The MQ API for Dummies - The Basics

Neil Johnston - neilj@uk.ibm.com
WebSphere MQ z/OS L3 – IBM Hursley

August 6th, 2012
Session #11515
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
  - C++
  - ActiveX (MQAX)
  - Perl
Interface

- Simple ‘handle’ based interface
  - Returned handle passed to subsequent call

- Each MQ API Call returns:
  - Completion Code
    - MQCC_OK 0
    - MQCC_WARNING 1
    - MQCC_FAILED 2
  - Reason Code
    - MQRC_xxxxxxxx 2xxx
    - MQRC_NONE 0

- Make sure you check the reason codes!
Data Structures

- Programmers should be familiar with:

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

• Use structure initialisers
  • MQMD md = { MQMD_DEFAULT };
  • Initialise to version 1

• Structures are versioned
  • Set the minimum version you need
    – md.Version = 2;
  • Don’t use current version
    – md.Version = MQMD_CURRENT_VERSION;

• Bear in mind that some structures are input/output
  • May need to reset values for subsequent call
    – Eg. MsgId & CorrelId field 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>
MQI Libraries

- **Windows**
  - `mqm.dll` server applications
  - `mqic32.dll` client applications

- **Unix**
  - `.../mqm/lib/libmqm.*` 32-bit server applications
  - `.../mqm/lib64/libmqm.*` 64-bit server applications
  - `.../mqm/lib/libmqic.*` 32-bit client applications
  - `.../mqm/lib64/libmqic.*` 64-bit client applications
  - `_r` threaded variants on some platforms.

- **Link with appropriate library – client or server**
  - Or dynamically load
Connect

- Queue Manager Name
- Connection Handle
- Completion Code
- Reason Code

- Basic connect
Connect with extended options

MQCONNX

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

Queue Manager Name
Connection Options

Connection Handle
Completion Code
Reason Code

QMGR
Connecting

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

- **Connections options**
  - Connection not thread specific
  - Client reconnect

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

// Set options for connection
cno.Options |= MQCNOHANDLE_SHARE_BLOCK | MQCNO_RECONNECT;
```

```c
MQCONNX( Qm,
    &cno,
    &hQm,
    &CompCode,
    &Reason);

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

- Don’t hardcode Queue Manager names
  - Pass as parameter or configure in INI file

- Best to use MQCONNX
  - Has options structure should it be needed

- Most expensive verb
  - Don’t issue it repeatedly for each request
    - Often problem for OO languages

- If MQI handle need to be used on different threads
  - Use MQCNO_HANDLE_SHARE_BLOCK

- 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 an Object

- **Indicate type of open required**
  - input, output, inquire etc
- **Indicate object name to open**
  - Queue name
  - Topic
Open an object

- MQOPEN an object
- 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

```c
MQOBJ hObj = MQO_UNUSABLE_HOBJ;
MQOD ObjDesc = {MQOD_DEFAULT};

ObjDesc.ObjectType = MQOT_Q;
strcpy(ObjectDesc.ObjectName, “Q1”);
```
Object Descriptor (MQOD)

```c
struct tagMQOD {
    MQCHAR4    StrucId;            /* Structure identifier */
    MQLONG     Version;            /* Structure version number */
    MQLONG     ObjectType;         /* Object type */
    MQCHAR48   ObjectName;         /* Object name */
    MQCHAR48   ObjectQMgrName;     /* Object queue manager name */
    MQCHAR48   DynamicQName;       /* Dynamic queue name */
    MQCHAR12   AlternateUserId;    /* Alternate user identifier */
    /* Ver:1 */
    MQLONG     RecsPresent;        /* Number of object records present */
    MQLONG     KnownDestCount;     /* Number of local queues opened successfully */
    MQLONG     UnknownDestCount;   /* Number of remote queues opened */
    MQLONG     InvalidDestCount;   /* Number of queues that failed to open */
    MQLONG     ObjectRecOffset;    /* Offset of first object record from start of MQOD */
    MQLONG     ResponseRecOffset;  /* Offset of first response record from start of MQOD */
    MQPTR      ObjectRecPtr;       /* Address of first object record */
    MQPTR      ResponseRecPtr;     /* Address of first response record */
    /* Ver:2 */
    MQBYTE40   AlternateSecurityId; /* Alternate security identifier */
    MQCHAR48   ResolvedQName;      /* Resolved queue name */
    MQCHAR48   ResolvedQMgrName;   /* Resolved queue manager name */
    /* Ver:3 */
    MQCHARV    ObjectString;       /* Object long name */
    MQCHARV    SelectionString;    /* Message Selector */
    MQCHARV    ResObjString;       /* Resolved long object name */
    MQLONG     ResolvedType;       /* Alias queue resolved object type */
    /* Ver:4 */
};
```
#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                     0x00010000
#define MQOO_NO_READ_AHEAD             0x00020000
#define MQOO_READ_AHEAD                0x00040000

- 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

Connection Handle
Object Handle
Message Descriptor
Put Message Options
Message Data
Completion Code
Reason Code

Updates structure
- Message Descriptor
- Put Message Options
Putting Application

- MQOPEN a queue
- MQPUT a message
  - Simple Hello World message:
    - Set message format to string
  - Put out 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;
```
Message Descriptor

struct tagMQMD {
    MQCHAR4   StrucId;        /* Structure identifier */
    MQLONG    Version;        /* Structure version number */
    MQLONG    Report;         /* Options for report messages */
    MQLONG    MsgType;        /* Message type */
    MQLONG    Expiry;         /* Message lifetime */
    MQLONG    Feedback;       /* Feedback or reason code */
    MQLONG    Encoding;       /* Numeric encoding of message data */
    MQLONG    CodedCharSetId; /* Character set identifier of message data */
    MQCHAR8   Format;         /* Format name of message data */
    MQLONG    Priority;       /* Message priority */
    MQLONG    Persistence;    /* Message persistence */
    MQBYTE24  MsgId;          /* Message identifier */
    MQBYTE24  CorrelId;       /* Correlation identifier */
    MQLONG    BackoutCount;   /* Backout counter */
    MQCHAR48  ReplyToQ;       /* Name of reply queue */
    MQCHAR48  ReplyToQMgr;    /* Name of reply queue manager */
    MQCHAR12  UserIdentifier; /* User identifier */
    MQBYTE32  AccountingToken; /* Accounting token */
    MQCHAR32  ApplIdentityData; /* Application data relating to identity */
    MQLONG    PutApplType;    /* Type of application that put the message */
    MQCHAR28  PutApplName;    /* Name of application that put the message */
    MQCHAR8   PutDate;        /* Date when message was put */
    MQCHAR8   PutTime;        /* Time when message was put */
    MQCHAR4   ApplOriginData; /* Application data relating to origin */
    MQLONG    GroupId;        /* Group identifier */
    MQLONG    MsgSeqNumber;   /* Sequence number of logical message within group */
    MQLONG    Offset;         /* Offset of data in physical message from start of logical message */
    MQLONG    OriginalLength; /* Length of original message */
};
struct tagMQPMO {
    MQCHAR4 StrucId;            /* Structure identifier */
    MQLONG Version;            /* Structure version number */
    MQLONG Options;            /* Options that control the action of MQPUT and MQPUT1 */
    MQLONG Timeout;            /* Reserved */
    MQHOBJ Context;            /* Object handle of input queue */
    MQLONG KnownDestCount;     /* Number of messages sent successfully to local queues */
    MQLONG UnknownDestCount;   /* Number of messages sent successfully to remote queues */
    MQLONG InvalidDestCount;   /* Number of messages that could not be sent */
    MQCHAR48 ResolvedQName;    /* Resolved name of destination queue */
    MQCHAR48 ResolvedQMgrName; /* Resolved name of destination queue manager */
    /* Ver:1 */
    MQLONG RecsPresent;        /* Number of put message records or response records present */
    MQLONG PutMsgRecFields;    /* Flags indicating which MQPMR fields are present */
    MQLONG PutMsgRecOffset;    /* Offset of first put message record from start of MQPMO */
    MQLONG ResponseRecOffset;  /* Offset of first response record from start of MQPMO */
    MQPTR PutMsgRecPtr;        /* Address of first put message record */
    MQPTR ResponseRecPtr;      /* Address of first response record */
    /* Ver:2 */
    MQHMSG OriginalMsgHandle;  /* Original message handle */
    MQHMSG NewMsgHandle;       /* New message handle */
    MQLONG Action;            /* The action being performed */
    MQLONG PubLevel;          /* Publication level */
    /* Ver:3 */
};
Put Options

- Options can be ‘ored’ together as required

```
#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
```
MQPUT Tips

- Always use explicit syncpoint setting
  - Defaults are not the same on z/OS and Distributed
  - Generally
    - MQPMO_SYNCPPOINT  – when persistent
    - MQPMO_NO_SYNCPPOINT  – when non-persistent

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

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

MQCONNX MQOPEN MQPUT
MQOPEN MQGET

Connection Handle
Object Handle
Message Descriptor
Get Message Options
Buffer Size

Message Data
Message Length
Completion Code
Reason Code

- Updates structures:
  - Message Descriptor
  - Get Message Options
Getting Application

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

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

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

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

- Options can be ‘ored’ together as required

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

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

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

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

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

- Updates structures:
  - Message Descriptor
  - Put Message Options
- Very similar to a normal P2P Put

Connection Handle
Object Handle
Message Descriptor
Put Message Options
Message Data

Completion Code
Reason Code

MQCONNX
MQOPEN
MQPUT

QMGR
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
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’
Subscribing Application

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

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

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

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

```c
struct tagMQSD {
    MQCHAR4 StrucId;           /* Structure identifier */
    MQLONG Version;            /* Structure version number */
    MQLONG Options;            /* Options associated with subscribing */
    MQCHAR48 ObjectName;       /* Object name */
    MQCHAR12 AlternateUserId;  /* Alternate user identifier */
    MQBYTE40 AlternateSecurityId; /* Alternate security identifier */
    MQLONG SubExpiry;          /* Expiry of Subscription */
    MQCHARV ObjectString;      /* Object long name */
    MQCHARV SubName;           /* Subscription name */
    MQCHARV SubUserData;       /* Subscription user data */
    MQBYTE24 SubCorrelId;      /* Correlation Id related to this subscription */
    MQLONG PubPriority;        /* Priority set in publications */
    MQBYTE32 PubAccountingToken; /* Accounting Token set in publications */
    MQCHAR32 PubApplIdentityData; /* Appl Identity Data set in publications */
    MQCHARV SelectionString;   /* Message selector structure */
    MQLONG SubLevel;           /* Subscription level */
    MQCHARV ResObjectString;   /* Resolved long object name */
};
```
Subscribe Options

- Options can be ‘ored’ together as required

#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
Subscribing Tips

• Managed handles make things simpler

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

• For durable subscriptions
  • MQSO_CREATE | MQSO_RESUME makes it simpler
Async Consume

• Alternative to MQGET

• Uses callback semantic

• Applications register a callback function

• Callback is invoked when messages arrive that match the registered criteria

• MQ manages buffers for you

• Message consumption controlled via MQCTL verb
Async Consume - MQCB(Register)

- **Operation:** MQOP_REGISTER
- **MQMD** Used for Msg Selection and conversion
- **Callback Descriptor:** Used to declare callback function.

**MQCONNX**
**MQOPEN**
**MQCB**
**Completion Code**
**Reason Code**
**Connection Handle**
**Operation**
**Callback Descriptor**
**Object Handle**
**Message Descriptor**
**Get Message Options**

**QMGR**
Async Consume - MQCTL(Start)

- Operation: MQOP_START
- Previously registered callback will be called repeatedly whenever a message is available
Async Consume App

- MQCB: Register a callback function
- MQCTL: Start consuming

```c
MQCB ( hConn,
      MQOP_REGISTER,
      &cbd,
      hObj,
      &md,
      &gmo,
      &CompCode,
      &Reason);

MQCTL ( hConn,
         MQOP_START,
         &ctlo,
         &CompCode,
         &Reason);

void MessageConsumer(MQHCONN   hConn,
                     MQMD    * pMsgDesc,
                     MQGMO   * pGetMsgOpts,
                     MQBYTE  * Buffer,
                     MQCBC   * pContext)
{
    processMessage();
    MQCTL(hConn, MQOP_STOP, &ctlo, &CC, &RC);
}

MQCBD  cbd = {MQCBD_DEFAULT};

cbd.CallbackFunction = (MQPTR) MessageConsumer;
```
typedef struct tagMQCBD MQCBD;

struct tagMQCBD {
    MQCHAR4    StrucId;                 /* Structure identifier */
    MQLONG     Version;                 /* Structure version number */
    MQLONG     CallBackType;            /* Callback function type */
    MQLONG     Options;                 /* Options controlling message consumption */
    MQPTR      CallbackArea;            /* User data passed to the function */
    MQPTR      CallbackFunction;        /* Callback function pointer */
    MQCHAR128  CallbackName;            /* Callback name */
    MQLONG     MaxMsgLength;            /* Maximum message length */
};
Close a handle

Connection Handle
Object Handle Close
Options

Completion Code
Reason Code

- Updates Object Handle

QMGR

MQCONNX
MQOPEN
MQPUT
MQOPEN
MQGET

MQCLOSE
Closing Application

- MQOPEN a queue
- MQCLOSE a queue
  - Normally we’d do something
  - Note address of MQHOBJ

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

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

- Options available depending on object type

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

- In triggered applications
  - Only close triggered queue if application ending

- If implementing queue cache
  - Close ‘rarely used’ queues in a timely fashion
    - Open queues cannot be deleted/purged and use memory

- For read ahead queues
  - Use MQCO_QUIESCE to avoid message loss
Disconnect from Queue Manager

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

Connection Handle

Completion Code
Reason Code

- Updates connection handle
Disconnecting Application

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

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

cno.Options |= MQCNO_HANDLE_SHARE_BLOCK | MQCNO_RECONNECT
```

```c
MQCONNX(Qm,
    &cno,
    &hQm,
    &CompCode,
    &Reason);

< Issue some MQI calls here >

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

• Ensure application disconnects if QM quiescing
  • Will prevent Queue Manager from ending

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

• MQDISC is an implicit commit
  • May want to consider issuing MQBACK() first

• Still call MQDISC
  • If MQI call returns MQRC_CONNECTION_BROKEN

• 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
    - do not try and monitor channels for example

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

- Check reason codes and log failures
  - MQ trace can be useful
Thank-you

Any questions?
<table>
<thead>
<tr>
<th>Time</th>
<th>Monday</th>
<th>Tuesday</th>
<th>Wednesday</th>
<th>Thursday</th>
<th>Friday</th>
</tr>
</thead>
<tbody>
<tr>
<td>08:00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Free MQ! - MQ Clients and what you can do with them</td>
</tr>
<tr>
<td>09:30</td>
<td>Clustering – the easier way to connect your Queue Managers</td>
<td>MQ on z/OS – vivisection</td>
<td>The Dark Side of Monitoring MQ - SMF 115 and 116 record reading and interpretation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11:00</td>
<td>Diagnosing problems for Message Broker</td>
<td>Lock it down - WebSphere MQ Security</td>
<td>Using IBM WebSphere Application Server and IBM WebSphere MQ Together</td>
<td></td>
<td>Spreading the message – MQ pubsub</td>
</tr>
<tr>
<td>12:15</td>
<td>Highly Available Messaging - Rock solid MQ</td>
<td>Putting the web into WebSphere MQ: A look at Web 2.0 technologies</td>
<td>The Doctor is In and Lots of Help with the MQ family - Hands-on Lab</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01:30</td>
<td>WebSphere MQ 101: Introduction to the world's leading messaging provider</td>
<td>What's new in the WebSphere MQ Product Family</td>
<td>Extending IBM WebSphere MQ and WebSphere Message Broker to the Cloud</td>
<td></td>
<td></td>
</tr>
<tr>
<td>03:00</td>
<td>First steps with WebSphere Message Broker: Application integration for the messy</td>
<td>What's new in Message Broker V8.0</td>
<td>Under the hood of Message Broker on z/OS - WLM, SMF and more</td>
<td></td>
<td>The Do's and Don'ts of z/OS Queue Manager Performance</td>
</tr>
<tr>
<td>04:30</td>
<td>The MQ API for Dummies - the Basics</td>
<td>What the **** is going on in my Queue Manager!?</td>
<td>Diagnosing problems for MQ</td>
<td></td>
<td>Shared Q using Shared Message Data Sets</td>
</tr>
<tr>
<td>06:00</td>
<td></td>
<td></td>
<td>For your eyes only - WebSphere MQ Advanced Message Security</td>
<td></td>
<td>MQ Q-Box - Open Microphone to ask the experts questions</td>
</tr>
</tbody>
</table>

This was session 11515 - The rest of the week ……
Please fill in evaluations (Session # 11515)