Application Programming with MQ Verbs (z/OS & Distributed)

17903, Dolphin, Oceanic 3, Tues Aug 11th 2015, 3:15 - 4:15pm

Mayur Raja
IBM Hursley Park, Winchester, UK
mayur_raja@uk.ibm.com

#SHAREorg

SHARE is an independent volunteer-run information technology association that provides education, professional networking and industry influence.

Legal Disclaimer

The information contained in this publication is provided for informational purposes only. While efforts were made to verify the completeness and accuracy of the information contained in this publication, it is provided AS IS without warranty of any kind, express or implied. In addition, this information is based on IBM's current product plans and strategy, which are subject to change by IBM without notice. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, this publication or any other materials. Nothing contained in this publication is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software.

References in this presentation to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. Product release dates and/or capabilities referenced in this presentation may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way. Nothing contained in these materials is intended to, nor shall have the effect of, stating or implying that any activities undertaken by you will result in any specific sales, revenue growth or other results.

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.

All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics may vary by customer.

Any statements regarding IBM's future direction, intent or product plans are subject to change or withdrawal without notice.

Please Note

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.

MQ Sessions this week

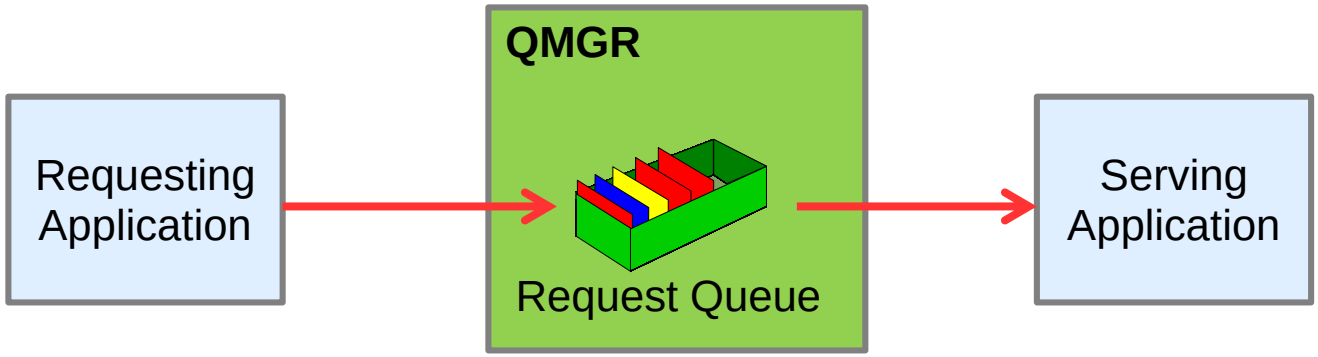
	Monday	Tuesday	Wednesday	Thursday	Friday
08:30			MQ for z/OS, Using and Abusing New Hardware and the New v8 Features	Nobody Uses Files Any More Do They? New Technologies for Old Technology, File Processing in MQ MFT and IIB	Monitoring and Auditing MQ Securing MQ Initiated CICS Workload
10:00	Introduction to MQ - Can MQ Really Make My Life Easier?	MQ for z/OS: The Insider Story	IBM Integration Bus MQ Flexibility	Common Problems and Problem Determination for MQ z/OS	IBM MQ and IBM Integration Bus - from Migration and Maintenance to Continuous Enhancements, How and Why to Stay Current
11:15	Introduction to IBM Integration Bus on z/OS	Introduction to the New MQ Appliance	MQ V8 Hands-on Labs! MQ V8 with CICS and COBOL! MQ SMF Labs!		
12:15					
1:45	What's New in the Messaging Family - MQ v8 and More		Getting Started with Performance of MQ on z/OS	IBM MQ: Are z/OS & Distributed Platforms Like Oil & Water?	
3:15	What's New in IBM Integration Bus	Live!: End to End Security of My Queue Manager on z/OS Application Programming with MQ Verbs	Digging into the MQ SMF Data	MQ Parallel Sysplex Exploitation, Getting the Best Availability from MQ on z/OS by Using Shared Queues	
4:30	MQ Security: New v8 Features Deep Dive	Live!: What's the Cloud Going to Do to My MQ Network? The Do's and Don'ts of IBM Integration Bus Performance	Giving It the Beans: Using IBM MQ as the Messaging Provider for JEE Applications in IBM WebSphere Application Server	Challenge the MQ & IIB Experts?	

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

Agenda

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

MQI - Simple Verbs



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

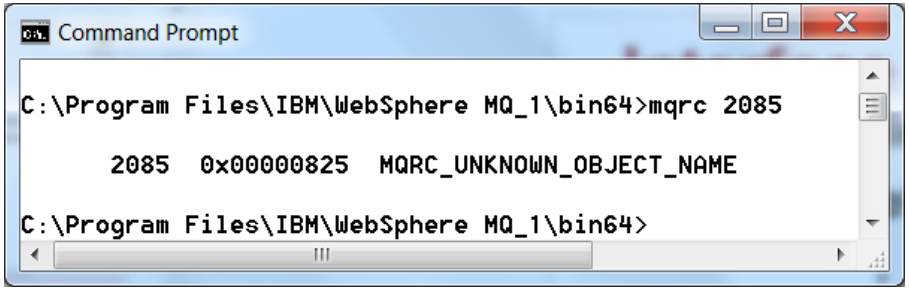
MQCONN (to QMGR)
MQOPEN (Request Queue)
MQGET (from Request Queue)
MQGET (from Request Queue)
..
..
MQCLOSE (Request Queue)
MQDISC (from QMGR)

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:

Name	Description	Purpose	MQVerb
MQOD	Object Descriptor	Describes what object to open	MQOPEN
MQMD	Message Descriptor	Attributes associated with a message	MQPUT, MQPUT1, MQGET
MQPMO	Put Message Options	Describes how a message should be put	MQPUT, MQPUT1
MQGMO	Get Message Options	Describes how a message should be got	MQGET
MQSD	Subscription Descriptor	Describes what to subscribe to	MQSUB

Data Structure Tips

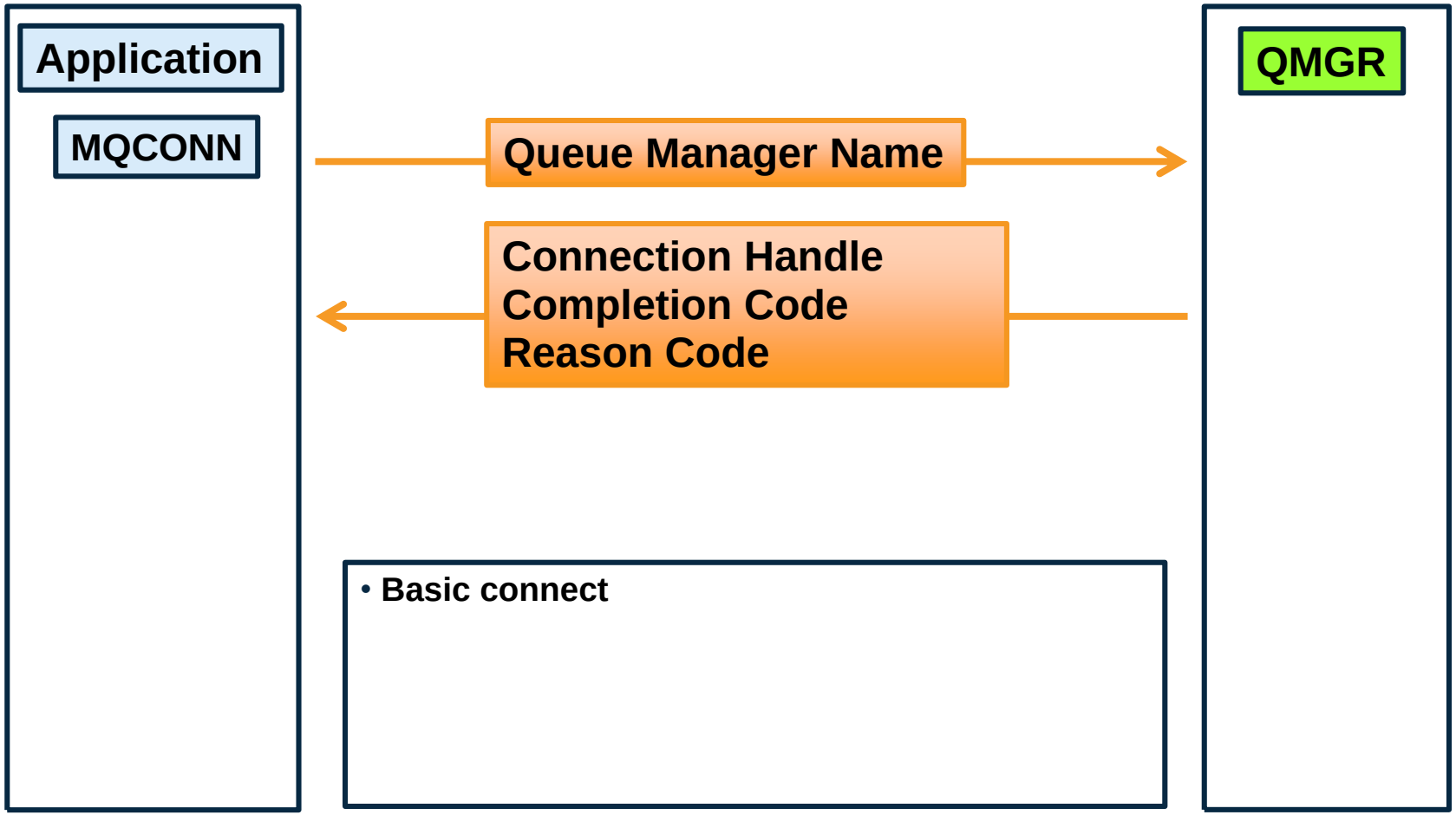
- Use structure initialisers
  - `MQMD md = { MQMD_DEFAULT };`
  - Initialises to Version 1
- Structures are versioned
  - Set the minimum version you need
    - `md.Version = 2;`
  - Don't use current version
    - `md.Version = MQMD_CURRENT_VERSION;`
- Bear in mind that some structures are input/output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

MQ Elementary Data Types

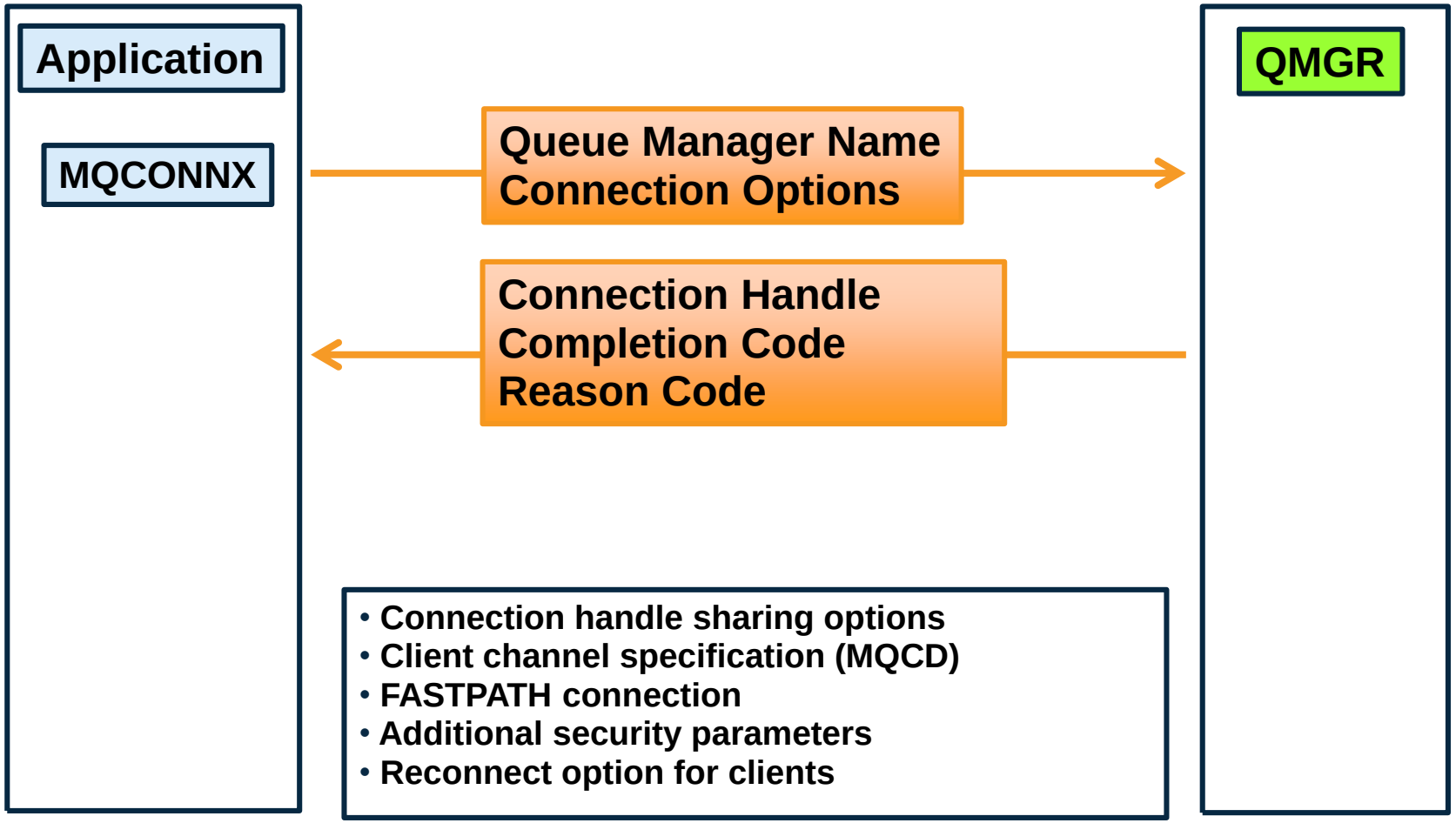
- The main MQI data types

DataType	Purpose
MQHCONN	4-byte Connection Handle
MQHOBJ	4-byte Object Handle
MQLONG	4-byte binary integer
MQPTR	Pointer
MQCHARn	A series of “n” bytes containing character data
MQBYTEn	A series of “n” bytes containing binary data
MQCHARV	Variable length string

Connect


Application
MQCONN
Queue Manager Name
Connection Handle
Completion Code
Reason Code
QMGR
• Basic connect

Connect with extended options



Connecting

```
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           */
}
```

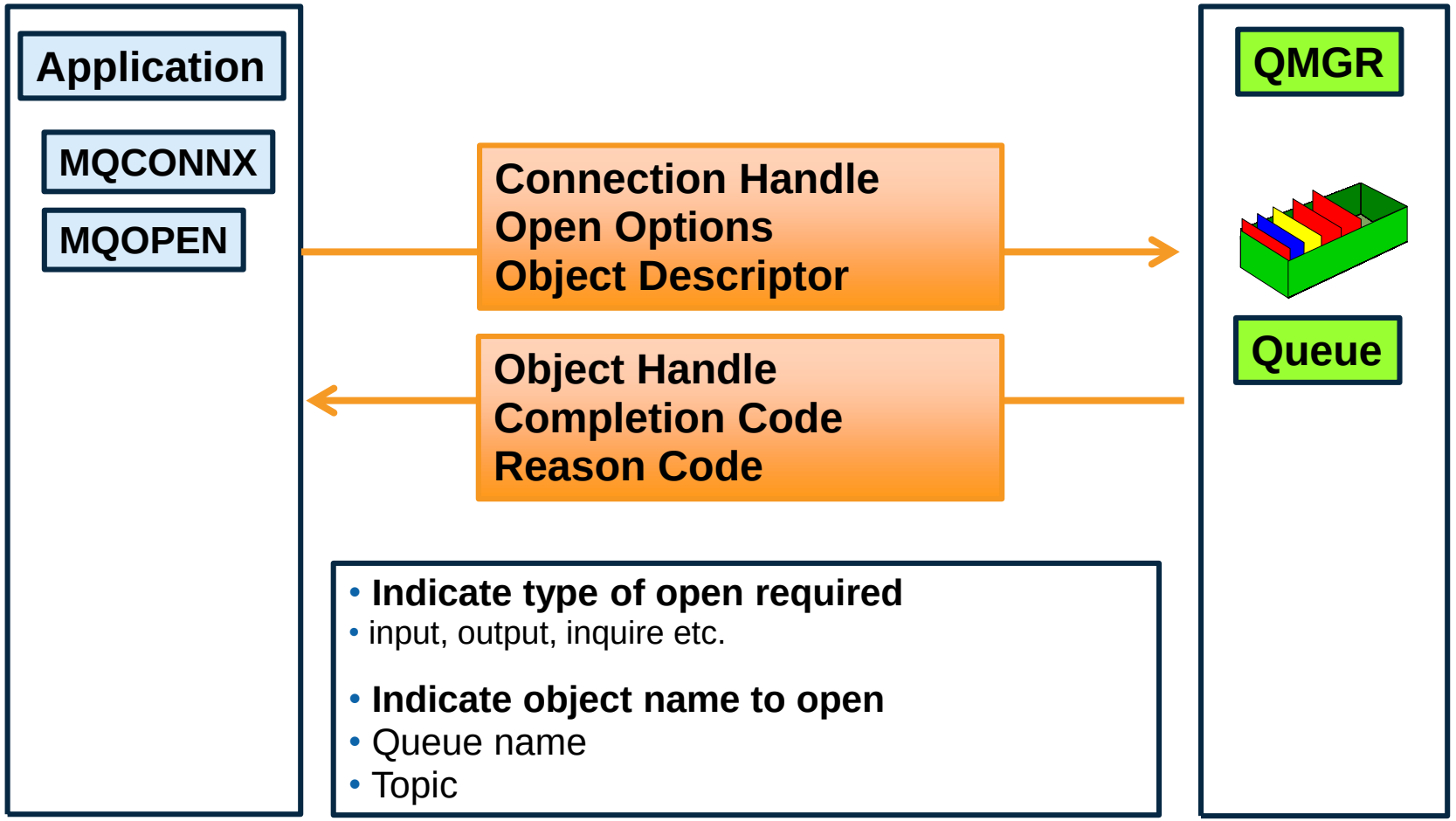
- **MQCONNX**
  - Don't hardcode QM name !
  - Always check Completion Code
  - Check Reason, and handle error
- Connection options
  - Connection not thread specific
  - Can be shared by multiple threads for a given process
  - But, only one thread can use the handle at a time
  - Client reconnect
    - No affinity to a given QMgr

Note: 17894 – MQ Security New V8 Features Deep Dive covers details of passing of UserID and password on an MQCONNX call for authentication.

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 if the MQI handle can be shared
  - Choose to block (MQCNO_HANDLE_SHARE_BLOCK) or reject (MQCNO_HANDLE_SHARE_NO_BLOCK) 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


Application
MQCONNX
MQOPEN
Connection Handle
Open Options
Object Descriptor
Object Handle
Completion Code
Reason Code
QMGR
Queue
• Indicate type of open required
• input, output, inquire etc.
• Indicate object name to open
• Queue name
• Topic

Open a queue

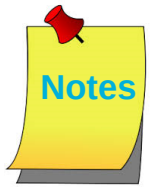
```
MQHOBJ hObj      = MQHO_UNUSABLE_HOBJ;
MQOD   ObjDesc   = {MQOD_DEFAULT};

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

```
OpenOpts = MQOO_OUTPUT
         | MQOO_FAIL_IF_QUIESCING;

MQOPEN( hQm,
        &ObjDesc,
        OpenOpts,
        &hObj,
        &CompCode,
        &Reason);
```

- **MQOPEN** a queue
- MQOD describes an object to open
  - ObjectType
    - MQOT_Q
  - ObjectName
    - String
- OpenOptions
  - MQOO_* (required options)
  - MQOO_INQUIRE | MQOO_INPUT_SHARED | MQOO_OUTPUT

Object Descriptor (MQOD) Structure

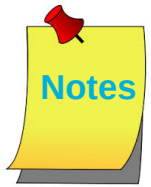
Field	Description	Version
StrucId	Structure identifier	1
Version	Structure version number	
ObjectType	Object type	
ObjectName	Object name	
ObjectQMgrName	Object queue manager name	
DynamicQName	Dynamic queue name	
AlternateUserId	Alternate user identifier	
RecsPresent	Number of object records present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



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                   0x00100000
```

- **Options can be 'ORed' together as required**



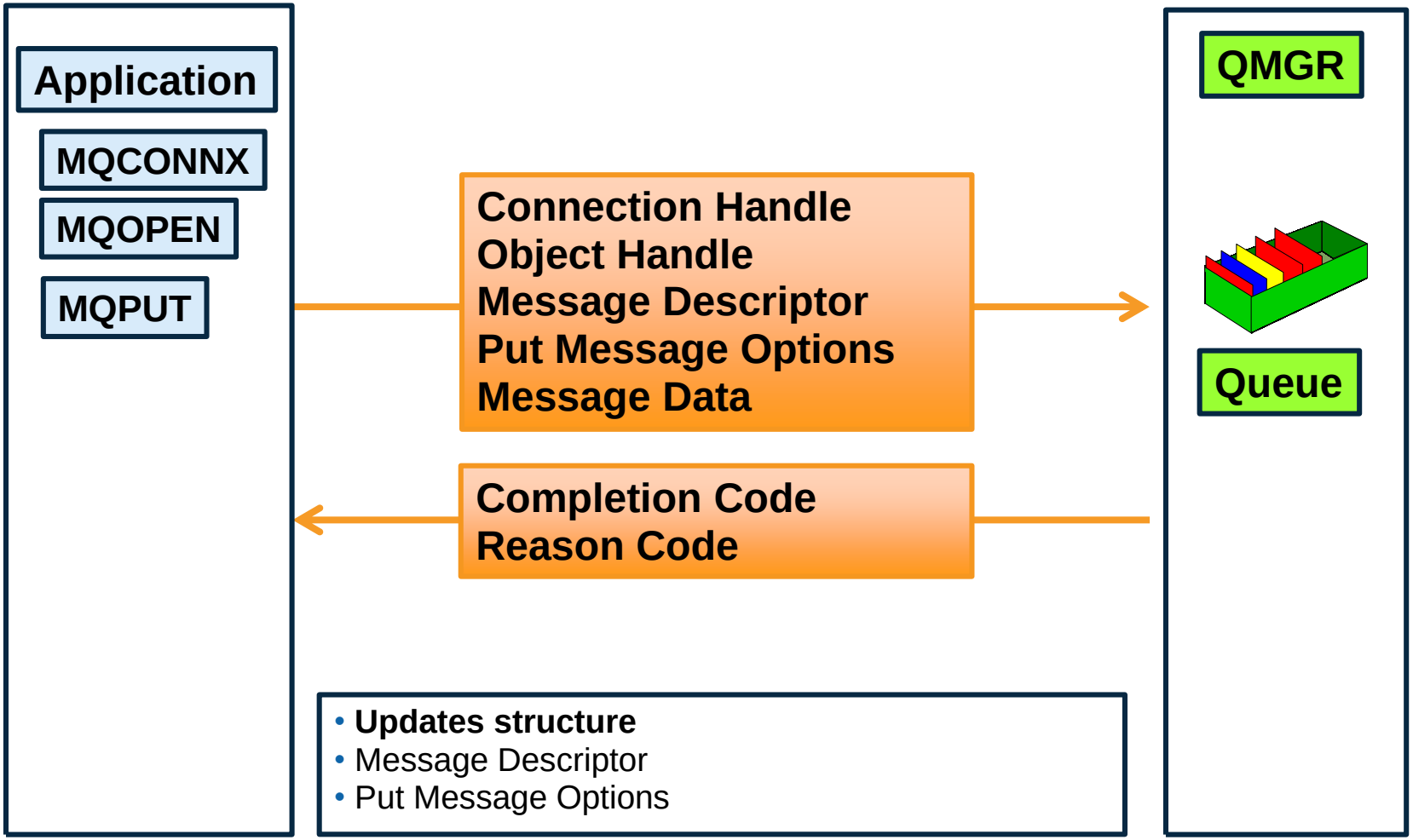
MQOPEN Tips

- 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 as a **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
  - Queue Creation is expensive
    - May be better to share a pre-defined reply to queue

Put a message



Put a Message

```
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);
```

- **MQPUT** a message
  - Simple "Hello World" message
  - Set message format to string
  - Put outside of syncpoint

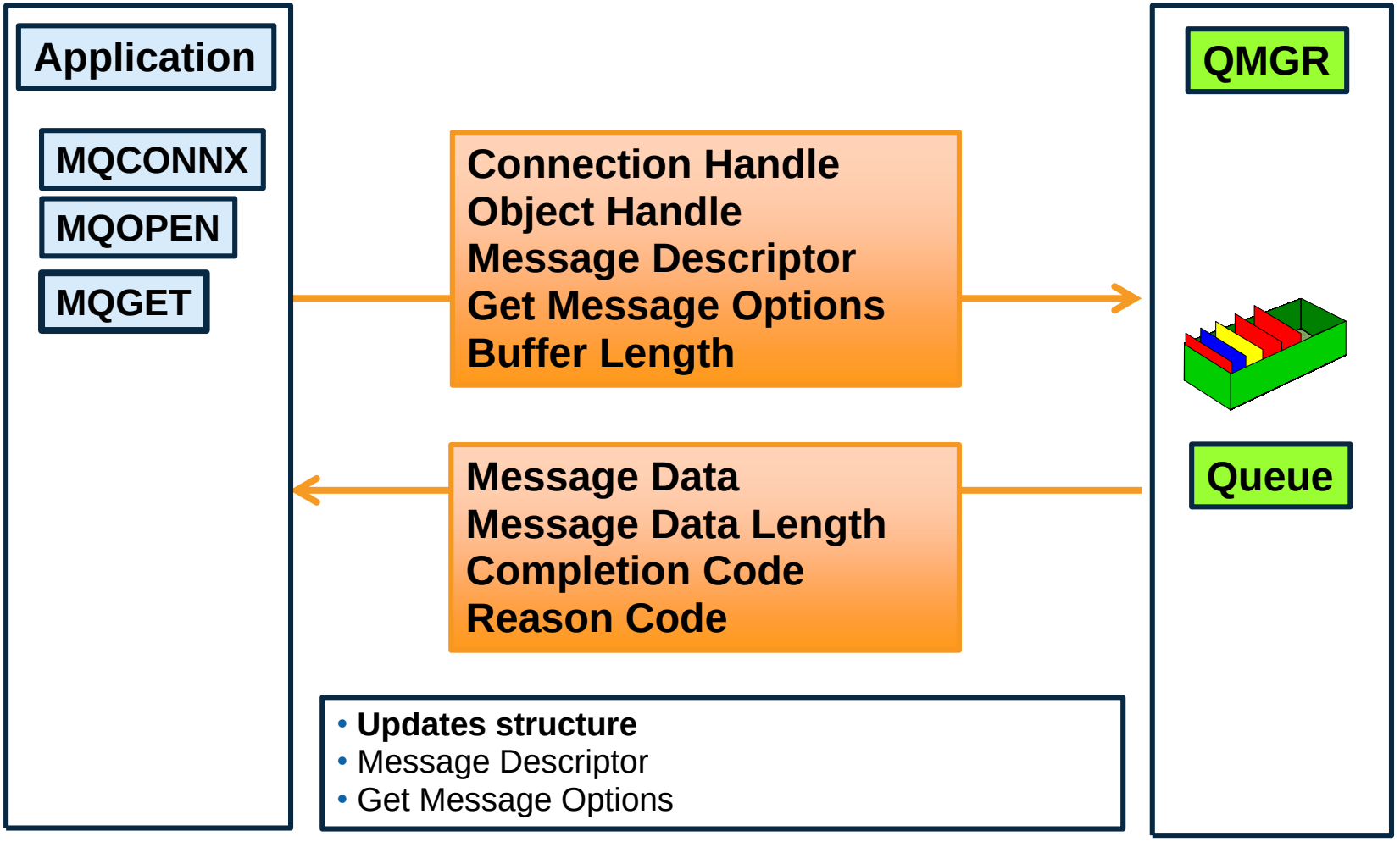
Put Options (MQPMO_*)

```
#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**

Notes

Get a message



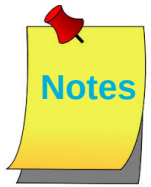
Getting Application

```
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);
```

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

Get Message Options (MQGMO) Structure

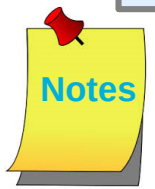
Field	Description	Version
StrucId	Structure identifier	1
Version	Structure version number	
Options	Options that control the action of MQGET	
WaitInterval	Wait Interval	
Signal1	Signal	
Signal2	Signal identifier	
ResolvedQName	Resolved name of destination queue	
MatchOptions	Options controlling selection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



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
```

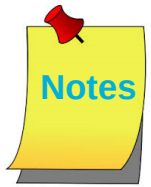
- **Options can be 'ORed' together as required**



Get Options (MQGMO_*) ..

```
#define MQGMO_CONVERT                  0x00004000
#define MQGMO_LOGICAL_ORDER            0x00008000
#define MQGMO_COMPLETE_MSG             0x00010000
#define MQGMO_ALL_MSGS_AVAILABLE       0x00020000
#define MQGMO_ALL_SEGMENTS_AVAILABLE   0x00040000
#define MQGMO_MARK_BROWSE_HANDLE       0x00100000
#define MQGMO_MARK_BROWSE_CO_OP        0x00200000
#define MQGMO_UNMARK_BROWSE_CO_OP      0x00400000
#define MQGMO_UNMARK_BROWSE_HANDLE     0x00800000
#define MQGMO_UNMARKED_BROWSE_MSG      0x01000000
#define MQGMO_PROPERTIES_FORCE_MQRFH2  0x02000000
#define MQGMO_NO_PROPERTIES            0x04000000
#define MQGMO_PROPERTIES_IN_HANDLE     0x08000000
#define MQGMO_PROPERTIES_COMPATIBILITY 0x10000000
#define MQGMO_PROPERTIES_AS_Q_DEF      0x00000000
```

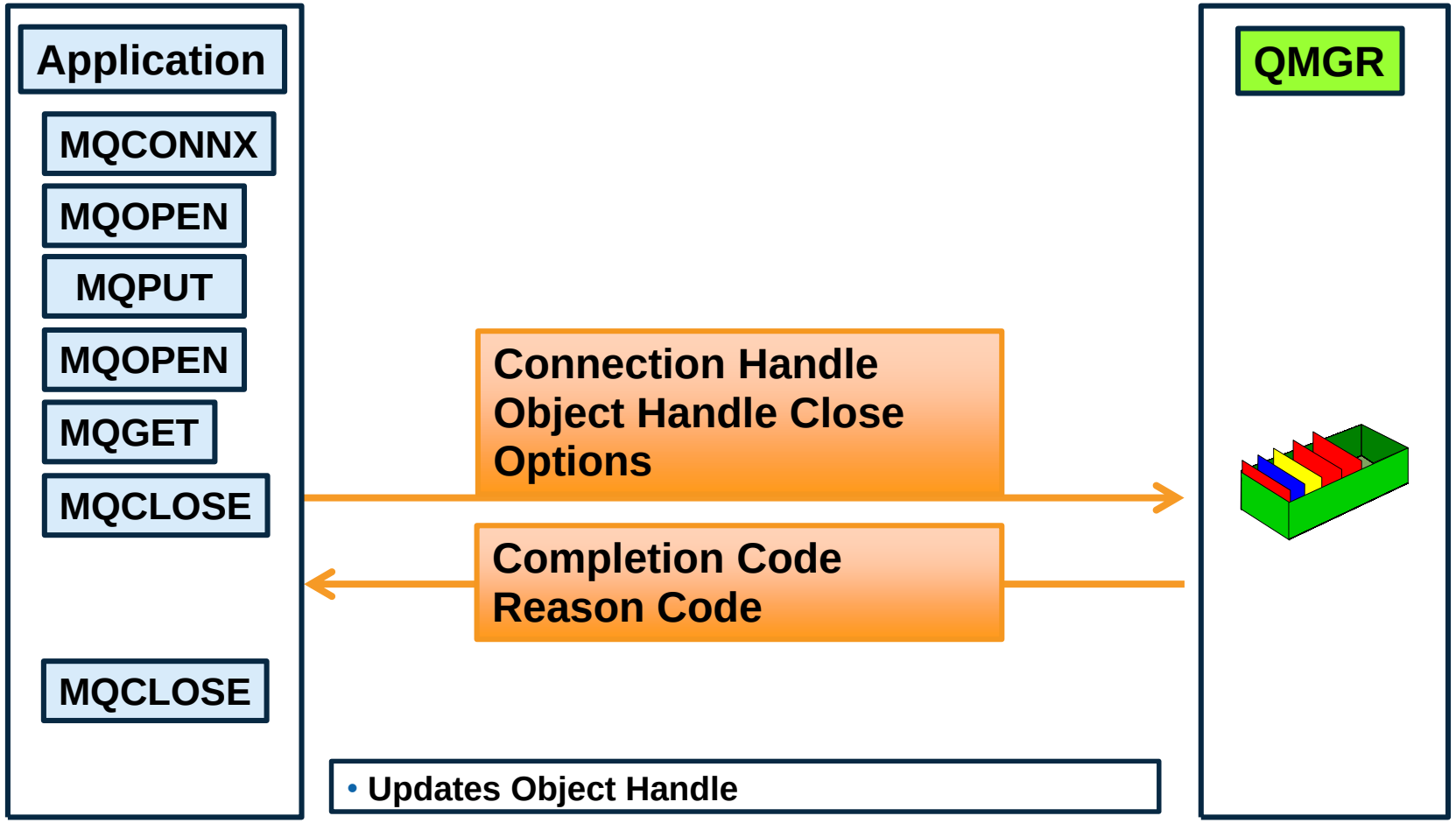
- **Options can be 'ORed' together as required**



MQGET Tips

- Avoid using default syncpoint setting
  - Defaults are not the same on z/OS (syncpoint) and Distributed (no syncpoint)
  - 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

Close a handle



Closing Application

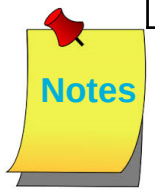
```
MQCLOSE(hConn,
        &hObj,
        MQCO_NONE,
        &CompCode,
        &Reason);
```

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

Close Options

- Options available depending on object type

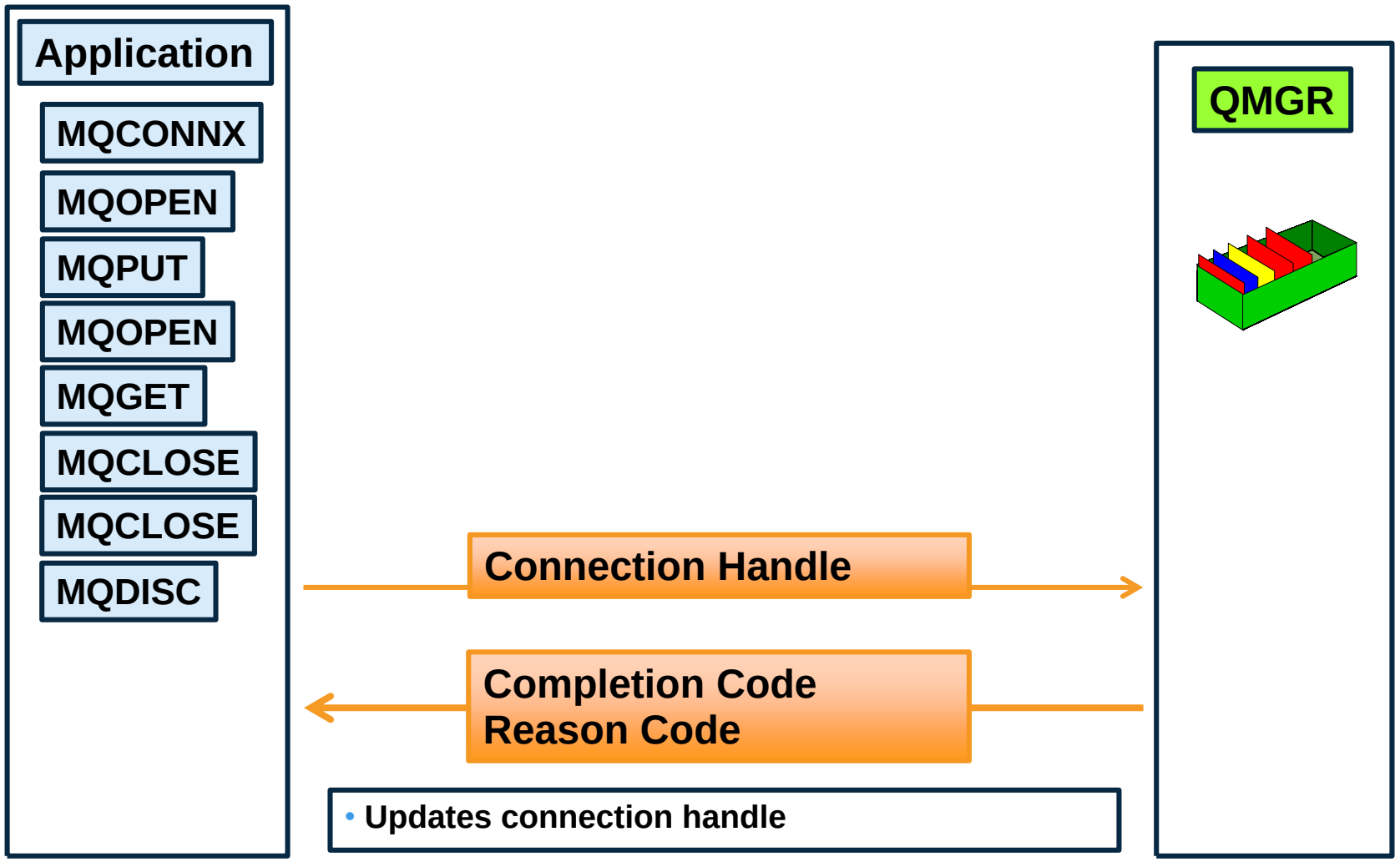
Close Option	Value	Description
MQCO_NONE	0x00000000	No optional close processing required
MQCO_IMMEDIATE	0x00000000	Any unconsumed messages in the Read Ahead Buffer are deleted and the queue is closed.
MQCO_DELETE	0x00000001	Deletes a Permanent Dynamic Queue Deletes a Temporary Dynamic Queue if it was 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.
MQCO_QUIESCE	0x00000020	Allows messages in the Read Ahead Buffer to be consumed before the queue is closed.



MQCLOSE Tips

- In triggered applications
  - Only close triggered queue if application is 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 (MQCO_QUIESCE) to avoid message loss

Disconnect from Queue Manager



Disconnecting Application

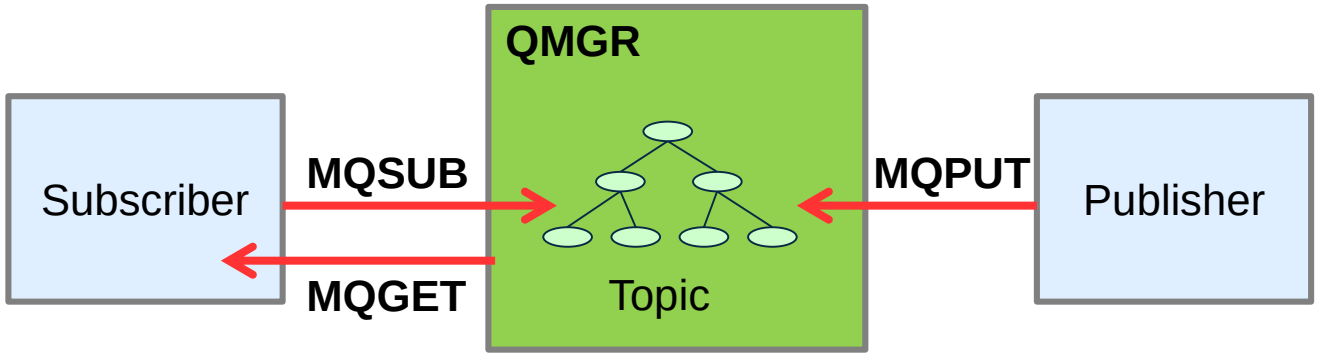
```
MQDISC(&hConn,
       &CompCode,
       &Reason);
```

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

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

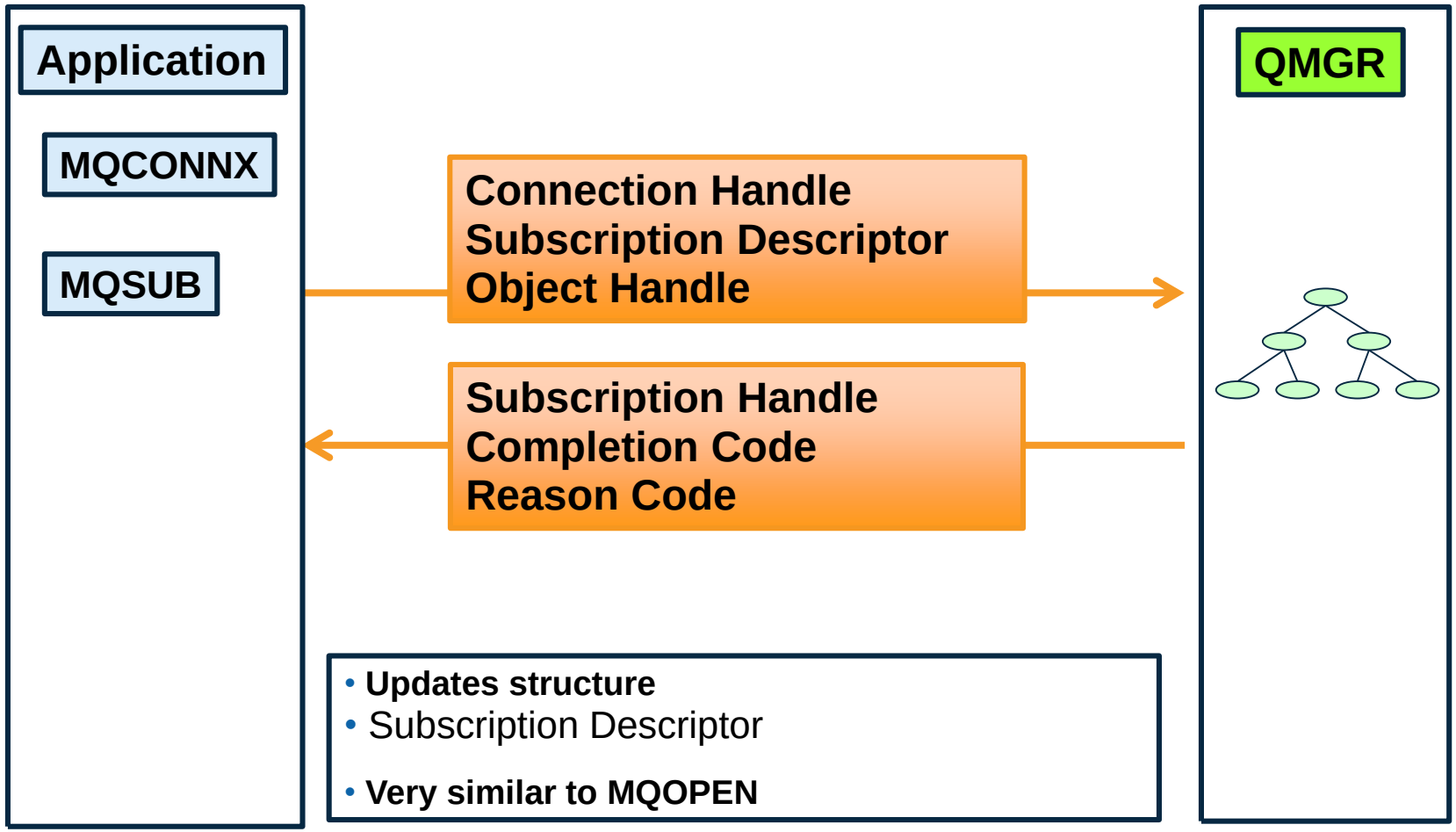
MQI – Simple verbs for Publish/Subscribe



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

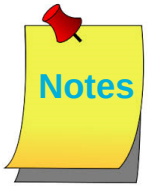
MQCONN (to QMGR)
MQOPEN (Topic)
MQPUT (publish) message to Topic
MQCLOSE (Topic)
MQDISC (from QMGR)

Susbcribe to a topic

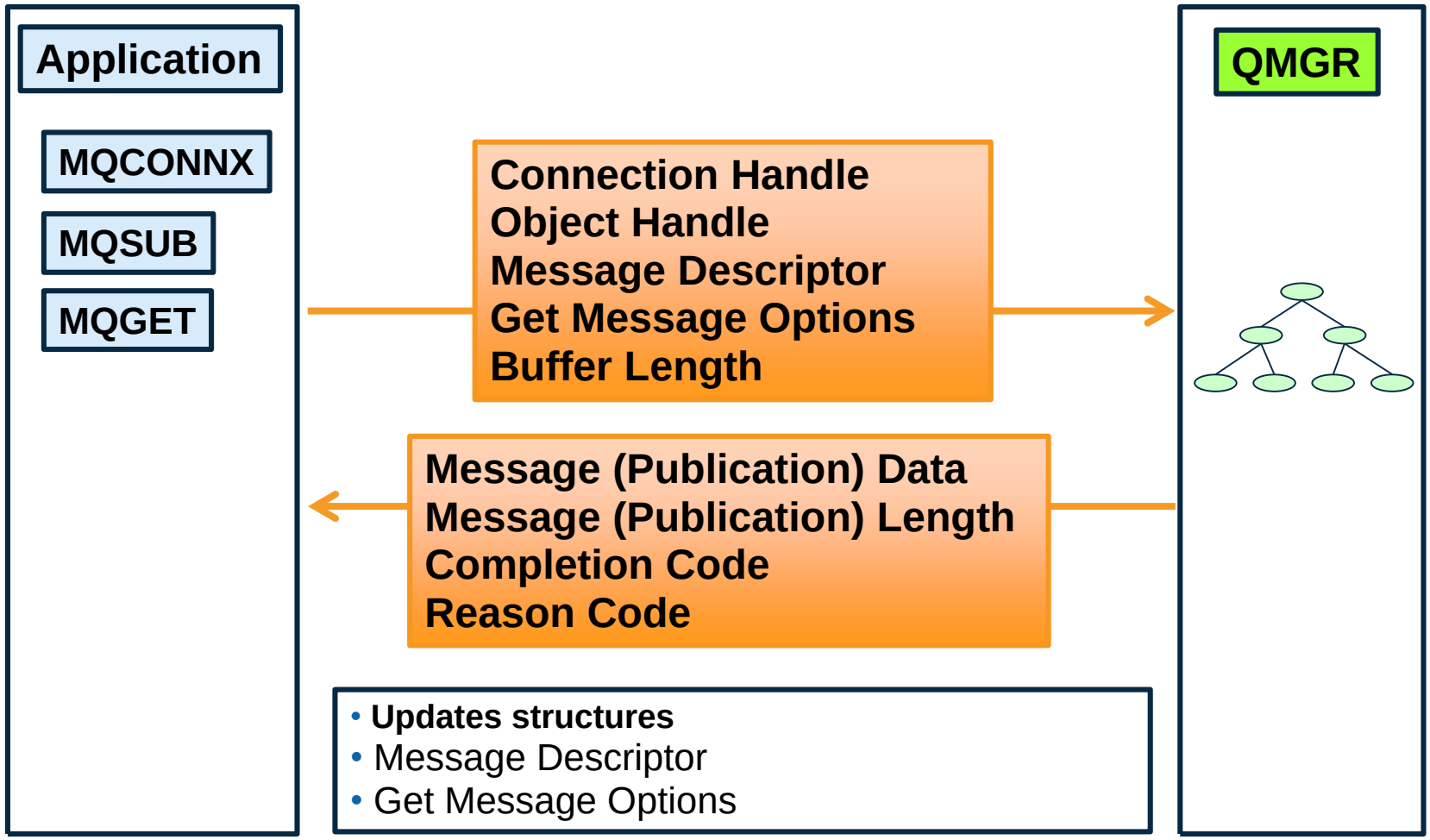


Publish Tips

- 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



Subscribing Application

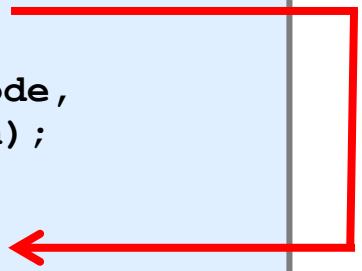
```
MQSD SubDesc = {MQSD_DEFAULT};
SubDesc.ObjectString.VSPtr    = "Price/Fruit/Apples";
SubDesc.ObjectString.VSLength = MQVS_NULL_TERMINATED;
SubDesc.Options               = MQSO_CREATE
                              | MQSO_MANAGED
                              | MQSO_FAIL_IF_QUIESCING;
```

- Subscription Descriptor (**MQSD**)
  - Describes the topic
  - MQSD.ObjectString (up to 10240 long)
  - MQSD.ObjectName (up to 48 bytes long)

  - MQSO_CREATE + MQSO_MANAGED
    - Creates a subscription with storage of messages managed by the Queue Manager

Subscribing Application …

```
MQSUB ( hQm,
        &SubDesc,
        &hObj,
        &hSub,
        &CompCode,
        &Reason);

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

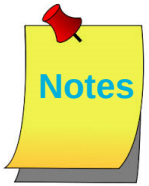


- **MQSUB** verb
  - Subscribe to a topic
- **MQGET** verb
  - Use **hObj** returned on MQSUB call
  - Consume publications
  - when **MQSO_MANAGED** used
    - Storage of msgs managed by QMGR

Subscribing Application

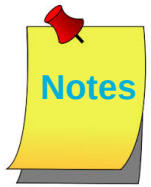
MQSUBRQ(*Hconn*, *Hsub*, *Action*, *SubRqOpts*, *Compcode*, *Reason*)
– Subscription Request

Use the MQSUBRQ call to make a request for the retained publication, when the subscriber has been registered with MQSO_PUBLICATIONS_ON_REQUEST.



Subscription Descriptor (MQSD)

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



Subscribe Options (MQSO_*)

```
#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
```

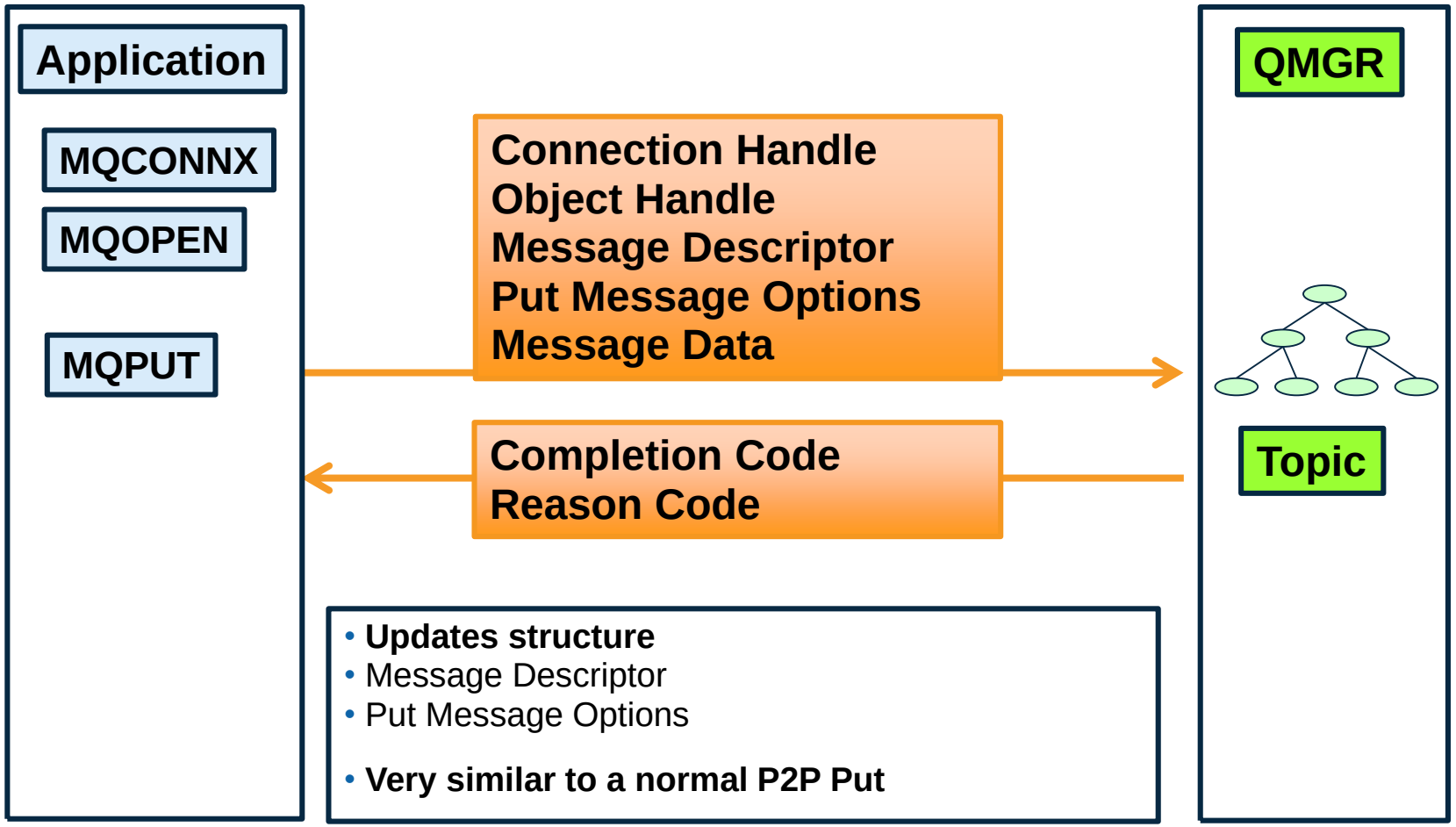
- **Options can be 'ORed' together as required**

Notes

Subscribe Tips

- Managed subscriptions 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
  - Resume will resume a previously created subscription request if one exists. If one does not exist then a new subscription request will be created.

Publish a message



Publishing Application

```
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);
```

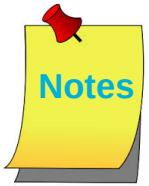
- **MQOD** describes a topic
  - ObjectType
    - MQOT_TOPIC
  - ObjectString/ObjectName
    - Topic string
- **MQOPEN** Topic
- **MQPUT** – publish msg

Publish Tips

- Choose topic string carefully
  - Use sensible topic hierarchy
    - Based on context of published data
  - Don’t use a 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’



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

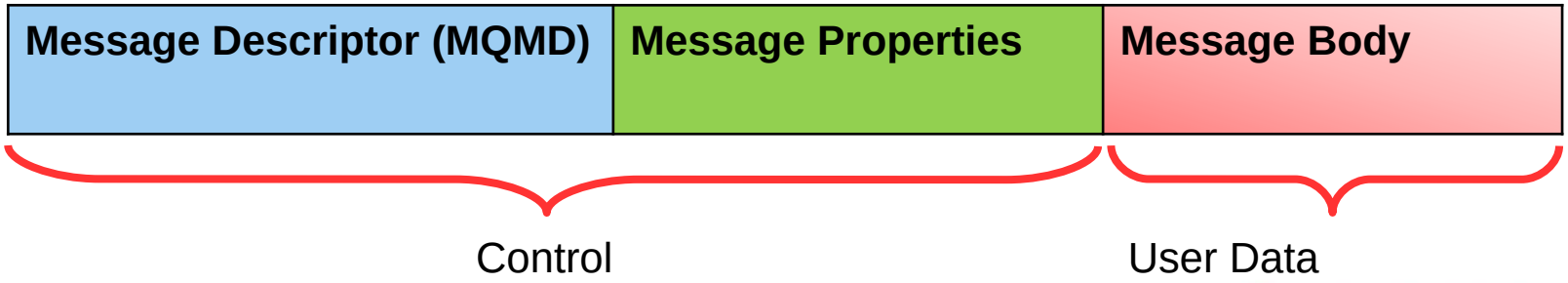
Message Properties

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

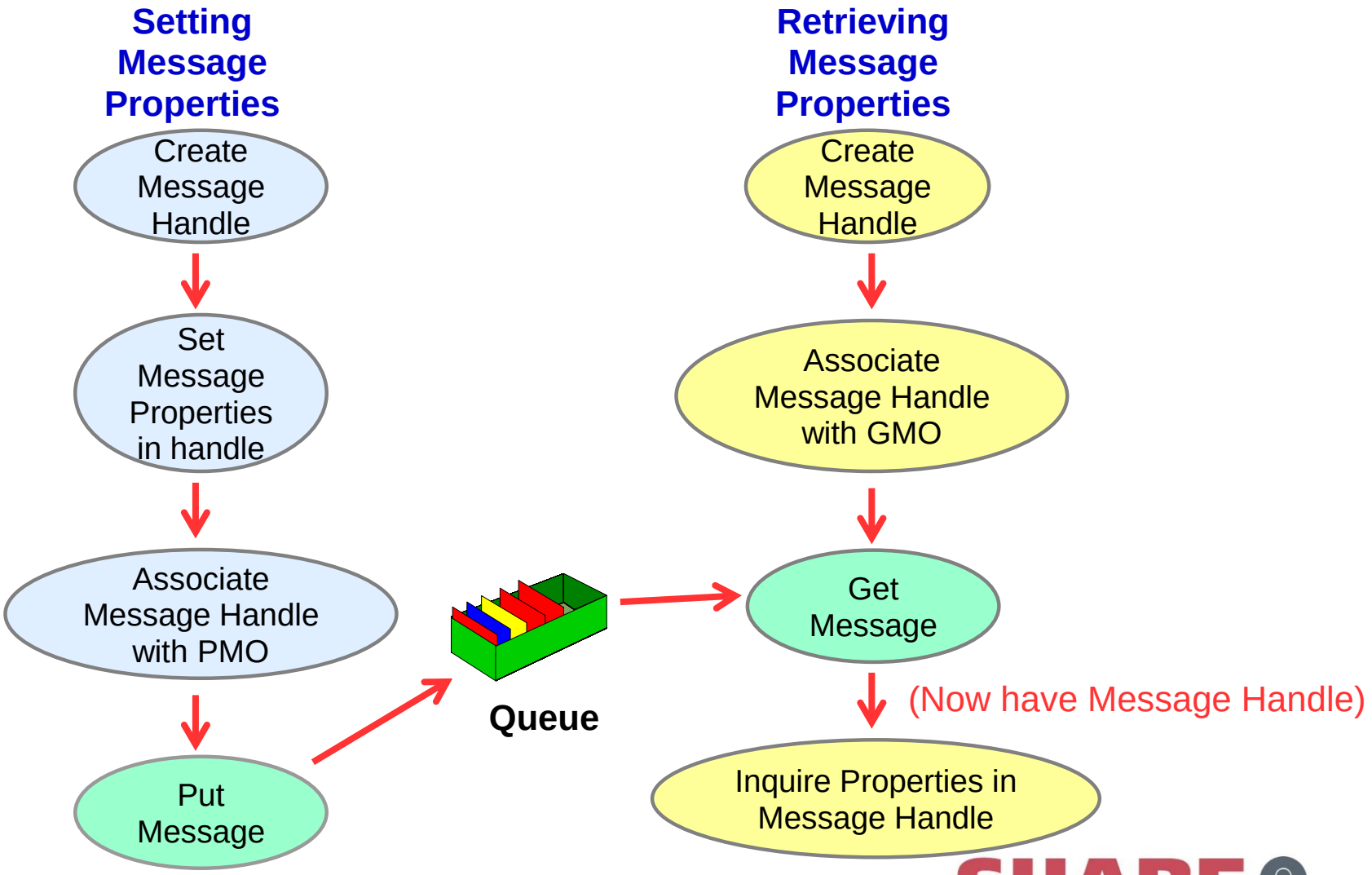
e.g.

Property	Value	Type
messageColour	Red	MQTYPE_STRING

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



Message Properties



Setting Message Properties

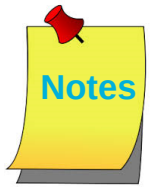
- First, create a **Message Handle**
  - Represents message
  - Can specify option to validate property names on set message property call

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

MQHMSG hMsg         = MQHM_NONE;

MQCRTMH( hconn,
         &CrtMsgHOpts,
         &hMsg,
         &CompCode,
         &Reason);
```

Message Handle

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

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

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

- 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



Setting Message Properties ..

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

```
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

```
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

```
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

```
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);
```

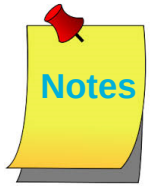
Retrieving Message Properties ..

- 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);
```

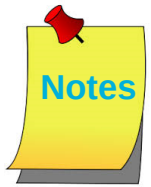
Retrieving Message Properties

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



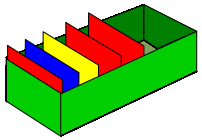
MQMHBUF and MQBUFMH

- 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



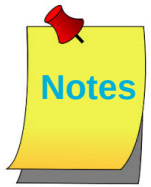
```
ObjDesc.SelectionString.VSPtr = "Colour = 'Red'";
ObjDesc.SelectionString.VSLength = MQVS_NULL_TERMINATED;
```

- MQSUB
  - Subscribing to specific publications on a topic


```
SubDesc.SelectionString.VSPtr = "City = 'Orlando'";
SubDesc.SelectionString.VSLength = MQVS_NULL_TERMINATED
```

Procedural MQI V's Object Oriented (Java): Basic connect and disconnect

```
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());
  }
}
```



Object Oriented (Java): connectAndPutMessage

```
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());
  }


}
```

```
connectAndPutMessage(){
```

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

Object Oriented (Java): Connect as a client to local QMGR

```
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());
  }

}
```

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

Object Oriented (Java): Connect to QMgr and Open Queue

```
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());
  }

}
```

```
// 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);
```

Object Oriented (Java): Create and PUT message

```
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());
  }

}
```

```
// 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);
```

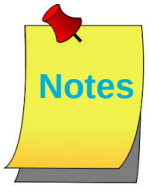
Object Oriented (Java): Close Queue and Disconnect

```
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());
  }

}
```

```
// 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();
```

Object Oriented (Java): GET and PUT1

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

// MQPUT1
queueManager.put((String) queueName, (MQMessage) msg);
```

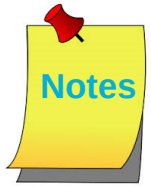


Object Oriented (Java): Subscribing to a Topic

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

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

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

// MQSUB
MQTopic topic = queueManager.accessTopic(topicString, topicObject, openAs, openOptionsForGet);
```

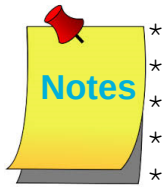
Note: There are other Java classes and methods

See: http://www-01.ibm.com/support/knowledgecenter/SSFKSJ_7.5.0/com.ibm.mq.javadoc.doc/WMQJavaClasses/com/ibm/mq/MQQueueManager.html



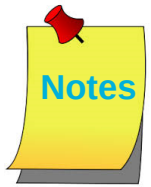
z/OS Batch COBOL: IDENTIFICATION DIVISION

```
CBL NODYNAM,LIB,OBJECT,RENT,RES,APOST
      *                                                               *
      * ------------------------------------------------------------- *
       IDENTIFICATION DIVISION.
      * ------------------------------------------------------------- *
       PROGRAM-ID. CSQ4BVK1.
      *REMARKS
      *****************************************************************
      * <copyright                                                    *
      * notice="lm-source"                                            *
      * pids="5655-W97"                                               *
      * years="1993,2014"                                             *
      * crc="2011116717" >                                            *
      * Licensed Materials - Property of IBM                          *
      *                                                               *
      * 5655-W97                                                      *
      *                                                               *
      * (C) Copyright IBM Corp. 1993, 2014 All Rights Reserved.       *
      * </copyright>                                                  *
      *****************************************************************
      *                                                               *
      *   IBM MQ for z/OS                                             *
      *                                                               *
      *   Module Name       : CSQ4BVK1                                *
      *                                                               *
      *   Environment       : z/OS Batch; COBOL II                    *
      *                                                               *
      *   Description       : Sample program to put a number of       *
      *                       messages to a queue.                    *
      *                                                               *
      *   Limitation        : Maximum message length set at 65535.    *
      *                                                               *
      *****************************************************************
```



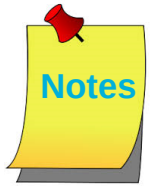
z/OS Batch COBOL: WORKING STORAGE SECTION

```
* ---------------------------------------------------------------- *
 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'.
```



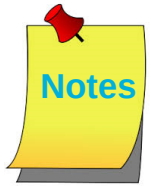
z/OS Batch COBOL: WORKING STORAGE + LINKAGE

```
*
*    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.
* ---------------------------------------------------------------- *
 01  PARMDATA.
     05  PARM-LEN                 PIC S9(03) BINARY.
     05  PARM-STRING              PIC X(100).
```



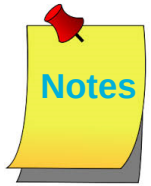
z/OS Batch COBOL: PROCEDURE DIVISION + PARAMETERS

```
* ---------------------------------------------------------------- *
 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.
*
```



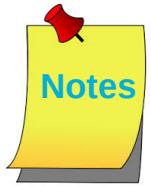
z/OS Batch COBOL: PARAMETERS + MSG BUFFER

```
*
*     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.
*
```



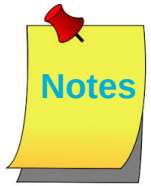
z/OS Batch COBOL: MQCONN call

```
*
*
*     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'.
*
```



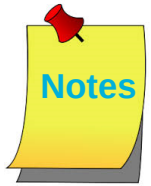
z/OS Batch COBOL: MQOPEN call

```
*
*    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'.
```



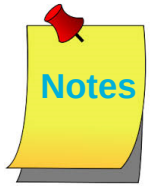
z/OS Batch COBOL: Prepare for MQPUT

```
*
*     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.
*
```



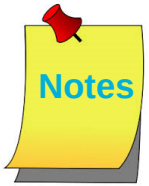
z/OS Batch COBOL: MQPUT calls

```
*
*     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
          END-IF
*
      END-PERFORM.
```



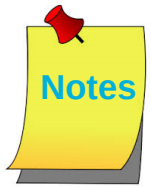
z/OS Batch COBOL: Display Msgs Put and MQCLOSE call


```
*
*     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.
*
```



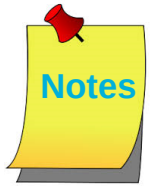
z/OS Batch COBOL: MQDISC call + Program End

```
 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
```



z/OS Batch COBOL: ERROR MESSAGE SECTIONS

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

Notes

Summary

- Simple MQI – very easy to get started with
  - Let most fields retain default values
  - Keep things simple if you can
- Check reason codes, and log any failures
  - MQ trace can be useful
- 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)
    - MQGEM provide graphical tools for issuing MQI
      - MO71 has an API Exerciser
    - Supportpac MA01 – q.exe (a powerful program for putting/getting messages)
    - MQExplorer has put/browse/clear message function

Any Questions ?

Please complete your session evaluation .. Thank You !

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

Copyright and Trademarks