17051 - MQ for z/OS and Distributed Application Programming with MQ Verbs

Thursday March 5th 2015
08:30 - 09:30am
Seneca, Sheraton Seattle
Mayur Raja (mayur_raja@uk.ibm.com)
IBM’s statements regarding its plans, directions, and intent are subject to change or withdrawal without notice, at IBM’s sole discretion.

Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.

The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract. The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon many factors, including 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 results similar to those stated here.
Agenda

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

Requesting Application

Request Queue

Serving Application

MQCONN (to QMGR)
MQOPEN (Request Queue)
MQPUT (to Request Queue)
MQPUT (to Request Queue)
..
..
MQCLOSE (Request Queue)
MQDISC (from QMGR)

MQCONN (to QMGR)
MQOPEN (Request Queue)
MQGET (from Request Queue)
MQGET (from Request Queue)
..
..
MQCLOSE (Request Queue)
MQDISC (from QMGR)

Complete your session evaluations online at www.SHARE.org/Seattle-Eval
Languages that the MQI can be coded in

• Procedural (MQI)
  – C
  – COBOL (z/OS)
  – Visual Basic
  – RPG (IBM i)
  – PL/1 (z/OS)
  – Assembler (z/OS)
  – pTAL (Portable Transaction Application Language for HP NonStop Systems)

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

- Simple ‘handle’ based interface
  - Returned handle passed to subsequent call
    - HCONN (MQCONN) and HOBJ (MQOPEN)

- 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>
<th>MQVerb</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQOD</td>
<td>Object Descriptor</td>
<td>Describes what object to open</td>
<td>MQOPEN</td>
</tr>
<tr>
<td>MQMD</td>
<td>Message Descriptor</td>
<td>Attributes associated with a message</td>
<td>MQPUT, MQPUT1, MQGET</td>
</tr>
<tr>
<td>MQPMO</td>
<td>Put Message Options</td>
<td>Describes how a message should be put</td>
<td>MQPUT, MQPUT1</td>
</tr>
<tr>
<td>MQGMO</td>
<td>Get Message Options</td>
<td>Describes how a message should be got</td>
<td>MQGET</td>
</tr>
<tr>
<td>MQSD</td>
<td>Subscription Descriptor</td>
<td>Describes what to subscribe to</td>
<td>MQSUB</td>
</tr>
</tbody>
</table>
Data Structure Tips

• Use structure initialisers
  – \texttt{MQMD \textit{md} = \{ MQMD\_DEFAULT \};}
  – Initialise to version 1

• Structures are versioned
  – Set the minimum version you need
    • \texttt{md.\textit{Version} = 2;}
  – Don’t use current version
    • \texttt{md.\textit{Version} = MQMD\_CURRENT\_VERSION;}

• Bear in mind that some structures are input/output
  – May need to reset values for subsequent call
    • Eg. \texttt{MsgId} & \texttt{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

- Connection handle sharing options
- Client channel specification (MQCD)
- FASTPATH connection
- Additional security parameters
- Reconnect option for clients

Application

MQCONNX

Queue Manager Name
Connection Options

Connection Handle
Completion Code
Reason Code

QMGR

Complete your session evaluations online at www.SHARE.org/Seattle-Eval
Connecting

- **MQCONNX**
  - Don’t hardcode QM name!
  - Always check Completion Code
  - Check Reason and handle error

- 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 needs 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

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

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

- **MQOPEN** a queue

- **MQOD** describes an object to open
  - **ObjectType**
    - MQOT_Q
  - **ObjectName**
    - String

- **OpenOptions**
  - MQOO_* (required options)

```c
MQOBJ hObj = MQHO_UNUSABLE_HOBJ;
MQOD ObjDesc = {MQOD_DEFAULT};
ObjDesc.ObjectType = MQOT_Q;
strcpy(ObjectDesc.ObjectName, "Q1");

OpenOpts = MQOO_OUTPUT
       | MQOO_FAIL_IFQUIESCING;

MQOPEN(hQm,
       &ObjDesc,
       OpenOpts,
       &hObj,
       &CompCode,
       &Reason);
```
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>StruclId</td>
<td>Structure identifier</td>
<td></td>
</tr>
<tr>
<td>Version</td>
<td>Structure version number</td>
<td></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></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></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></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 (MQOO_*)

- `#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
  – Pass in as parameters

• Try not to open Queues exclusively
  – Will reduce options for workload balancing

• Only use MQPUT1 if you really do want to put one message
  – MQPUT1 = MQOPEN + MQPUT + MQCLOSE

• Cache handles of frequently used queues
  – MQOPEN is relatively expensive
    • Loads queue definition
    • Performs open security checks
**MQOPEN Tips..**

- If running **client**, and getting **non-persistent messages**
  - Use read ahead for performance gain
    - MQOO_READ_AHEAD
  - Messages are read ahead of time into in memory buffers
  - Reduces interactions on client channel
  - Any messages in buffers are lost if client terminates

- If opening model queue to create a reply to queue:
  - Be aware of how many instances of queues you may be creating
    - Particularly with large numbers of clients
  - May be better to share a reply queue
Put a message

Application
MQCONNX
MQOPEN
MQPUT

QMGR
Queue

Connection Handle
Object Handle
Message Descriptor
Put Message Options
Message Data

Completion Code
Reason Code

• Updates structure
  • Message Descriptor
  • Put Message Options
Put a Message

• **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
| MQPMO_FAIL_IF_QUIESCING;

MQPUT ( hConn,
        hObj,
        &md,
        &pmo,
        strlen(msg),
        msg,
        &CompCode,
        &Reason);
```
Put Options (MQPMO_*)

```c
#define MQPMO_SYNCPOINT                0x00000002
#define MQPMO_NO_SYNCPOINT             0x00000004
#define MQPMO_DEFAULT_CONTEXT          0x00000020
#define MQPMO_NEW_MSG_ID               0x00000040
#define MQPMO_NEW_CORREL_ID            0x00000080
#define MQPMO_PASS_IDENTITY_CONTEXT    0x00000100
#define MQPMO_PASS_ALL_CONTEXT         0x00000200
#define MQPMO_SET_IDENTITY_CONTEXT     0x00000400
#define MQPMO_SET_ALL_CONTEXT          0x00000800
#define MQPMO_ALTERNATE_USER_AUTHORITY 0x00001000
#define MQPMO_FAIL_IF_QUIESCING        0x00002000
#define MQPMO_NO_CONTEXT               0x00004000
#define MQPMO_LOGICAL_ORDER            0x00008000
#define MQPMO_ASYNC_RESPONSE           0x00010000
#define MQPMO_SYNC_RESPONSE            0x00020000
#define MQPMO_RESOLVE_LOCAL_Q          0x00040000
#define MQPMO_WARN_IF_NO_SUBS_MATCHED  0x00080000
#define MQPMO_RETAIN                   0x00200000
#define MQPMO_MD_FOR_OUTPUT_ONLY       0x00800000
#define MQPMO_SCOPE_QMGR               0x04000000
#define MQPMO_SUPPRESS_REPLYTO         0x08000000
#define MQPMO_NOT_OWN_SUBS             0x10000000
#define MQPMO_RESPONSE_AS_Q_DEF        0x00000000
#define MQPMO_RESPONSE_AS_TOPIC_DEF    0x00000000
```

Options can be ‘ORed’ together as required.
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

Complete your session evaluations online at www.SHARE.org/Seattle-Eval
Getting Application

- **MQOPEN** a queue
  - For input
- **MQGET** a message
  - Syncpoint if persistent
  - Always ask for convert
  - Wait for message
    - e.g. Wait for one min

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

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;

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

<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></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></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></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></td>
</tr>
</tbody>
</table>
Get Options (MQGMO_*)

#define MQGMO_WAIT                     0x00000001
#define MQGMO_NO_WAIT                  0x00000000
#define MQGMO_SET_SIGNAL               0x00000008
#define MQGMO_FAIL_IF_QUIESCING        0x00002000
#define MQGMO_SYNCPOINT                0x00000002
#define MQGMO_SYNCPOINT_IF_PERSISTENT  0x00001000
#define MQGMO_NO_SYNCPOINT             0x00000004
#define MQGMO_MARK_SKIP_BACKOUT        0x00000080
#define MQGMO_BROWSE_FIRST             0x00000010
#define MQGMO_BROWSE_NEXT              0x00000020
#define MQGMO_BROWSE_MSG_UNDER_CURSOR  0x00000800
#define MQGMO_MSG_UNDER_CURSOR         0x00000100
#define MQGMO_LOCK                     0x00000200
#define MQGMO_UNLOCK                   0x00000400
#define MQGMO_ACCEPT_TRUNCATED_MSG     0x00000040

Options can be ‘ORed’ together as required
Get Options (MQGMO_*) ..

<table>
<thead>
<tr>
<th>Definition</th>
<th>Hex Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>#define MQGMO_CONVERT</td>
<td>0x00004000</td>
</tr>
<tr>
<td>#define MQGMO_LOGICAL_ORDER</td>
<td>0x00008000</td>
</tr>
<tr>
<td>#define MQGMO_COMPLETE_MSG</td>
<td>0x00010000</td>
</tr>
<tr>
<td>#define MQGMO_ALL_MSGS_AVAILABLE</td>
<td>0x00020000</td>
</tr>
<tr>
<td>#define MQGMO_ALL_SEGMENTS_AVAILABLE</td>
<td>0x00040000</td>
</tr>
<tr>
<td>#define MQGMO_MARK_BROWSE_HANDLE</td>
<td>0x00100000</td>
</tr>
<tr>
<td>#define MQGMO_MARK_BROWSE_CO_OP</td>
<td>0x00200000</td>
</tr>
<tr>
<td>#define MQGMO_UNMARK_BROWSE_CO_OP</td>
<td>0x00400000</td>
</tr>
<tr>
<td>#define MQGMO_UNMARK_BROWSE_HANDLE</td>
<td>0x00800000</td>
</tr>
<tr>
<td>#define MQGMO_UNMARKED_BROWSE_MSG</td>
<td>0x01000000</td>
</tr>
<tr>
<td>#define MQGMO_PROPERTIES_FORCE_MQRFH2</td>
<td>0x02000000</td>
</tr>
<tr>
<td>#define MQGMO_NO_PROPERTIES</td>
<td>0x04000000</td>
</tr>
<tr>
<td>#define MQGMO_PROPERTIES_IN_HANDLE</td>
<td>0x08000000</td>
</tr>
<tr>
<td>#define MQGMO_PROPERTIES_COMPATIBILITY</td>
<td>0x10000000</td>
</tr>
<tr>
<td>#define 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, use
    • 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 by MQGET

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

• Consider using MQCB to consume messages instead
  – Callback semantics, often easier to code
MQI – Simple verbs for Publish/Subscribe

MQCONN (to QMGR)
MQSUB (Topic)
MQGET publication from Topic
MQCLOSE (Topic)
MQDISC (from QMGR)

MQCONN (to QMGR)
MQOPEN (Topic)
MQPUT (publish) message to Topic
MQCLOSE (Topic)
MQDISC (from QMGR)
Subscribe to a topic

- Updates structure
  - Subscription Descriptor
- Very similar to MQOPEN
Publish Tips

- Subscription handle can be used to terminate the subscription (maybe because no publications were made to the topic that was subscribed to)
Get (wait for) a Publication

- Updates structure
  - Message Descriptor
  - Get Message Options

Applications:
- MQCONNX
- MQSUB
- MQGET

MQGET:
- Message (Publication) Data
- Message (Publication) Length
- Completion Code
- Reason Code
- Buffer Size
- Connection Handle
- Object Handle
- Message Descriptor
- Get Message Options

QMGR
Subscribing Application

- **Subscription Descriptor (MQSD)**
  - Describes the topic
  - MQSD.ObjectString
  - MQSD.ObjectName

- **MQSUB verb**
  - Subscribe to a topic

- **MQGET verb**
  - Use **hObj** returned on MQSUB call
  - Consume publications
  - when **MQSO_MANAGED** used
    - Storage of msgs managed by QMGR

```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>SubExpiry</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>Publiation 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>
### Subscribe Options (MQSO_*)

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQSO_NON_DURABLE</td>
<td>0x00000000</td>
</tr>
<tr>
<td>MQSO_READ_AHEAD_AS_Q_DEF</td>
<td>0x00000000</td>
</tr>
<tr>
<td>MQSO_ALTER</td>
<td>0x00000004</td>
</tr>
<tr>
<td>MQSO_CREATE</td>
<td>0x00000002</td>
</tr>
<tr>
<td>MQSO_RESUME</td>
<td>0x00000004</td>
</tr>
<tr>
<td>MQSO_DURABLE</td>
<td>0x00000008</td>
</tr>
<tr>
<td>MQSO_GROUP_SUB</td>
<td>0x00000010</td>
</tr>
<tr>
<td>MQSO_MANAGED</td>
<td>0x00000020</td>
</tr>
<tr>
<td>MQSO_SET_IDENTITY_CONTEXT</td>
<td>0x00000040</td>
</tr>
<tr>
<td>MQSO_FIXED_USERID</td>
<td>0x00000100</td>
</tr>
<tr>
<td>MQSO_ANY_USERID</td>
<td>0x00000200</td>
</tr>
<tr>
<td>MQSO_PUBLICATIONS_ON_REQUEST</td>
<td>0x00000800</td>
</tr>
<tr>
<td>MQSO_NEW_PUBLICATIONS_ONLY</td>
<td>0x00001000</td>
</tr>
<tr>
<td>MQSO_FAIL_IF_QUIESCING</td>
<td>0x00002000</td>
</tr>
<tr>
<td>MQSO_ALTERNATE_USER_AUTHORITY</td>
<td>0x00040000</td>
</tr>
<tr>
<td>MQSO_WILDCARD_CHAR</td>
<td>0x00100000</td>
</tr>
<tr>
<td>MQSO_WILDCARD_TOPIC</td>
<td>0x00200000</td>
</tr>
<tr>
<td>MQSO_SET_CORREL_ID</td>
<td>0x00400000</td>
</tr>
<tr>
<td>MQSO_SCOPE_QMGR</td>
<td>0x04000000</td>
</tr>
<tr>
<td>MQSO_NO_READ_AHEAD</td>
<td>0x08000000</td>
</tr>
<tr>
<td>MQSO_READ_AHEAD</td>
<td>0x10000000</td>
</tr>
</tbody>
</table>

Options can be ‘ORed’ together as required.
Subscribe 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
Publish a message

- Updates structure
  - Message Descriptor
  - Put Message Options
- Very similar to a normal P2P Put
Publishing Application

- **MQOD** describes a topic
  - **ObjectType**
    - MQOT_TOPIC
  - **ObjectString/ObjectName**
    - Topic string
- **MQOPEN** Topic
- **MQPUT** – publish msg

```c
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);
```
Publish 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
Publish Tips

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

- MQCONNX
- MQOPEN
- MQPUT
- MQOPEN
- MQGET
- MQCLOSE
- MQCLOSE

Connection Handle
Object Handle Close
Options

Completion Code
Reason Code

- Updates Object Handle

Complete your session evaluations online at www.SHARE.org/Seattle-Eval
Closing Application

- MQCLOSE a queue
  - Supply hObj from MQOPEN/MQSUB call

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

- Options available depending on object type

| MQCO_NONE | 0x00000000 | No optional close processing required |
| MQCO_IMMEDIATE | 0x00000000 | Any unconsumed messages in the Read Ahead Buffer are deleted and the queue is closed. |
| MQCO_DELETE | 0x00000001 | Deletes a Permanent Dynamic Queue | Deletes a Temporary Dynamic Queue if it was created by the hObj specified on this close request. |
| MQCO_DELETE_PURGE | 0x00000002 | Deletes a Permanent Dynamic Queue and any Messages on the queue | Deletes a Temporary Dynamic Queue and any Messages on the queue if it was created by the hObj specified on this close request. |
| MQCO_KEEP_SUB | 0x00000004 | Closes handle to Subscription but keeps Durable Subscription. |
| MQCO_REMOVE_SUB | 0x00000008 | Removes Durable Subscription and closes handle to Subscription. |
| MQCOQUIESCE | 0x00000020 | Allows messages in the Read Ahead Buffer to be consumed before the queue is closed. |

Complete your session evaluations online at www.SHARE.org/Seattle-Eval
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

**Application**
- MQCONNX
- MQOPEN
- MQPUT
- MQOPEN
- MQGET
- MQCLOSE
- MQCLOSE
- MQDISC

**QMGR**

**Connection Handle**

**Completion Code**

**Reason Code**

- Updates connection handle

---

Complete your session evaluations online at [www.SHARE.org/Seattle-Eval](http://www.SHARE.org/Seattle-Eval)
Disconnecting Application

- MQDISC from Queue Manager
  - Supply hConn from MQCONN/MQCONNX call

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

• Ensure application disconnects if QM quiescing
  – Otherwise, will prevent Queue Manager from ending

• MQDISC will close all queues/topics and subscriptions
  – May wish to close some queues individually

• MQDISC is generally an implicit commit
  – May want to consider issuing MQBACK first, if required

• 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

• Plenty of samples to help you along
  – In a variety of languages
    • eg. `<install dir>\Tools\c\Samples`
    • `<hlq>.SCSQC37S`
    • Articles/papers (just search the internet)

• Check reason codes, and log any failures
  – MQ trace can be useful
Message Properties

• **Data (or Control Information)**
  - Associated with a message

• **Consists of:**
  - Textual name
  - Value - of a particular type

• **Supported Types**
  - MQTYPE_BOOLEAN
  - MQTYPE_BYTE_STRING
  - MQTYPE_INT8 / 16 / 32 / 64
  - MQTYPE_FLOAT32 / 64
  - MQTYPE_STRING
  - MQTYPE_NULL

• **Can be used as message selectors to:**
  - Get selective messages from queues
  - Filter publications to topics
Message Properties

- **Control information about a message**
  - MQMD fields – predefined
  - Message Properties – any name + value of particular type

  e.g.

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>messageColour</td>
<td>Red</td>
<td>MQTYPE_STRING</td>
</tr>
</tbody>
</table>

- **User Data – the Message Body**
  - User-defined format (as per today)

---

Complete your session evaluations online at www.SHARE.org/Seattle-Eval
Message Properties

**Setting Message Properties**

- Create Message Handle
- Set Message Properties in handle
- Associate Message Handle with PMO
- Put Message

**Retrieving Message Properties**

- Create Message Handle
- Associate Message Handle with GMO
- Get Message
- Inquire Properties in Message Handle

Queue

(Now have Message Handle)

Complete your session evaluations online at www.SHARE.org/Seattle-Eval
Setting Message Properties

- First, create a **Message Handle**
  - Represents message
  - Can specify option to validate property names on set message property call

```c
MQCMHO CrtMsgHOpts = {MQCMHO_DEFAULT};
CrtMsgHOpts.options = MQCMHO_VALIDATE;
MQHMSG hMsg = MQHM_NONE;
MQCRTMH( hconn,
    &CrtMsgHOpts,
    &hMsg,
    &CompCode,
    &Reason);
```
Message Handle

- Retrieved on MQGET
- Can be provided on MQPUT
  - MQPMO.Action
    - MQACTP_NEW
    - MQACTP_FORWARD
    - MQACTP_REPLY
    - MQACTP_REPORT
  - Represents relationship between two messages
- MQDLTMH delete Message Handle

```c
MQCRTMH(hConn, &cmho, &hMsg, &CompCode, &Reason);
gmo.MsgHandle = hMsg;
MQGET(hConn, ....);

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

- Next, set message properties on the handle that was just created

```c
MQSMPO   SetPropOpts = {MQSMPO_DEFAULT};
MQCHARV  Name        = {MQCHARV_DEFAULT};
Name.VSPtr           = "messageColour";
Name.VSLength        = strlen(Name.VSPtr);
MQPD     PropDesc    = {MQPD_DEFAULT};
MQLONG   Type        = MQTYPE_STRING;
MQBYTE*  Value       = "Red";
MQLONG   ValueLength = (MQLONG)strlen(Value);

MQSETMP (hConn,
         hMsg,
         &SetPropOpts,
         &Name,
         &PropDesc,
         Type,
         ValueLength,
         &Value,
         &CompCode,
         &Reason);
```
Setting Message Properties ..

- Next, associate the handle with a V3 MQPMO structure and put the message

```c
MQPMO pmo       = {MQPMO_DEFAULT};
pmo.Options     = MQPMO_NO_SYNCPOINT;
pmo.Version     = 3;
pmo.NewMsgHandle = hMsg;
pmo.Action      = MQACTP_NEW;

MQMD md        = {MQMD_DEFAULT};
char msg       = "Hello World!";
memcpy(md.Format, MQFMT_STRING, MQ_FORMAT_LENGTH);

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

'Hello World' message now on queue with messageColour 'Red'
Retrieving Message Properties

- First, create a message handle

```c
MQCMHO CrtMsgHOpts = {MQCMHO_DEFAULT};
MQHMSG hMsg         = MQHM_NONE;
MQCRTMH( hconn,
          &CrtMsgHOpts,
          &hMsg,
          &CompCode,
          &Reason);
```
Retrieving Message Properties..

- Create a V4 GMO
- Set message handle in GMO
- Set GMO options
  - Indicate properties should be passed back in message handle
- Get message

```c
MQGMO gmo = {MQGMO_DEFAULT};
gmo.Options = MQGMO_NO_SYNCPOINT;
gmo.Version = 4;
gmo.MsgHandle = hMsg;
gmo.Options = MQGMO_PROPERTIES_IN_HANDLE;

MQGET ( hQm, 
    hObj, 
    &MsgDesc, 
    &gmo, 
    strlen(pBuffer), 
    pBuffer, 
    &DataLength, 
    &CompCode, 
    &Reason);
```
Retrieving Message Properties ..

- Set up parameters for inquire message properties call
- Issue inquire call

MQIMPO InqPropOpts = {MQIMPO_DEFAULT};
MQCHARV Name       = {MQCHARV_DEFAULT};
Name.VSPtr         = "messageColour";
Name.VSLength      = strlen(Name.VSPtr);
MQPD PropDesc      = {MQPD_DEFAULT};
MQLONG ValueLength = VALUELENGTH;
PMQBYTE Value      = (PMQBYTE)malloc(ValueLength);

MQINQMP (hConn,
    hMsg,
    &InqPropOpts,
    &Name,
    &PropDesc,
    &Type,
    ValueLength,
    Value,
    &DataLength,
    &CompCode,
    &Reason);
Retrieving Message Properties

• Can also use ‘%’ wildcard with MQIMPO_INQ_FIRST and MQIMPO_INQ_NEXT options to iterate over all matching properties
MQMHBUF and MQBUFMH

- MQMHBUF – Convert message handle into an RFH2 format message in a buffer
- MQBUFMH – Converts RFH2 format message in a buffer into a Message Handle
- These calls can save the need to write application logic to parse RFH2 headers
Using Properties for Message Selection

- MQOPEN
  - Getting message from a queue

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

- MQSUB
  - Subscribing to specific publications on a topic

```c
SubDesc.SelectionString.VSPtr = "City = 'Seattle'";
SubDesc.SelectionString.VSLength = MQVS_NULL_TERMINATED
```
Procedural MQI V's Object Oriented (Java): Basic connect and disconnect

```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.getLocalizedMessage());
    }
}
```
public void connectAndPutMessage(){

    String queueManagerName = "QMDEMO";
    String queueName = "Q1";

    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());
    }
}

Connect and put message public void connectAndPutMessage(){

    String queueManagerName = "QMDEMO";
    String queueName = "Q1";

    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());
    }
}

Complete your session evaluations online at www.SHARE.org/Seattle-Eval
public void connectAndPutMessage()
{
    String queueManagerName = "QMDEMO";
    String queueName = "Q1";

    // Minimum set of properties to establish a TCP client-mode connection:
    MQEnvironment.channel = "DEMO.SVRCONN";
    MQEnvironment.hostname = "localhost";
    MQEnvironment.port = 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=");
        System.err.println(e.completionCode);
        System.err.println("RC=");
        System.err.println(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 {
        // 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();

    } catch (MQException e) {
        System.err.println("An exception occurred with CC=" + e.completionCode + " RC=" + e.reasonCode);
        System.err.println(e.getLocalizedMessage());
    }
}

Complete your session evaluations online at wwwSHARE.org/Seattle-Eval
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=" + ecompletionCode + " 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
    {
        // 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");

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

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

// MQPUT1
queueManager.put((String) queueName, (MQMessage) msg);
Object Oriented (Java): Subscribing to a Topic

```java
// 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);
```

**Note:** There are other Java classes and methods

IDENTIFICATION DIVISION.

PROGRAM-ID. CSQ4BVK1.

REMARKS

Sample program to put a number of messages to a queue.

Limitation: Maximum message length set at 65535.
z/OS Batch COBOL: WORKING STORAGE SECTION

* ENVIRONMENT DIVISION.
* ------------------------------------------------------------- *
* DATA DIVISION.
* ------------------------------------------------------------- *
* FILE SECTION.
* ------------------------------------------------------------- *
* WORKING-Storage SECTION.
* ------------------------------------------------------------- *

* W00 - General work fields
*
01 W00-RETURN-CODE       PIC S9(4) BINARY VALUE ZERO.
01 W00-LOOP              PIC S9(9) BINARY VALUE 0.
01 W00-NUMPUTS           PIC S9(9) BINARY VALUE 0.
01 W00-ERROR-MESSAGE     PIC X(48) VALUE SPACES.
*
* Parameter variables
*
01 W00-QMGR              PIC X(48).
01 W00-QNAME             PIC X(48).
01 W00-PADCHAR           PIC X(1) VALUE '*'.
01 W00-MSGBUFFER.
   02 W00-MSGBUFFER-ARRAY PIC X(1) OCCURS 65535 TIMES.
01 W00-NUMMSGS-NUM       PIC 9(4) VALUE 0.
01 W00-NUMMSGS           PIC S9(9) BINARY VALUE 1.
01 W00-MSGLENGTH-NUM     PIC 9(4) VALUE 0.
01 W00-MSGLENGTH         PIC S9(9) BINARY VALUE 100.
01 W00-PERSISTENCE       PIC X(1) VALUE 'N'.
   88 PERSISTENT         VALUE 'P'.
   88 NOT-PERSISTENT     VALUE 'N'.
W03 - API fields

01 W03-HCONN PIC S9(9) BINARY VALUE 0.
01 W03-HOBJ PIC S9(9) BINARY VALUE 0.
01 W03-OPENOPTIONS PIC S9(9) BINARY.
01 W03-COMPCODE PIC S9(9) BINARY.
01 W03-REASON PIC S9(9) BINARY.

API control blocks

01 MQM-OBJECT-DESCRIPTOR.
COPY CMQODV.

01 MQM-MESSAGE-DESCRIPTOR.
COPY CMQMDV.

01 MQM-PUT-MESSAGE-OPTIONS.
COPY CMQPMOV.

MQV contains constants (for filling in the control blocks) and return codes (for testing the result of a call)

01 MQM-CONSTANTS.
COPY CMQV SUPPRESS.

------------------------------------------ LINKAGE SECTION. ------------------------------------------
PROCEDURE DIVISION USING PARMDATA.

A-MAIN SECTION.

If no parameters passed to program then
call USAGE-ERROR and exit

IF PARM-LEN = 0 THEN
   PERFORM USAGE-ERROR
   MOVE 8 TO W00-RETURN-CODE
   GO TO A-MAIN-END
END-IF.

Move parameters into corresponding variables

UNSTRING PARM-STRING DELIMITED BY ALL ','
   INTO W00-QMGR
      W00-QNAME
      W00-NUMMSGS-NUM
      W00-PADCHAR
      W00-MSGLENGTH-NUM
      W00-PERSISTENCE.
   MOVE W00-MSGLENGTH-NUM TO W00-MSGLENGTH.
   MOVE W00-NUMMSGS-NUM TO W00-NUMMSGS.
Display parameters to be used in the program

DISPLAY '==========================================='.
DISPLAY 'PARAMETERS PASSED :'.
DISPLAY ' QMGR        - ', W00-QMGR.
DISPLAY ' QNAME       - ', W00-QNAME.
DISPLAY ' NUMMSGS    - ', W00-NUMMSGS.
DISPLAY ' PADCHAR     - ', W00-PADCHAR.
DISPLAY ' MSGLENGTH   - ', W00-MSGLENGTH.
DISPLAY ' PERSISTENCE - ', W00-PERSISTENCE.
DISPLAY '==========================================='.

Setup the message buffer

PERFORM WITH TEST BEFORE VARYING W00-LOOP FROM 1 BY 1 UNTIL (W00-LOOP > W00-MSGLENGTH)

MOVE W00-PADCHAR TO W00-MSGBUFFER-ARRAY(W00-LOOP)

END-PERFORM.
Connect to the queue manager

CALL 'MQCONN' USING W00-QMGR
    W03-HCONN
    W03-COMPCODE
    W03-REASON.

If connection failed then display error message and exit

IF (W03-COMPCODE NOT = MQCC-OK) THEN
    MOVE 'MQCONN' TO W00-ERROR-MESSAGE
    PERFORM DISPLAY-ERROR-MESSAGE
    MOVE W03-REASON TO W00-RETURN-CODE
    GO TO A-MAIN-END
END-IF.

DISPLAY 'MQCONN SUCCESSFUL'.
Open the queue for output. Fail the call if the queue manager is quiescing.

* COMPUTE W03-OPENOPTIONS = MQOO-OUTPUT + MQOO-FAIL-IF-QUIESCING.

MOVE W00-QNAME TO MQOD-OBJECTNAME.

* CALL 'MQOPEN' USING W03-HCONN
  MQOD
  W03-OPENOPTIONS
  W03-HOBJ
  W03-COMPCODE
  W03-REASON.

* If open failed then display error message and exit.
* IF (W03-COMPCODE NOT = MQCC-OK) THEN
  MOVE 'MQOPEN' TO W00-ERROR-MESSAGE
  PERFORM DISPLAY-ERROR-MESSAGE
  MOVE W03-REASON TO W00-RETURN-CODE
  GO TO A-MAIN-DISCONNECT
END-IF.
DISPLAY 'MQOPEN SUCCESSFUL'.

Complete your session evaluations online at www.SHARE.org/Seattle-Eval
Set persistence depending on parameter passed

IF PERSISTENT THEN
  MOVE MQPER-PERSISTENT TO MQMD-PERSISTENCE
ELSE
  MOVE MQPER-NOT-PERSISTENT TO MQMD-PERSISTENCE
END-IF.

Put string format messages

MOVE MQFMT-STRING TO MQMD-FORMAT.

Set the put message options to fail the call if the queue manager is quiescing

MOVE MQPMO-FAIL-IF-QUIESCING TO MQPMO-OPTIONS.
* * Loop until specified number of messages put to queue
* *
PERFORM WITH TEST BEFORE VARYING W00-LOOP FROM 0 BY 1
UNTIL (W00-LOOP >= W00-NUMMSGS)
*
MOVE MQMI-NONE TO MQMD-MSGID
MOVE MQCI-NONE TO MQMD-CORRELID
*
CALL 'MQPUT' USING W03-HCONN
W03-HOBJ
MQMD
MQPMO
W00-MSGLENGTH
W00-MSGBUFFER
W03-COMPCODE
W03-REASON
*
* If put failed then display error message
* and break out of loop
*
IF (W03-COMPCODE NOT = MQCC-OK) THEN
MOVE 'MQPUT' TO W00-ERROR-MESSAGE
PERFORM DISPLAY-ERROR-MESSAGE
MOVE W00-NUMMSGS TO W00-LOOP
MOVE W03-REASON TO W00-RETURN-CODE
ELSE
ADD 1 TO W00-NUMPUTS
ENDIF
*
END-PERFORM.
Display the number of messages successfully put to the queue.

```
DISPLAY W00-NUMPUTS, 'MESSAGES PUT TO QUEUE'.
```

Close the queue.

```
CALL 'MQCLOSE' USING W03-HCONN W03-HOBJ MQCO-NONE W03-COMPCODE W03-REASON.
IF (W03-COMPCODE NOT = MQCC-OK) THEN
  MOVE 'MQCLOSE' TO W00-ERROR-MESSAGE
  PERFORM DISPLAY-ERROR-MESSAGE
  MOVE W03-REASON TO W00-RETURN-CODE
ELSE
  DISPLAY 'MQCLOSE SUCCESSFUL'
END-IF.
```
A-MAIN-DISCONNECT.
*    Disconnect from the queue manager
*
CALL 'MQDISC' USING W03-HCONN
     W03-COMPCODE
     W03-REASON.
IF (W03-COMPCODE NOT = MQCC-OK) THEN
   MOVE 'MQDISC' TO W00-ERROR-MESSAGE
   PERFORM DISPLAY-ERROR-MESSAGE
   MOVE W03-REASON TO W00-RETURN-CODE
ELSE
   DISPLAY 'MQDISC SUCCESSFUL'
END-IF.
*
A-MAIN-END.
*
*
MOVE W00-RETURN-CODE TO RETURN-CODE
STOP RUN.
*
z/OS Batch COBOL: ERROR MESSAGE SECTIONS

* USAGE-ERROR SECTION.
* ------------------------------------------------------------- *

DISPLAY '=================================================='.
DISPLAY 'PARAMETERS FOR PROGRAM :'.
DISPLAY '   QMGR        - QUEUE MANGER'.
DISPLAY '   QNAME      - QUEUE NAME'.
DISPLAY '   NUMMSGS   - NUMBER OF MESSAGES'.
DISPLAY '   PADCHAR   - MESSAGE PADDING CHARACTER'.
DISPLAY '   MSGLENGTH - LENGTH OF MESSAGE(S)'.
DISPLAY '   PERSISTENCE - PERSISTENCE OF MESSAGE(S)'.
DISPLAY '=================================================='.

* USAGE-ERROR-END.
* *
* RETURN TO PERFORMING FUNCTION
*
EXIT.
*
* DISPLAY-ERROR-MESSAGE SECTION.
* ------------------------------------------------------------- *

DISPLAY '************************************************'.
DISPLAY '* ', W00-ERROR-MESSAGE.
DISPLAY '* COMPLETION CODE : ', W03-COMPCODE.
DISPLAY '* REASON CODE     : ', W03-REASON.
DISPLAY '************************************************'.

* DISPLAY-ERROR-MESSAGE-END.
* *
* RETURN TO PERFORMING FUNCTION
*

Complete your session evaluations online at www.SHARE.org/Seattle-Eval
Note: MQ for z/OS ships many other programs:

- Put and Get samples
- Browse sample
- Print message sample
- Publish/Subscribe samples
- Other samples

in:

- COBOL,
- Assembler,
- PL/1
- C

See: http://www-01.ibm.com/support/knowledgecenter/SSFKSJ_8.0.0/com.ibm.mq.dev.doc/q025180_.htm
Questions?

Complete your session evaluations online at www.SHARE.org/Seattle-Eval
### Session Schedule

<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>10:00</td>
<td>17036: Introduction to MQ - Can MQ Really Make My Life Easier? [z/OS &amp; Distributed]</td>
<td>17052: MQ Beyond the Basics - Advanced API and Internals Overview [z/OS &amp; Distributed]</td>
<td>17046: Paging Dr. MQ - Health Check Your Queue Managers to Ensure They Won't Be Calling in Sick! [z/OS]</td>
<td>17057: Not Just Migrating, but Picking up New Enhancements as You Go - We've Given You the Shotgun, You Know Where Your Feet Are [z/OS &amp; Distributed]</td>
<td></td>
</tr>
<tr>
<td>11:15</td>
<td>17041: First Steps with IBM Integration Bus: Application Integration in the New World [z/OS &amp; Distributed]</td>
<td>16732: MQ V8 Hands-on Labs! MQ V8 with CICS and COBOL! MQ SMF Labs! Room: Redwood</td>
<td>17046: Paging Dr. MQ - Health Check Your Queue Managers to Ensure They Won't Be Calling in Sick! [z/OS]</td>
<td>17053: MQ &amp; DB2 – MQ Verbs in DB2 &amp; InfoSphere Data Replication (Q Replication) Performance [z/OS]</td>
<td></td>
</tr>
</tbody>
</table>

Complete your session evaluations online at www.SHARE.org/Seattle-Eval
Remember to submit your evaluation please

Complete your session evaluations online at www.SHARE.org/Seattle-Eval
Copyright and Trademarks

© IBM Corporation 2015. 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.