>
<td>Signal identifier</td>
<td></td>
</tr>
<tr>
<td>ResolvedQName</td>
<td>Resolved name of destination queue</td>
<td></td>
</tr>
<tr>
<td>MatchOptions</td>
<td>Options controlling selection criteria used for MQGET</td>
<td></td>
</tr>
<tr>
<td>GroupStatus</td>
<td>Flag indicating whether message retrieved is in a group</td>
<td>2</td>
</tr>
<tr>
<td>SegmentStatus</td>
<td>Flag indicating whether message retrieved is a segment of a logical message</td>
<td></td>
</tr>
<tr>
<td>Sementation</td>
<td>Flag indicating whether further segmentation is allowed for the message retrieved</td>
<td></td>
</tr>
<tr>
<td>MsgToken</td>
<td>Message token</td>
<td></td>
</tr>
<tr>
<td>ReturnedLength</td>
<td>Length of message data returned (bytes)</td>
<td>3</td>
</tr>
<tr>
<td>MsgHandle</td>
<td>The handle to a message that is to be populated with the properties of the message being retrieved from the queue.</td>
<td>4</td>
</tr>
</tbody>
</table>
### Get Options

<table>
<thead>
<tr>
<th>Option Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQGMO_WAIT</td>
<td>0x00000001</td>
</tr>
<tr>
<td>MQGMO_NO_WAIT</td>
<td>0x00000000</td>
</tr>
<tr>
<td>MQGMO_SET_SIGNAL</td>
<td>0x00000008</td>
</tr>
<tr>
<td>MQGMO_FAIL_IF_QUIESCING</td>
<td>0x00002000</td>
</tr>
<tr>
<td>MQGMO_SYNCPOINT</td>
<td>0x00000002</td>
</tr>
<tr>
<td>MQGMO_SYNCPOINT_IF_PERSISTENT</td>
<td>0x00001000</td>
</tr>
<tr>
<td>MQGMO_NO_SYNCPOINT</td>
<td>0x00000004</td>
</tr>
<tr>
<td>MQGMO_MARK_SKIP_BACKOUT</td>
<td>0x00000080</td>
</tr>
<tr>
<td>MQGMO_BROWSE_FIRST</td>
<td>0x00000010</td>
</tr>
<tr>
<td>MQGMO_BROWSE_NEXT</td>
<td>0x00000020</td>
</tr>
<tr>
<td>MQGMO_BROWSE_MSG_UNDER_CURSOR</td>
<td>0x00000800</td>
</tr>
<tr>
<td>MQGMO_MSG_UNDER_CURSOR</td>
<td>0x00000100</td>
</tr>
<tr>
<td>MQGMO_LOCK</td>
<td>0x00000200</td>
</tr>
<tr>
<td>MQGMO_UNLOCK</td>
<td>0x00000400</td>
</tr>
<tr>
<td>MQGMO_ACCEPT_TRUNCATED_MSG</td>
<td>0x00000040</td>
</tr>
<tr>
<td>MQGMO_CONVERT</td>
<td>0x00004000</td>
</tr>
<tr>
<td>MQGMO_LOGICAL_ORDER</td>
<td>0x00008000</td>
</tr>
<tr>
<td>MQGMO_COMPLETE_MSG</td>
<td>0x00010000</td>
</tr>
<tr>
<td>MQGMO_ALL_MSGS_AVAILABLE</td>
<td>0x00020000</td>
</tr>
<tr>
<td>MQGMO_ALL_SEGMENTS_AVAILABLE</td>
<td>0x00040000</td>
</tr>
<tr>
<td>MQGMO_MARK_BROWSE_HANDLE</td>
<td>0x00100000</td>
</tr>
<tr>
<td>MQGMO_MARK_BROWSE_CO_OP</td>
<td>0x00200000</td>
</tr>
<tr>
<td>MQGMO_UNMARK_BROWSE_CO_OP</td>
<td>0x00400000</td>
</tr>
<tr>
<td>MQGMO_UNMARK_BROWSE_HANDLE</td>
<td>0x00800000</td>
</tr>
<tr>
<td>MQGMO_UNMARKED_BROWSE_MSG</td>
<td>0x01000000</td>
</tr>
<tr>
<td>MQGMO_PROPERTIES_FORCE_MQRFH2</td>
<td>0x02000000</td>
</tr>
<tr>
<td>MQGMO_NO_PROPERTIES</td>
<td>0x04000000</td>
</tr>
<tr>
<td>MQGMO_PROPERTIES_IN_HANDLE</td>
<td>0x08000000</td>
</tr>
<tr>
<td>MQGMO_PROPERTIES_COMPATIBILITY</td>
<td>0x10000000</td>
</tr>
<tr>
<td>MQGMO_PROPERTIES_AS_Q_DEF</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

Options can be ‘ORed’ together as required.
MQGET Tips

• Avoid using default syncpoint setting
  – Defaults are not the same on z/OS and Distributed
  – Generally
    • MQGMO_SYNCPOINT_IF_PERSISTENT

• Use MQGMO_FAIL_IF_QUIESCING
  – Ensure your application ends promptly

• Generally use MQGMO_CONVERT
  – Even if you ‘think’ you don’t need it

• Remember to reset MsgId & CorrelId fields
  – These fields are used for selection and are returned

• Handle ‘poison message’
  – Look at BackoutCount in MQMD

• Consider using MQCB to consume messages instead
  – Callback semantics, often easier to code
Publish a message

 MQCONNX
 MQOPEN

 MQPUT

 Connection Handle
 Object Handle
 Message Descriptor
 Put Message Options
 Message Data

 Completion Code
 Reason Code

 Updates structure
 • Message Descriptor
 • Put Message Options

 Very similar to a normal P2P Put
Publishing Application

- MQOPEN a topic
- MQOD describes a topic to publish to
  - ObjectType
    - MQOT_Q for point-to-point
    - MQOT_TOPIC for publish
  - ObjectString/ObjectName
- MQPUT a message

```
MQOD ObjDesc = {MQOD_DEFAULT};

ObjDesc.ObjectType = MQOT_TOPIC;
ObjDesc.Version = MQOD_VERSION_4;
ObjDesc.ObjectString.VSPtr = "Price/Fruit/Apples";
ObjDesc.ObjectString.VSLength = MQVS_NULL_TERMINATED;
```

```
OpnOpts = MQOO_OUTPUT |
MQOO_FAIL_IF QUIESCING;
MQOPEN(hConn, &ObjDesc,
OpnOpts, &hObj,
&CompCode, &Reason);

MQPUT(hConn, hObj,
&MsgDesc, pmo,
strlen(pBuffer), pBuffer,
&CompCode, &Reason);
```
Publishing Tips

• Choose topic string carefully
  – Use sensible topic hierarchy
    • Based on context of published data

  – Don’t use different topic for each publish
    • This is probably meta data, use message property

  – Topic strings can be up to 10K bytes
    • But don’t use long topics unless necessary

• Consider using Topic object and Topic string
  – Administrator can set point in topic tree
    • Known as ‘topic tree isolation’
Subscribe to a topic

MQCONNX
MQSUB
MQGET

Connection Handle
Subscription Descriptor
Object Handle

Subscription Handle
Completion Code
Reason Code

Updates structure
- Subscription Descriptor

Very similar to MQOPEN
Subscribing Application

- MQSUB verb
- Subscription Descriptor (MQSD) describes the topic
  - MQSD.ObjectString
  - MQSD.ObjectName
- Consume publications from the returned hObj
  - when MQSO_MANAGED used

```c
MQSD SubDesc = {MQSD_DEFAULT};
SubDesc.ObjectString.VSPtr = "Price/Fruit/Apples";
SubDesc.ObjectString.VSLength = MQVS_NULL_TERMINATED;
SubDesc.Options = MQSO_CREATE |
                   MQSO_MANAGED |
                   MQSO_FAIL_IF_QUIESCING;
MQSUB ( hQm, &SubDesc, &hObj, &hSub, &CompCode, &Reason);
MQGET ( hQm, hObj, &MsgDesc, &gmo, strlen(pBuffer), pBuffer, &DataLength, &CompCode, &Reason);
```
# Subscription Descriptor (MQSD)

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StrucId</td>
<td>Structure identifier</td>
</tr>
<tr>
<td>Version</td>
<td>Structure version number</td>
</tr>
<tr>
<td>Options</td>
<td>Options that control the action of MQSUB</td>
</tr>
<tr>
<td>ObjectName</td>
<td>Object Name</td>
</tr>
<tr>
<td>AlternateUserId</td>
<td>Alternate User Id</td>
</tr>
<tr>
<td>AlternateSecurityId</td>
<td>Alternate Security Id</td>
</tr>
<tr>
<td>SubExpire</td>
<td>Subscription expiry</td>
</tr>
<tr>
<td>ObjectString</td>
<td>Object string</td>
</tr>
<tr>
<td>SubName</td>
<td>Subscription name</td>
</tr>
<tr>
<td>SubUserData</td>
<td>Subscription user data</td>
</tr>
<tr>
<td>PubPriority</td>
<td>Publication priority</td>
</tr>
<tr>
<td>PubAccountingToken</td>
<td>Publication accounting token</td>
</tr>
<tr>
<td>PubAppIdentityData</td>
<td>Publication application identity data</td>
</tr>
<tr>
<td>SelectionString</td>
<td>String providing selection criteria</td>
</tr>
<tr>
<td>SubLevel</td>
<td>Subscription Level</td>
</tr>
<tr>
<td>ResObjectString</td>
<td>Resolved object string</td>
</tr>
</tbody>
</table>

Complete your session evaluations online at www.SHARE.org/Pittsburgh-Eval
Subscribe Options

#define MQSO_NON_DURABLE               0x00000000
#define MQSO_READ_AHEAD_AS_Q_DEF       0x00000000
#define MQSO_ALTER                     0x00000001
#define MQSO_CREATE                    0x00000002
#define MQSO_RESUME                    0x00000004
#define MQSO_DURABLE                   0x00000008
#define MQSO_GROUP_SUB                 0x00000010
#define MQSO_MANAGED                   0x00000020
#define MQSO_SET_IDENTITY_CONTEXT      0x00000040
#define MQSO_FIXED_USERID              0x00000100
#define MQSO_ANY_USERID                0x00000200
#define MQSO_PUBLICATIONS_ON_REQUEST   0x00000800
#define MQSO_NEW_PUBLICATIONS_ONLY     0x00001000
#define MQSO_FAIL_IF_QUIESCING         0x00002000
#define MQSO_ALTERNATE_USER_AUTHORITY  0x00040000
#define MQSO_WILDCARD_CHAR             0x00100000
#define MQSO_WILDCARD_TOPIC            0x00200000
#define MQSO_SET_CORREL_ID             0x00400000
#define MQSO_SCOPE_QMGR                0x04000000
#define MQSO_NO_READ_AHEAD             0x08000000
#define MQSO_READ_AHEAD                0x10000000

- Options can be ‘ORed’ together as required
Subscribing Tips

• Managed handles make things simpler

• Only use durable subscriptions when necessary
  – Avoid build up of messages

• For durable subscriptions
  – Combine the create and resume options to make it simpler
Close a handle

MQCONNX
MQOPEN
MQPUT
MQOPEN
MQGET
MQCLOSE

Connection Handle
Object Handle Close
Options

Completion Code
Reason Code

• Updates Object Handle

QMGR
Closing Application

• MQOPEN a queue
• MQCLOSE a queue
  – Normally we’d do something!
  – Note address of MQHOBJ

```
MQHCONN hConn;
MQHOBJ hObj = MQHO_UNUSABLE_HOBJ;
MQOD ObjDesc = {MQOD_DEFAULT};

ObjDesc.ObjectType = MQOT_Q;
strcpy(ObjDesc.ObjectName, “Q1”);
```

```
OpnOpts = MQOO_INPUT_SHARED
    | MQOO_FAIL_IF_QUIESCING;

MQOPEN( hConn,
    &od,
    OpnOpts,
    &hObj,
    &CompCode,
    &Reason);

< Issue some MQI calls here >

MQCLOSE( hConn,
    &hObj,
    MQCO_NONE,
    &CompCode,
    &Reason);
```
Close Options

- Options available depending on object type

<table>
<thead>
<tr>
<th>Close Options</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQCO_DELETE</td>
<td>0x00000001</td>
<td>Permanent Dynamic Queue</td>
</tr>
<tr>
<td>MQCO_DELETE_PURGE</td>
<td>0x00000002</td>
<td>Permanent Dynamic Queue</td>
</tr>
<tr>
<td>MQCO_KEEP_SUB</td>
<td>0x00000004</td>
<td>Durable Subscription</td>
</tr>
<tr>
<td>MQCO_REMOVE_SUB</td>
<td>0x00000008</td>
<td>Durable Subscription</td>
</tr>
<tr>
<td>MQCO_QUIESCE</td>
<td>0x00000020</td>
<td>Read Ahead input handle</td>
</tr>
</tbody>
</table>
MQCLOSE Tips

• In triggered applications
  – Only close triggered queue if application ending

• If implementing queue cache
  – Close ‘rarely used’ queues in a timely fashion
  • Open queues can not be deleted/purged and use memory

• For read ahead queues
  – Use the quiesce close option to avoid message loss
Disconnect from Queue Manager

MQCONNX
MQOPEN
MQPUT
MQOPEN
MQGET
MQCLOSE
MQDISC

Connection Handle
Completion Code
Reason Code

• Updates connection handle
Disconnected Application

- MQCONN to Queue Manager
- MQDISC from Queue Manager
  - Normally we’d do something!
  - Note address of MQHCONN

```c
MQCONN hConn = MQHC_UNUSABLE_HCONN;
MQCHAR48 Qm = "QM1";
MQCNO cno = {MQCNO_DEFAULT};

cno.Options |= MQCNO_HANDLE_SHARE_BLOCK | MQCNO_RECONNECT
```

```
MQCONNX (Qm,
    &cno,
    &hConn,
    &CompCode,
    &Reason);

< Issue some MQI calls here >

MQDISC( &hConn,
    &CompCode,
    &Reason);
```
**MQDISC Tips**

- Ensure application disconnects if QM quiescing
  - Will prevent Queue Manager from ending
- MQDISC will close all queues/topics and subscriptions
  - May wish to close some queues individually
- MQDISC is an implicit commit
  - May want to consider issuing MQBACK() first
- Still call MQDISC
  - If MQI call returns with a connection broken reason code
- Application ending without MQDISC
  - Will backout on Distributed
  - Will commit or backout depending on exit reason on z/OS
  - Try to always do explicit MQDISC if possible
Summary

• Simple MQI – very easy to get started
  – Let most fields retain default values
  – Keep things simple if you can
    • do not try and monitor channels for example

• Plenty of samples to help you along
  – In a variety of languages
    • eg. <install dir>\Tools\c\Samples
    • <hlq>.SCSQC37S

• Check reason codes and log failures
  – MQ trace can be useful
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 MQDLTLMH
Setting Message Properties

- First step is to create a message handle
- Options available to configure validation of property names

```c
MQCONNX(Qm, &cno, &hConn, &CompCode, &Reason);
MQCRTMH(hconn, &CrtMsgHOpts, &hMsg, &CompCode, &Reason);

MQCMHO CrtMsgHOpts = {MQCMHO_DEFAULT};
CrtMsgHOpts.options = MQCMHO_VALIDATE;

MQHMSG hMsg = MQHM_NONE;
```
Setting Message Properties

- Message properties are then set on the handle

- Property types
  - MQTYPE_BOOLEAN
  - MQTYPE_BYTE_STRING
  - MQTYPE_INT8 / 16 / 32 / 64
  - MQTYPE_FLOAT32 / 64
  - MQTYPE_STRING
  - MQTYPE_NULL

```c
MQCRTMH( hConn,
    &CrtMsgHOpts,
    &hMsg,
    &CompCode,
    &Reason);

MQSETMP ( hConn,
    hMsg,
    &SetPropOptts,
    &Name,
    &PropDesc,
    Type,
    ValueLength,
    &Value,
    &CompCode,
    &Reason);

MQSMPO  SetPropOptts = {MQSMPO_DEFAULT};
MQCHARV Name = {MQCHARV_DEFAULT};
Name.VSPtr = "CustomProperty";
Name.VSLength = strlen(Name.VSPtr);
MQPD      PropDesc = {MQPD_DEFAULT};
MQLONG    Type = MQTYPE_STRING;
MQBYTE*   Value = "abc123";
MQLONG    ValueLength = (MQLONG)strlen(Value);
```
Setting Message Properties

- Reference the handle from a version 3 MQPMO structure and MQPUT the message

```c
MQMD md = {MQMD_DEFAULT};
MQPMO pmo = {MQPMO_DEFAULT};
char msg = "Hello World!";
memcpy(md.Format, MQFMT_STRING, MQ_FORMAT_LENGTH);
pmo.Options = MQPMO_NO_SYNCPOINT;

pmo.Version = 3;
pmo.NewMsgHandle = hMsg;
pmo.Action = MQACTP_NEW;
```
Retrieving Message Properties

- A version 4 MQGMO structure contains a reference to the MQMH of the returned message
- Can also use ‘%’ wildcard with MQIMPO_INQ_FIRST and MQIMPO_INQ_NEXT options to iterate over all matching properties

```c
if (gmo.version >= 4){
    MQHMSG hMsg = gmo.MsgHandle;
    MQIMPO InqPropOpts = {MQIMPO_DEFAULT};
    MQCHARV Name = {MQCHARV_DEFAULT};
    Name.VSPtr = "CustomProperty";
    Name.VSLength = strlen(Name.VSPtr);
    MQPD PropDesc = {MQPD_DEFAULT};
    MQLONG ValueLength = VALUELENGTH;
    PMQBYTE Value = (PMQBYTE)malloc(ValueLength); }
```
Using Properties: Selection of messages

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

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

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

ObjDesc.SelectionString.VSPtr = "Colour = 'Blue'";
ObjDesc.SelectionString.VSLength = MQVS_NULL_TERMINATED;
```
Comparing Procedural MQI to Object Oriented (Java): Basic connection

```java
public void connectAndPutMessage()
{

    String queueManagerName = "QMDEMO";

    // Minimum set of properties to establish a TCP client-mode connection:
    MQEnvironment.channel = "DEMO.SVRCONN";   // Defaults to ""
    MQEnvironment.hostname = "localhost";      // Defaults to "localhost"
    MQEnvironment.port = 1414;                  // Defaults to 1414

    try {
        // MQCONN
        MQQueueManager queueManager = new MQQueueManager(queueManagerName);

        // MQDISC
        queueManager.disconnect();
    }
    catch (MQException e) {
        System.err.println("An exception occurred with CC=" + e.completionCode + " RC=" +
               e.reasonCode);
        System.err.println(e.localizedMessage());
    }
}
```
Comparing Procedural MQI to Object Oriented (Java): PUT a message

```java
public void connectAndPutMessage()
{
    String queueManagerName = "QMDEMO";
    String queueName = "Q1";

    // Minimum set of properties to establish a TCP client-mode connection:
    MQEnvironment.channel = "DEMO.SVRCONN";  // Defaults to ""
    MQEnvironment.hostname = "localhost";     // Defaults to "localhost"
    MQEnvironment.port = 1414;                // Defaults to 1414

    try
    {
        // MQCONN
        MQQueueManager queueManager = new MQQueueManager(queueManagerName);

        // Configure the open options
        int openOpts = MQConstants.MQOO_FAIL_IF_QUIESCING + MQConstants.MQOO_OUTPUT;

        // MQOPEN
        MQQueue queue = queueManager.accessQueue(queueName, openOpts);

        // Create the message
        MQMessage message = new MQMessage();
        message.format = MQConstants.MQFMT_STRING;
        message.writeString("My message text");

        // Create the MQPMO - represented by MQPutMessageOptions object with
        // options field
        MQPutMessageOptions mqpmo = new MQPutMessageOptions();
        mqpmo.options = MQConstants.MQPMO_NO_SYNCPOINT;

        // MQPUT
        queue.put(message, mqpmo);

        // The message object is updated by the PUT
        // For example, might want to record the messageID:
        // byte[] returnedMessageID = message.messageID;

        // MQCLOSE
        queue.close();

        // MQDISC
        queueManager.disconnect();
    } catch (MQException e) {
        System.err.println("An exception occurred with CC=" + e.completionCode + " RC=" + e.reasonCode);
        System.err.println(e.getLocalizedMessage());
    }
}
```
Comparing Procedural MQI to Object Oriented (Java): PUT a message

```java
public void connectAndPutMessage()
{
    String queueManagerName = "QMDEMO";
    String queueName = "Q1";

    MQEnvironment.channel = "DEMO.SVRCONN";
    MQEnvironment.hostname = "localhost";
    MQEnvironment.port = 1414;

    try {
        MQQueueManager queueManager = new MQQueueManager(queueManagerName);

        // Configure the open options
        int openOpts = MQConstants.MQOO_FAIL_IF_QUIESCING + MQConstants.MQOO_OUTPUT;

        // MQOPEN
        MQQueue queue = queueManager.accessQueue(queueName, openOpts);

        // Create the message
        MQMessage message = new MQMessage();
        message.format = MQConstants.MQFMT_STRING;
        message.writeString("My message text");

        // Create the MQPMO - represented by MQPutMessageOptions object with options field
        MQPutMessageOptions mqpmo = new MQPutMessageOptions();
        mqpmo.options = MQConstants.MQPMO_NO_SYNCPOINT;

        // MQPUT
        queue.put(message, mqpmo);

        // The message object is updated by the PUT
        // For example, might want to record the messageID:
        // byte[] returnedMessageID = message.messageId;

        // MQCLOSE
        queue.close();

        // MQDISC
        queueManager.disconnect();
    } catch (MQException e) {
        System.err.println("An exception occurred with CC=" + e.completionCode + " RC=" + e.reasonCode);
        System.err.println(e.getLocalizedMessage());
    }
}
```
public void connectAndPutMessage()
{
    String queueManagerName = "QMDEMO";
    String queueName = "Q1";

    // Minimum set of properties to establish a TCP client-mode connection:
    MQEnvironment.channel = "DEMO.SVRCONN"; // Defaults to ""
    MQEnvironment.hostname = "localhost"; // Defaults to "localhost"
    MQEnvironment.port = 1414; // Defaults to 1414

    try {
        MQQueueManager queueManager = new MQQueueManager(queueManagerName);
        // Configure the open options
        int openOpts = MQConstants.MQOO_FAIL_IF_QUIESCING + MQConstants.MQOO_OUTPUT;
        // MQOPEN
        MQQueue queue = queueManager.accessQueue(queueName, openOpts);

        // Create the message
        MQMessage message = new MQMessage();
        message.format = MQConstants.MQFMT_STRING;
        message.writeString("My message text");

        // Create the MQPMO - represented by MQPutMessageOptions object with
        // options field
        MQPutMessageOptions mqpmo = new MQPutMessageOptions();
        mqpmo.options = MQConstants.MQPMO_NO_SYNCPOINT;

        // MQPUT
        queue.put(message, mqpmo);

        // The message object is updated by the PUT
        // For example, might want to record the messageID:
        // byte[] returnedMessageID = message.messageId;

        // MQCLOSE
        queue.close();

        // MQDISC
        queueManager.disconnect();
    } catch (MQException e) {
        System.err.println("An exception occurred with CC=" + e.completionCode + " RC=" +
                           e.reasonCode);
        System.err.println(e.getLocalizedMessage());
    }
}
### Comparing Procedural MQI to Object Oriented (Java): PUT a message

```java
public void connectAndPutMessage()
{
    String queueManagerName = "QMDEMO";
    String queueName = "Q1";

    // Minimum set of properties to establish a TCP client-mode connection:
    MQEnvironment.channel = "DEMO.SVRCONN";  // Defaults to ""
    MQEnvironment.hostname = "localhost";       // Defaults to "localhost"
    MQEnvironment.port = 1414;                  // Defaults to 1414

    try {
        MQQueueManager queueManager = new MQQueueManager(queueManagerName);

        // Configure the open options
        int openOpts = MQConstants.MQOO_FAIL_IF_QUIESCING + MQConstants.MQOO_OUTPUT;

        MQQueue queue = queueManager.accessQueue(queueName, openOpts);

        // Create the message
        MQMessage message = new MQMessage();
        message.format = MQConstants.MQFMT_STRING;
        message.writeString("My message text");

        // Create the MQPMO - represented by MQPutMessageOptions object with
        // options field
        MQPutMessageOptions mqpmo = new MQPutMessageOptions();
        mqpmo.options = MQConstants.MQPMO_NO_SYNCPOINT;

        // MQPUT
        queue.put(message, mqpmo);

        // The message object is updated by the PUT
        // For example, might want to record the messageID:
        // byte[] returnedMessageID = message.messageId;

        // MQCLOSE
        queue.close();

        // MQDISC
        queueManager.disconnect();
    } catch (MQException e) {
        System.err.println("An exception occurred with CC=" + e.completionCode + " RC=" + e.reasonCode);
        System.err.println(e.getLocalizedMessage());
    }
}
```
Comparing Procedural MQI to Object Oriented (Java): PUT a message

```java
public void connectAndPutMessage()
{
    String queueManagerName = "QMDEMO";
    String queueName = "Q1";

    // Minimum set of properties to establish a TCP client-mode connection:
    MQEnvironment.channel = "DEMO.SVRCONN"; // Defaults to ""
    MQEnvironment.hostname = "localhost"; // Defaults to "localhost"
    MQEnvironment.port = 1414; // Defaults to 1414

    try {
        // MQCONN
        MQQueueManager queueManager = new MQQueueManager(queueManagerName);

        // Configure the open options
        int openOpts = MQConstants.MQOO_FAIL_IF_QUIESCING + MQConstants.MQOO_OUTPUT;

        // MQOPEN
        MQQueue queue = queueManager.accessQueue(queueName, openOpts);

        // Create the message
        MQMessage message = new MQMessage();
        message.format = MQConstants.MQFMT_STRING;
        message.writeString("My message text");

        // Create the MQPMO - represented by MQPutMessageOptions object with
        // options field
        MQPutMessageOptions mqpmo = new MQPutMessageOptions();
        mqpmo.options = MQConstants.MQPMO_NO_SYNCPOINT;

        // MQPUT
        queue.put(message, mqpmo);

        // The message object is updated by the PUT
        // For example, might want to record the messageID:
        // byte[] returnedMessageID = message.messageId;

        // MQCLOSE
        queue.close();

        // MQDISC
        queueManager.disconnect();
    }
    catch (MQException e) {
        System.err.println("An exception occurred with CC=" + e.completionCode + " RC=" + e.reasonCode);
        System.err.println(e.getLocalizedMessage());
    }
}
```
Comparing Procedural MQI to Object Oriented (Java): Accessing other MQI verbs

// MQGET
queue.get((MQMessage) msg);
queue.get(msg, (MQGetMessageOptions) gmo);
queue.get(msg, gmo, (int) maxMsgSize);

// MQPUT1
queueManager.put(((String) queueName, (MQMessage) msg);
Comparing Procedural MQI to Object Oriented (Java): Accessing other MQI verbs

// Setup for MQSUB
int openAs = MQConstants.MQTOPIC_OPEN_AS_SUBSCRIPTION;

int openOptionsForGet = MQConstants.MQSO_CREATE
                     + MQConstants.MQSO_FAIL_IFQUIESCING
                     + MQConstants.MQSO_MANAGED
                     + MQConstants.MQSO_NON_DURABLE;

String topicString = "/sport/football";
String topicObject = "ADMINISTRATIVE.TOPIC";

// MQSUB
MQTopic topic = queueManager.accessTopic(topicString,
topicObject, openAs, openOptionsForGet);
This was session 16203 – The rest of the week…

<table>
<thead>
<tr>
<th>Time</th>
<th>Monday</th>
<th>Tuesday</th>
<th>Wednesday</th>
<th>Thursday</th>
<th>Friday</th>
</tr>
</thead>
<tbody>
<tr>
<td>08:30</td>
<td></td>
<td></td>
<td>Application programming with MQ verbs</td>
<td>The Dark Side of Monitoring MQ - SMF 115 and 116 Record Reading and Interpretation</td>
<td>CICS and MQ - Workloads Unbalanced!</td>
</tr>
<tr>
<td>10:00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11:15</td>
<td>Introduction to MQ</td>
<td>What's New in IBM Integration Bus &amp; WebSphere Message Broker</td>
<td>MQ – Take Your Pick Lab</td>
<td>Using IBM WebSphere Application Server and IBM WebSphere MQ Together</td>
<td></td>
</tr>
<tr>
<td>12:15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01:30</td>
<td></td>
<td>All about the new MQ v8</td>
<td>MQ Security: New v8 features deep dive</td>
<td>New MQ Chinit monitoring via SMF</td>
<td></td>
</tr>
<tr>
<td>03:00</td>
<td>MQ Beyond the Basics</td>
<td>MQ &amp; DB2 – MQ Verbs in DB2 &amp; InfoSphere Data Replication (Q Replication) Performance</td>
<td>What's wrong with MQ?</td>
<td>IIIB - Internals of IBM Integration Bus</td>
<td></td>
</tr>
<tr>
<td>04:15</td>
<td>First Steps with IBM Integration Bus: Application Integration in the new world</td>
<td>MQ for z/OS v8 new features deep dive</td>
<td>MQ Clustering - The Basics, Advances and What's New in v8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Copyright and Trademarks

© IBM Corporation 2014. All Rights Reserved.

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at “Copyright and trademark information” at www.ibm.com/legal/copytrade.shtml.