

Introduction to the MQI

Morag Hughson – morag@mqgem.com

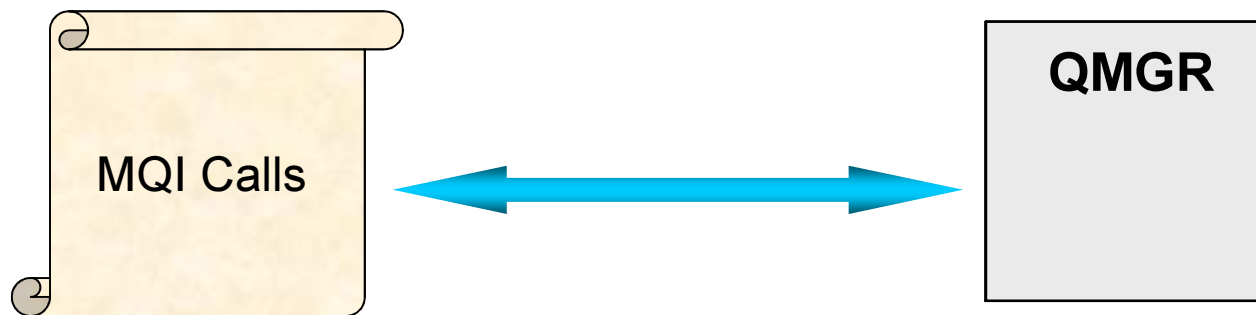
MQGem Software

COBOL

C

Agenda

- **MQI Concepts**
- **MQI Structures & Datatypes**
- **Basic MQI walkthrough**
 - ▶ With Demonstrations
 - ▶ A number of verbs we do not cover
 - MQCMIT, MQBACK, MQINQ, MQSET etc



Languages

■ Procedural (MQI)

- ▶ C
- ▶ COBOL
- ▶ Visual Basic
- ▶ RPG
- ▶ PL/1
- ▶ Assembler
- ▶ TAL

■ Object-Oriented (Classes)

- ▶ Java
- ▶ JMS/XMS
- ▶ C++
- ▶ .NET languages
- ▶ ActiveX (MQAX)
- ▶ Perl

Interface

- **Simple 'handle' based interface**
 - ▶ Returned handle passed to subsequent call

- **Each verb returns**

- ▶ **Completion Code**

- MQCC_OK 0
 - MQCC_WARNING 1
 - MQCC_FAILED 2

- ▶ **Reason Code**

- MQRC_XXXXXXXX 2xxx
 - MQRC_NONE 0

- **Make sure you check the reason codes!**



```
C:\>nqrc 2085
      2085 0x00000825 MQRC_UNKNOWN_OBJECT_NAME
C:\>_
```

Data Structures

- Programmers should be familiar with:

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

Data Structure Tips

- **Use structure initialisers**

- ▶ `MQMD md = { MQMD_DEFAULT };`
- ▶ Initialise to version 1

- **Structures are versioned**

- ▶ Set the minimum version you need
 - `md.Version = 2;`
- ▶ Don't use current version
 - `md.Version = MQMD_CURRENT_VERSION;`

- **Bear in mind that some structures are input/output**

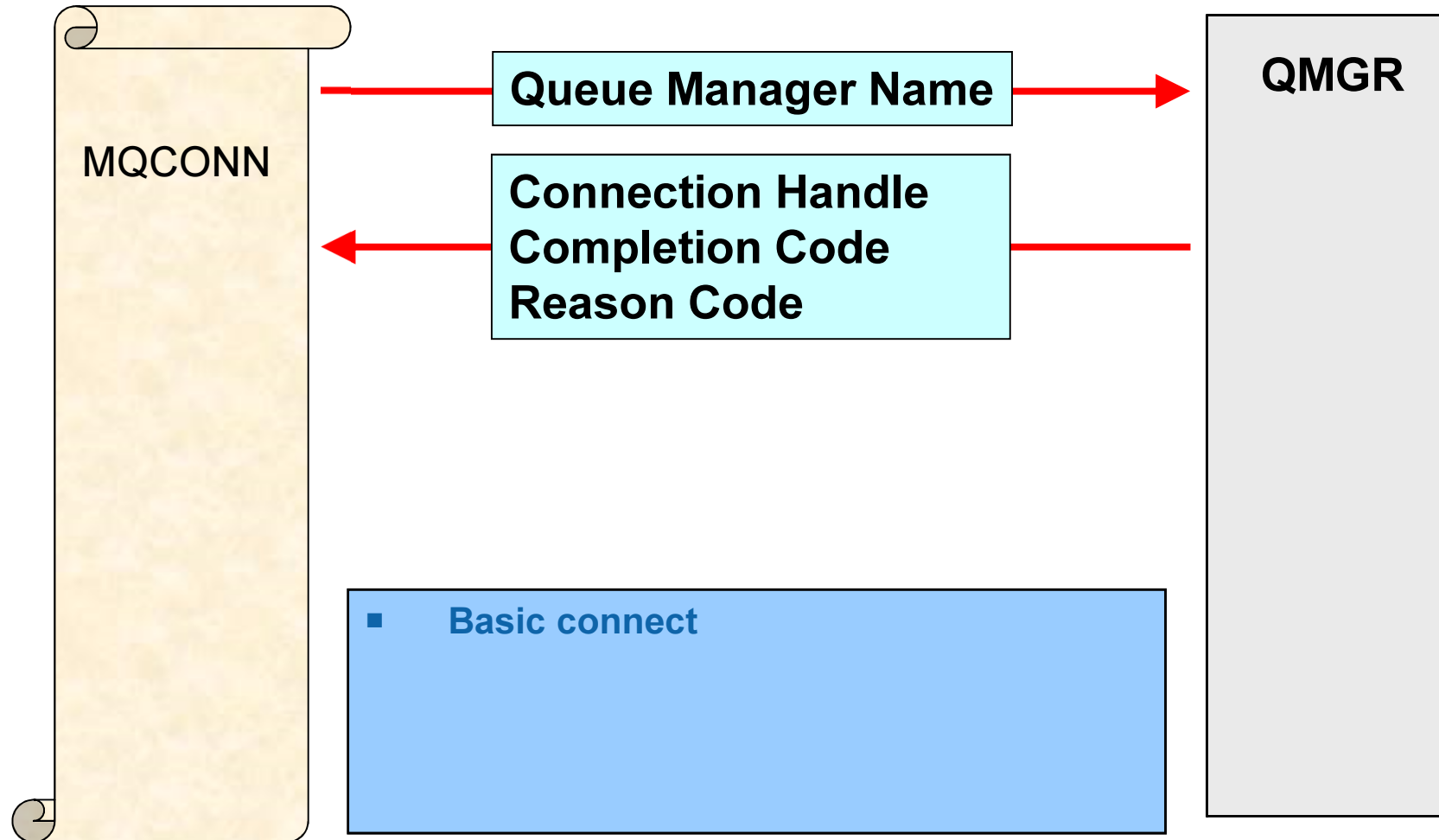
- ▶ May need to reset values for subsequent call
 - Eg. `MsgId` & `CorrelId` fields of `MQMD` on `MQGET` call

MQ Elementary Data Types

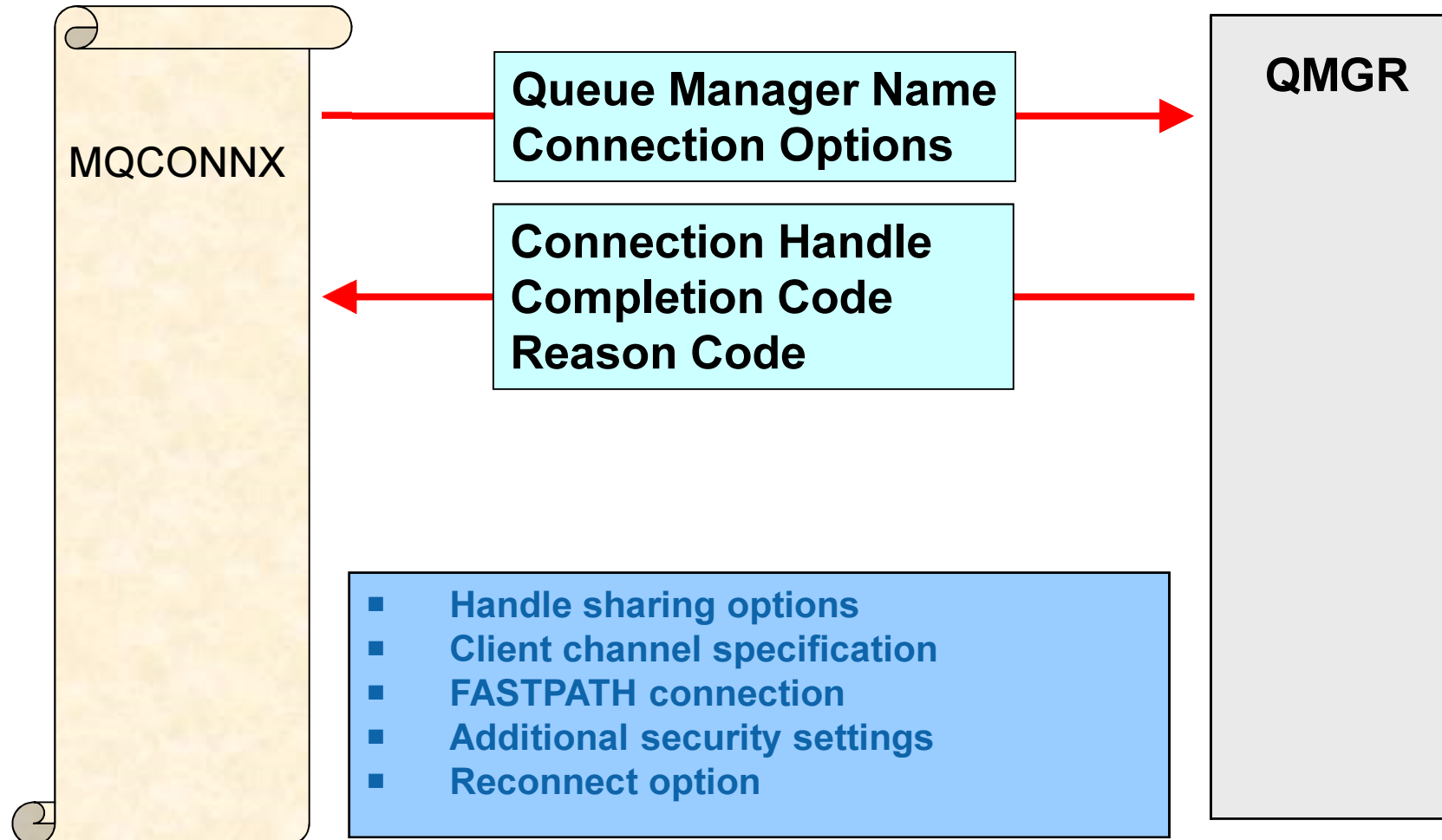
- The main MQI data types

Data Type	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



Connect with extended options



Connecting

■ MQCONN

- ▶ Don't hardcode QM name
- ▶ Always check reason codes

■ Connections options

- ▶ Connection not thread specific
- ▶ Client reconnect

```
MQCONN( Qm,
        &cno,
        &hQm,
        &CompCode,
        &Reason);

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

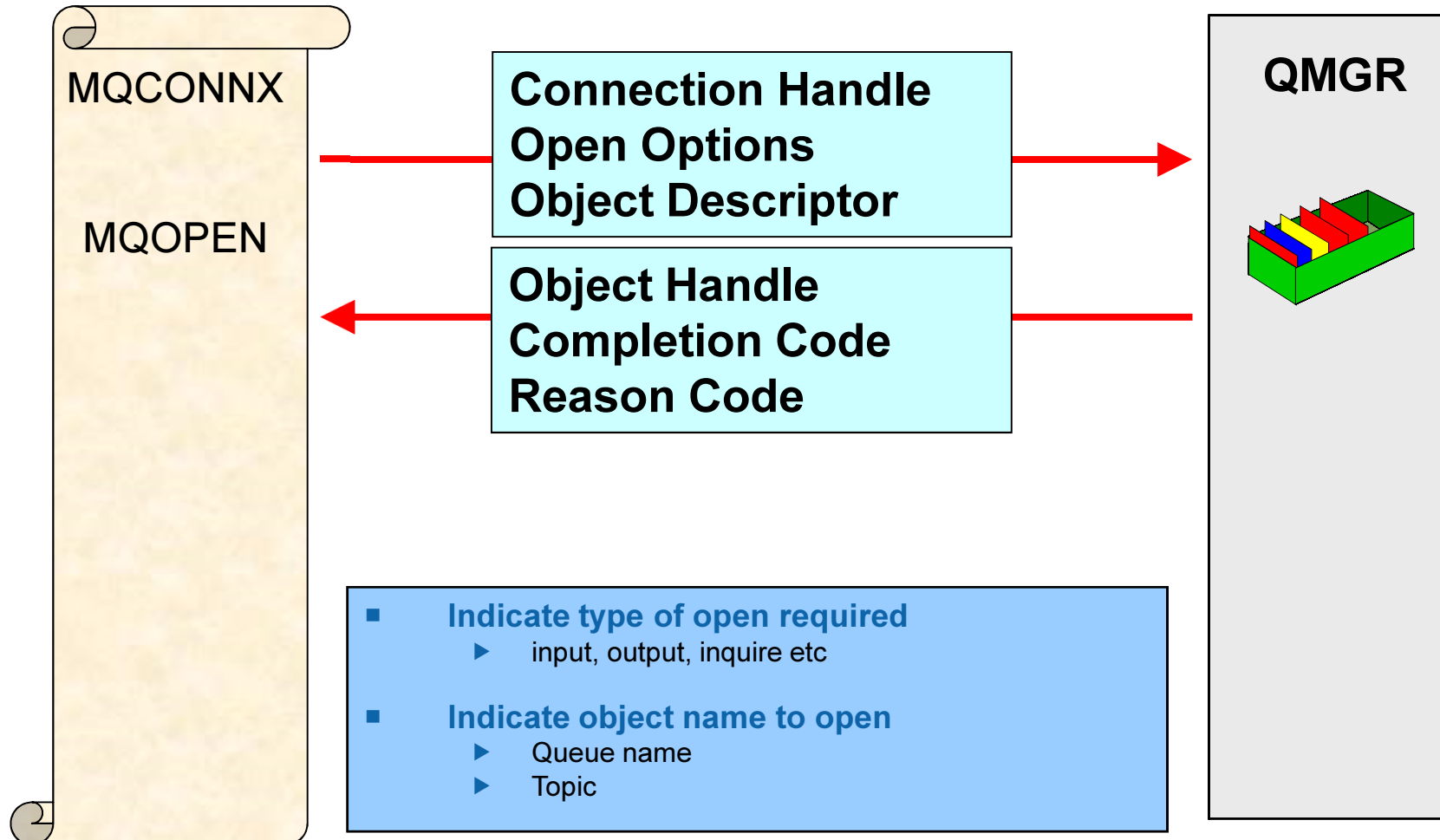
```
MQHCONN hQm = MQHC_UNUSABLE_HCONN;
MQCHAR48 Qm = "QM1";
MQCNO cno = {MQCNO_DEFAULT};

cno.Options |= MQCNO_HANDLE_SHARE_BLOCK |
              MQCNO_RECONNECT;
```

MQCONN(X) Tips

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

Open a Queue



Open a queue

- **MQOPEN** a queue
- **OpenOptions**
 - ▶ MQOO_ flags which are required
- **MQOD** describes a object to open
 - ▶ ObjectType
 - MQOT_Q for point-to-point
 - MQOT_TOPIC for publish
 - ▶ ObjectString/ObjectName

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

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

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

Object Descriptor (MQOD)

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	3
AlternateSecurityId	Alternate security identifier	
ResolvedQName	Resolved queue name	
ResolvedQMGrName	Resolved queue manager name	4
ObjectString	Long object name	
SelectionString	Selection string	
ResObjectString	Resolved long object name	
ResolvedType	Resolved object type	

Open Options

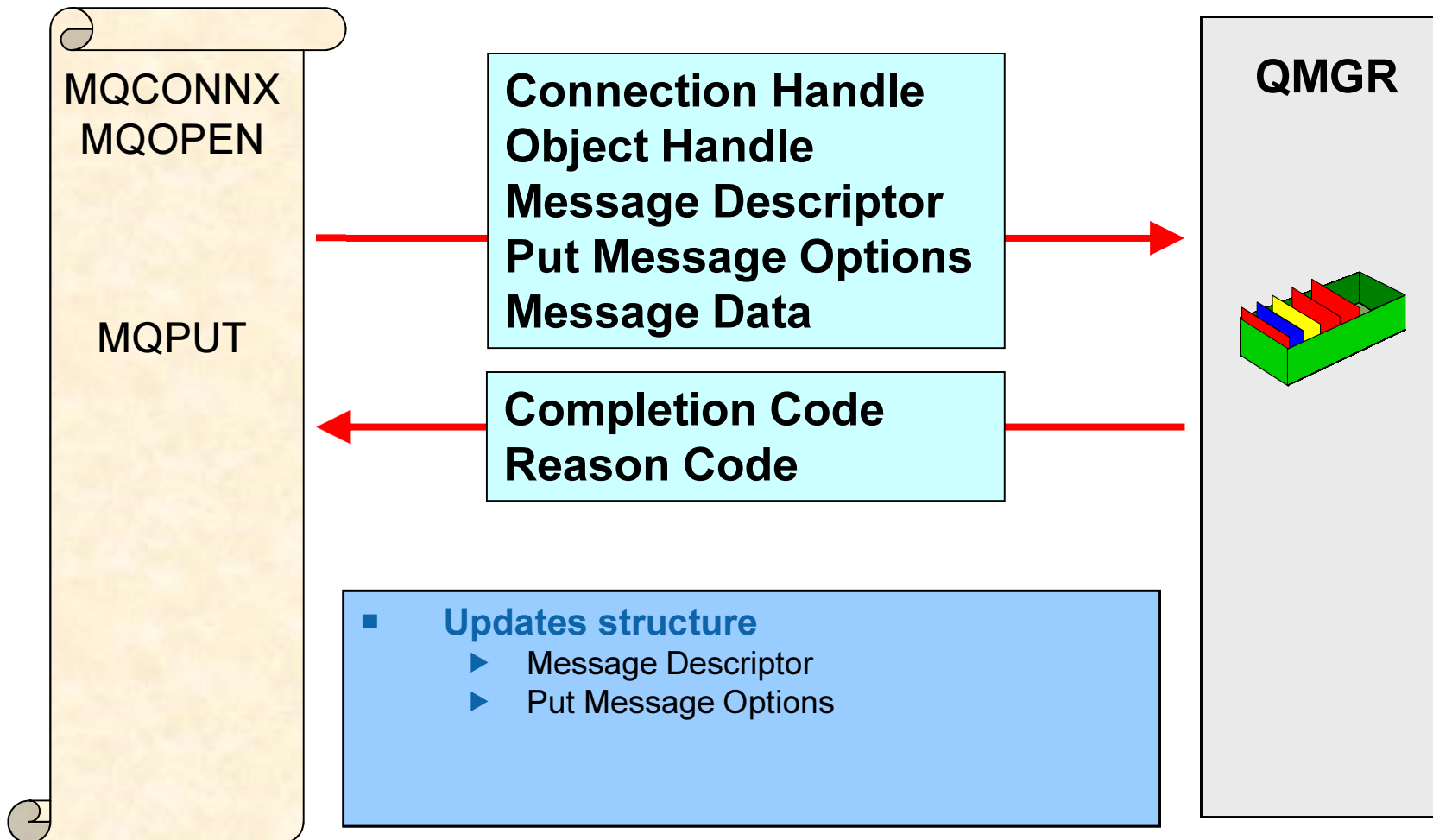
```
#define MQOO_BIND_AS_Q_DEF          0x00000000
#define MQOO_READ_AHEAD_AS_Q_DEF   0x00000000
#define MQOO_INPUT_AS_Q_DEF        0x00000001
#define MQOO_INPUT_SHARED           0x00000002
#define MQOO_INPUT_EXCLUSIVE        0x00000004
#define MQOO_BROWSE                 0x00000008
#define MQOO_OUTPUT                 0x00000010
#define MQOO_INQUIRE               0x00000020
#define MQOO_SET                   0x00000040
#define MQOO_SAVE_ALL_CONTEXT       0x00000080
#define MQOO_PASS_IDENTITY_CONTEXT 0x00000100
#define MQOO_PASS_ALL_CONTEXT       0x00000200
#define MQOO_SET_IDENTITY_CONTEXT   0x00000400
#define MQOO_SET_ALL_CONTEXT        0x00000800
#define MQOO_ALTERNATE_USER_AUTHORITY 0x00001000
#define MQOO_FAIL_IF QUIESCING     0x00002000
#define MQOO_BIND_ON_OPEN           0x00004000
#define MQOO_BIND_NOT_FIXED         0x00008000
#define MQOO_CO_OP                  0x00020000
#define MQOO_NO_READ_AHEAD          0x00080000
#define MQOO_READ_AHEAD             0x00100000
```

- Options can be 'ORed' together as required

MQOPEN Tips

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

Put a message



Putting Application

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

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

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

```
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;
```

Message Descriptor (MQMD)

Field (V1)	Description
StrucId	Structure identifier
Version	Structure version number
Report	Options for report messages
MsgType	Message Type
Expiry	Message lifetime
Feedback	Feedback or reason code
Encoding	Numeric encoding of message data
CodedCharSetId	Character set identifier of message data
Format	Format name of message data
Priority	Message priority
Persistence	Message persistence
MsgId	Message identifier
CorrelId	Correlation identifier
BackoutCount	Backout counter
ReplyToQ	Name of reply queue
ReplyToQMgr	Name of reply queue manager
UserIdentifier	User identifier
AccountingToken	Accounting token
AppIdentityData	Application data relating to identity
PutAppType	Type of application that put the message
PutAppName	Name of application that put the message
PutDate	Date when message was put
PutTime	Time when message was put
AppOriginData	Application data relating to origin

Field (V2)	Description
GroupId	Group identifier
MsgSeqNumber	Sequence number of logical message within group
Offset	Offset of data in physical message from start of logical message
MsgFlags	Message flags
OriginalLength	Length of original message

Put Message Options (MQPMO)

Field	Description	Version
StrucId	Structure identifier	1
Version	Structure version number	
Options	Options that control the action of MQPUT and MQPUT1	
Context	Object handle of input queue	
KnownDestCount	Number of messages sent successfully to local queues	
UnknownDestCount	Number of messages sent successfully to remote queue	
InvalidDestCount	Number of messages that could not be sent	
ResolvedQName	Resolved name of destination queue	
ResolvedQMGrName	Resolved name of destination queue manager	
RecsPresent	Number of put messages records or response records present	2
PutMsgRecFields	Flags indicating which MQPMR fields are present	
PutMsgRecOffset	Offset of first put-message records from start of MQPMO	
ResponseRecOffset	Offset of first response record from start of MQPMO	
PutMsgRecPtr	Address of first put message record	
ResponseRecPtr	Address of first response record	3
OriginalMsgHandle	Original message handle	
NewMsgHandle	New message handle	
Action	Type of put being performed and the relationship between the original message and the new message	
PubLevel	Level of subscription targeted by the publication	

Put Options

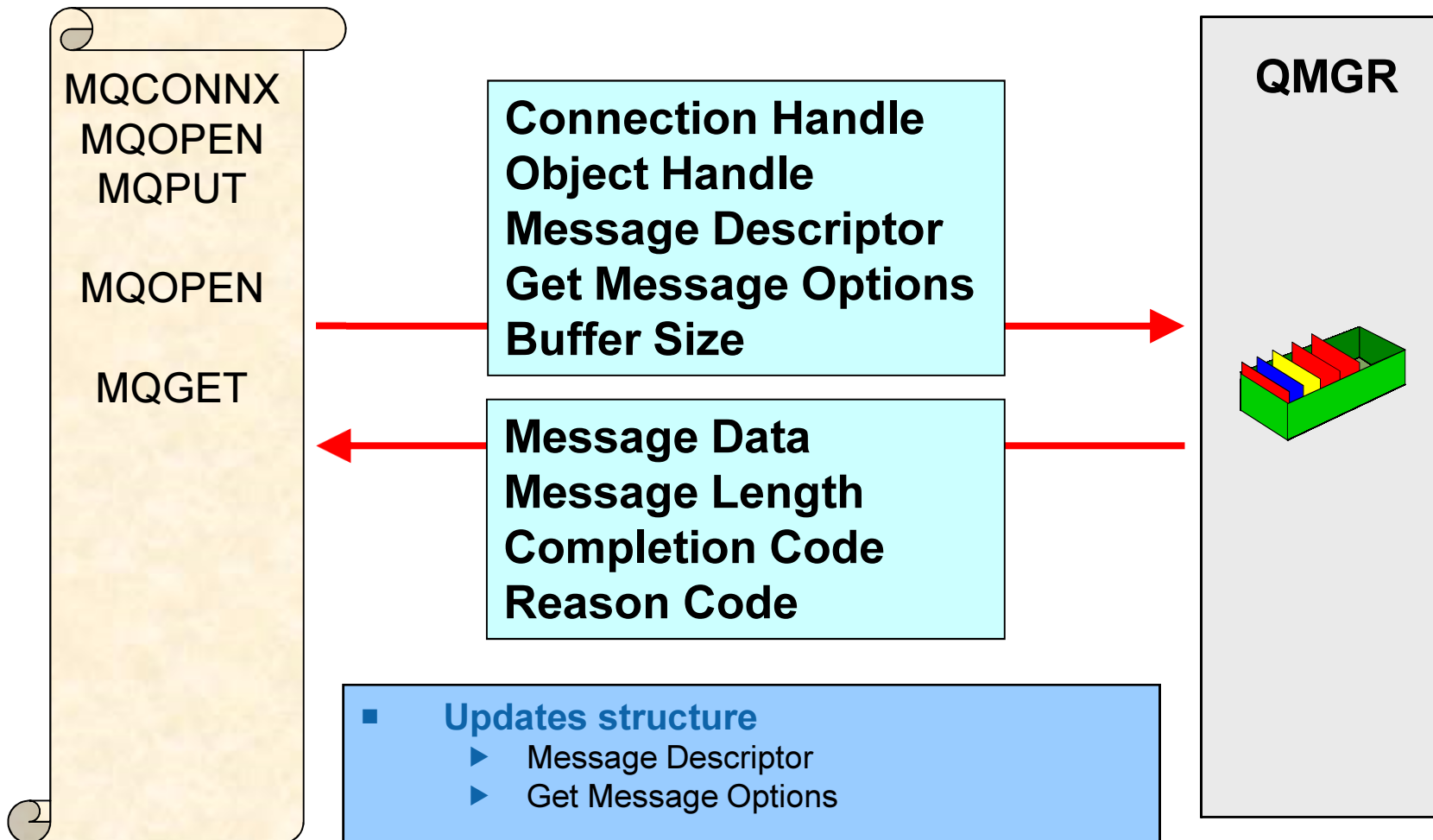
```
#define MQPMO_SYNCPOINT 0x00000002
#define MQPMO_NO_SYNCPOINT 0x00000004
#define MQPMO_DEFAULT_CONTEXT 0x00000020
#define MQPMO_NEW_MSG_ID 0x00000040
#define MQPMO_NEW_CORREL_ID 0x00000080
#define MQPMO_PASS_IDENTITY_CONTEXT 0x00000100
#define MQPMO_PASS_ALL_CONTEXT 0x00000200
#define MQPMO_SET_IDENTITY_CONTEXT 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

MQPUT Tips

- **Always use explicit syncpoint setting**
 - ▶ Defaults are not the same on z/OS and Distributed
 - ▶ Generally
 - Syncpoint when persistent
 - No syncpoint when non-persistent
- **Try not to use extreme message sizes**
 - ▶ QM optimized for message 4K – 1MB
- **Consider async put response for performance gain**
 - ▶ If on client and sending many non-persistent messages

Get a message



Getting Application

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

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

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

```
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;
```


Get Message Options (MQGMO)

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	3
MsgToken	Message token	
ReturnedLength	Length of message data returned (bytes)	4
MsgHandle	The handle to a message that is to be populated with the properties of the message being retrieved from the queue.	

Get Options

```
#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
```

```
#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
```

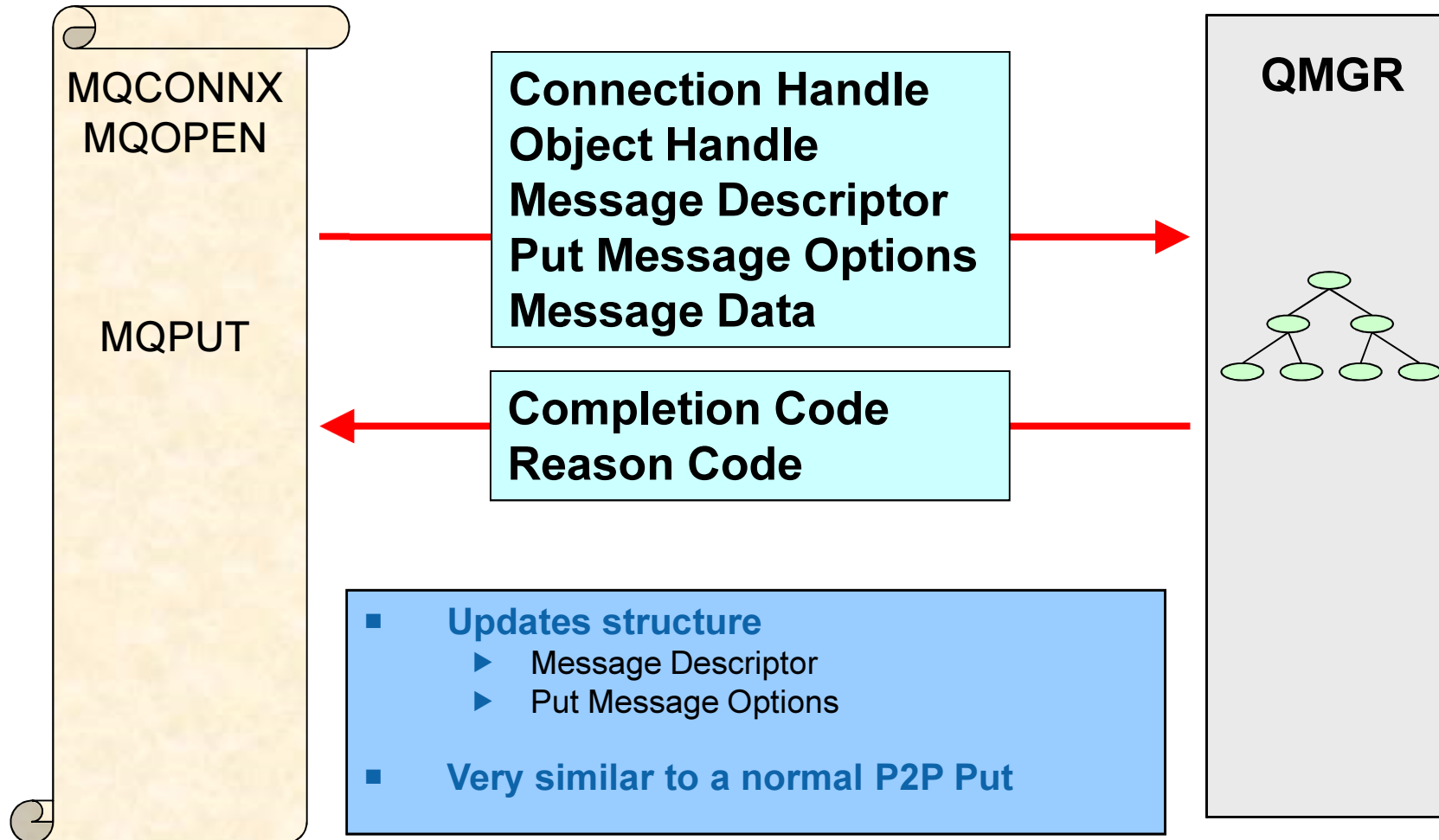
```
#define MQGMO_PROPERTIES_AS_Q_DEF 0x00000000
```

Options can be 'ORed' together as required

MQGET Tips

- **Avoid using default syncpoint setting**
 - ▶ Defaults are not the same on z/OS and Distributed
 - ▶ Generally
 - MQGMO_SYNCPOINT_IF_PERSISTENT
- **Use MQGMO_FAIL_IF QUIESCING**
 - ▶ Ensure your application ends promptly
- **Generally use MQGMO_CONVERT**
 - ▶ Even if you 'think' you don't need it
- **Remember to reset Msgld & Correlld fields**
 - ▶ These fields are used for selection and are returned
- **Handle 'poison message'**
 - ▶ Look at BackoutCount in MQMD
- **Consider using MQCB to consume messages instead**
 - ▶ Callback semantics, often easier to code

Publish a message



Publishing Application

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

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

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

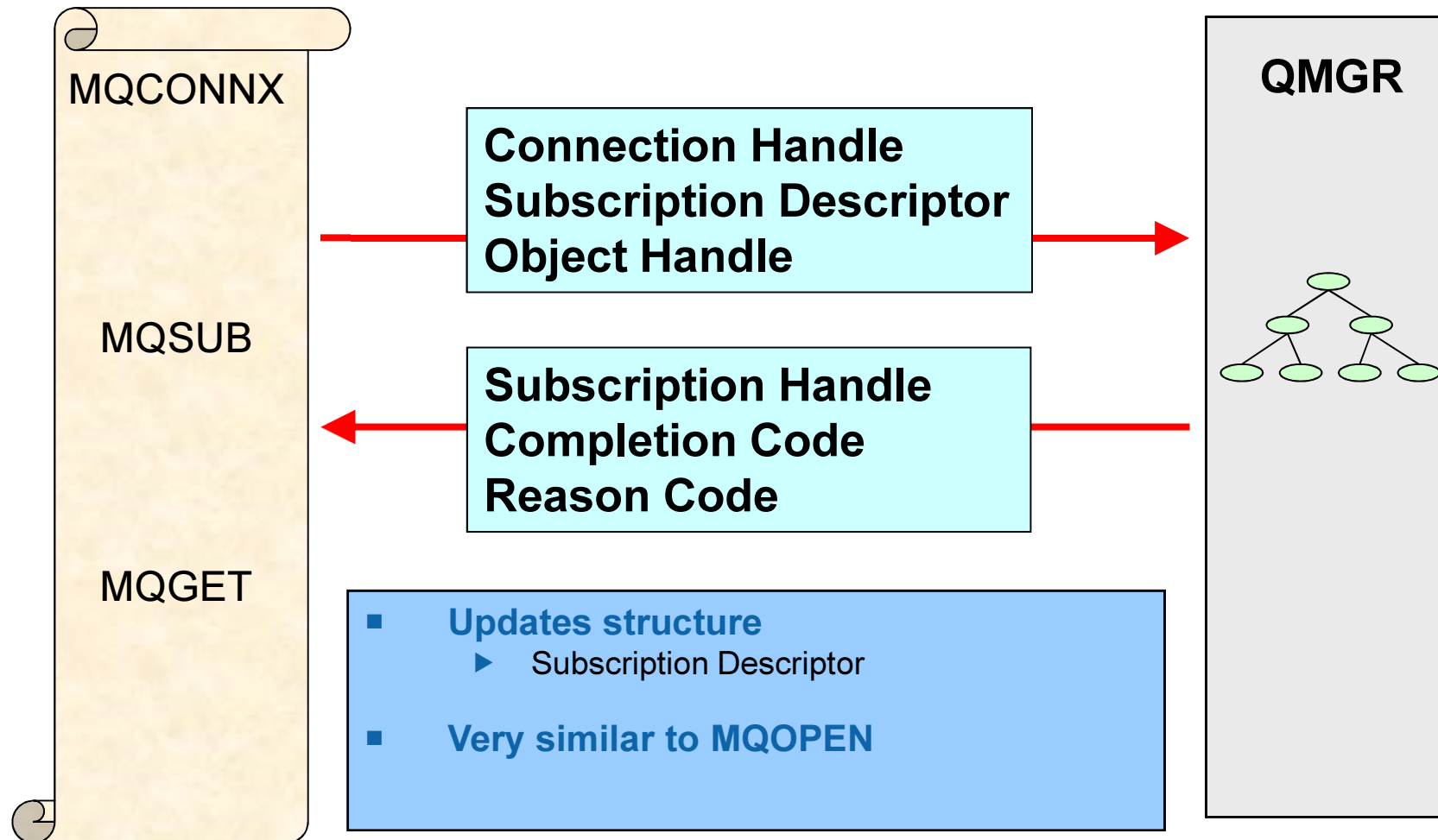
```
MQOD    ObjDesc = {MQOD_DEFAULT};

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

Publishing Tips

- **Choose topic string carefully**
 - ▶ Use sensible topic hierarchy
 - Based on context of published data
 - ▶ Don't use different topic for each publish
 - This is probably meta data, use message property
 - ▶ Topic strings can be up to 10K bytes
 - But don't use long topics unless necessary
- **Consider using Topic object and Topic string**
 - ▶ Administer can set point in topic tree
 - Known as 'topic tree isolation'

Subscribe to a topic



Subscribing Application

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

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

```
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)

Field	Description
StructId	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

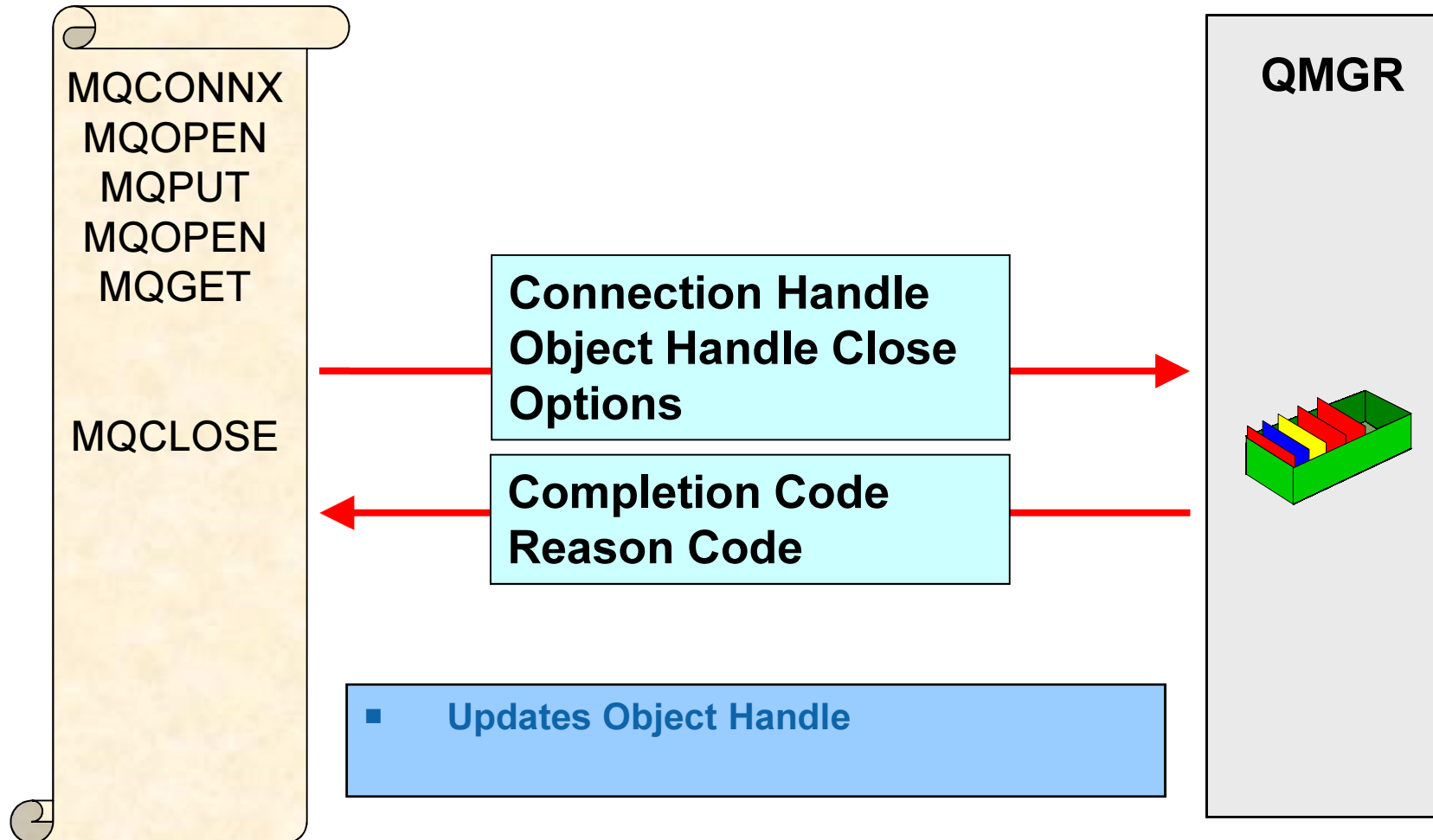
```
#define MQSO_NON_DURABLE          0x00000000
#define MQSO_READ_AHEAD_AS_Q_DEF 0x00000000
#define MQSO_ALTER                0x00000001
#define MQSO_CREATE              0x00000002
#define MQSO_RESUME              0x00000004
#define MQSO_DURABLE             0x00000008
#define MQSO_GROUP_SUB          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
```

- Options can be 'ORed' together as required

Subscribing Tips

- **Managed handles make things simpler**
- **Only use durable subscriptions when necessary**
 - ▶ Avoid build up of messages
- **For durable subscriptions**
 - ▶ Combine the create and resume options to make it simpler

Close a handle



Closing Application

- MQOPEN a queue
- MQCLOSE a queue
 - ▶ Normally we'd do something !
 - ▶ Note address of MQHOBJ

```
OpnOpts = MQOO_INPUT_SHARED
          | MQOO_FAIL_IF_QUIESCING;

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

< Issue some MQI calls here >

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

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

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

Close Options

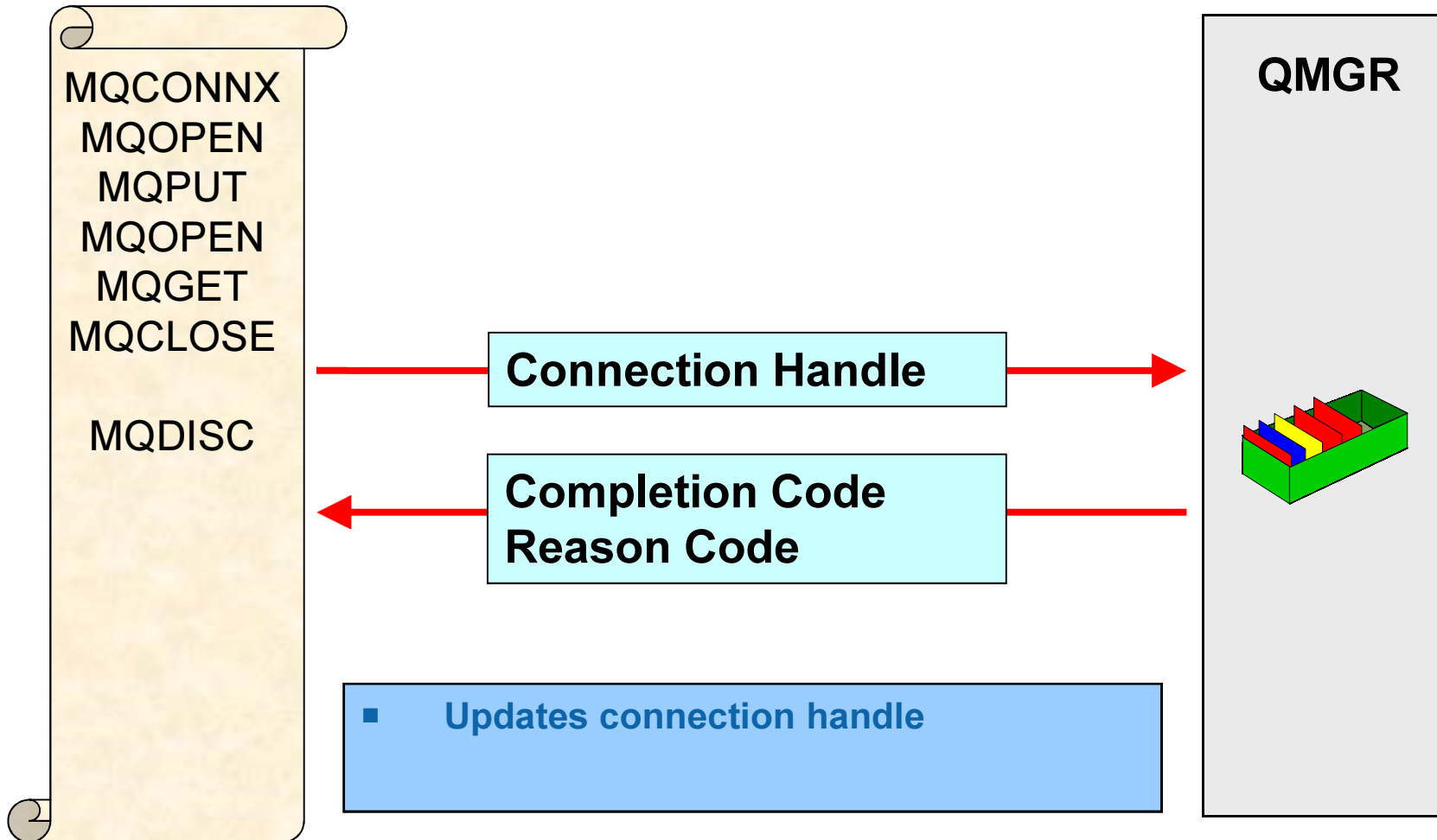
- Options available depending on object type

MQCO_DELETE	0x00000001	Permanent Dynamic Queue
MQCO_DELETE_PURGE	0x00000002	Permanent Dynamic Queue
MQCO_KEEP_SUB	0x00000004	Durable Subscription
MQCO_REMOVE_SUB	0x00000008	Durable Subscription
MQCO_QUIESCE	0x00000020	Read Ahead input handle

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



Disconnecting Application

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

```
MQCONNX (Qm ,  
         &cno ,  
         &hConn ,  
         &CompCode ,  
         &Reason) ;  
  
< Issue some MQI calls here >  
  
MQDISC ( &hConn ,  
         &CompCode ,  
         &Reason) ;
```

```
MQHCONN  hConn = MQHC_UNUSABLE_HCONN ;  
MQCHAR48 Qm    = "QM1" ;  
MQCNO    cno   = {MQCNO_DEFAULT} ;  
  
cno.Options |= MQCNO_HANDLE_SHARE_BLOCK |  
              MQCNO_RECONNECT
```

MQDISC Tips

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

Summary

- **Simple MQI – very easy to get started**
 - ▶ Let most fields have default values
 - ▶ Keep things simple if you can, for example
 - do not try and monitor channels
 - do not try to inquire queue depths

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

- **Check reason codes and log failures**
 - ▶ MQ trace can be useful

- **Also check out**
 - ▶ “An Introduction to and Comparison of the Different MQ APIs” by Matt Whitehead

Questions & Answers ?

Morag Hughson – morag@mqgem.com

MQGem Software

