WebSphere MQ Application Design, the Good, the Bad and the Ugly
Session 12624

Paul S Dennis
dennisps@uk.ibm.com
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 verb returns
  – Completion Code
    • MQCC_OK 0
    • MQCC_WARNING 1
    • MQCC_FAILED 2
  – Reason Code
    • MQRC_xxxxxxx 2xxx
    • MQRC_NONE 0

• GOOD: Checking 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

• **GOOD: Use structure initialisers**
  - MQMD md = { MQMD_DEFAULT };
  - Initialise to version 1

• **Structures are versioned**
  - Set the minimum version you need
    - md.Version = 2;
  - BAD: Using current version
    - md.Version = MQMD_CURRENT_VERSION;

• **Bear in mind that some structures are input/output**
  - May need to reset values for subsequent call
    - GOOD: Eg. MsgId & CorrelId fields of MQMD on MQGET call
MQ Elementary Data Types

- **The main MQI data types**

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

- Basic connect

MQCONN

Queue Manager Name

Connection Handle Completion Code Reason Code

QMGR
Connect with extended options

MQCONNX

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

Queue Manager Name
Connection Options

Connection Handle
Completion Code
Reason Code

QMGR
Connecting

- **MQCONNX**
  - UGLY: hardcoded QM name
  - GOOD: Always check reason codes

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

```c
MQHCONN hQm = MQHC_UNUSABLE_HCONN;
MQCHAR48 Qm = “QM1”; 
MQCNO cno = {MQCNO_DEFAULT};
cno.Options |= MQCNO_HANDLE_SHARE_BLOCK |
MQCNO_RECONNECT;

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

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

- **UGLY**: hardcoded Queue Manager names
  - Pass as parameter or configure in INI file
- **GOOD**: Best to use MQCONNX
  - Has options structure should it be needed
- **BAD**: 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
- **GOOD**: If reconnecting use exponential back-off with random wait
  - Try to avoid client storms
- **Can dynamically load MQ libraries if client or local binding**
  - Preferable to shipping two versions of the program
Open a Queue

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

- **MQOPEN a queue**
- **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
MQHOBJ hObj = MQHO_UNUSABLE_HOBJ;
MQOD ObjDesc = {MQOD_DEFAULT};

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

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

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

- Options can be ‘ORed’ together as required.
MQOPEN Tips

• **UGLY:** hardcoded queue/topic names
• **BAD:** Opening queues exclusively
  – Will reduce options for workload balancing
• **GOOD:** Use MQPUT1 if only opening queue to put one message
• **GOOD:** Consider queue cache for common used queues
  – MQOPEN is relatively expensive – load and security check
• **GOOD:** Use read ahead for performance gain
  – If client and non-persistent messaging
• **If opening model reply queues**
  – Be aware of how many instances of queues you may be creating
    • Particularly large numbers of clients.
  – May be better to share reply queue
Put a message

MQCONNX
MQOPEN
MQPUT

Connection Handle
Object Handle
Message Descriptor
Put Message Options
Message Data

Completion Code
Reason Code

• Updates structure
  – Message Descriptor
  – Put Message Options
Putting Application

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

```c
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)

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

<table>
<thead>
<tr>
<th>Field (V2)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GroupId</td>
<td>Group identifier</td>
</tr>
<tr>
<td>MsgSeqNumber</td>
<td>Sequence number of logical message within group</td>
</tr>
<tr>
<td>Offset</td>
<td>Offset of data in physical message from start of logical message</td>
</tr>
<tr>
<td>MsgFlags</td>
<td>Message flags</td>
</tr>
<tr>
<td>OriginalLength</td>
<td>Length of original message</td>
</tr>
</tbody>
</table>
## Put Message Options (MQPMO)

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

- Options can be ‘ORed’ together as required

```c
#define MQPMO_SYNCPOINT                0x00000002
#define MQPMO_NO_SYNCPOINT             0x00000004
#define MQPMO_DEFAULT_CONTEXT          0x00000020
#define MQPMO_NEW_MSG_ID               0x00000040
#define MQPMO_NEW_CORREL_ID            0x00000080
#define MQPMO_PASS_IDENTITY_CONTEXT   0x00000100
#define MQPMO_PASS_ALL_CONTEXT         0x00000200
#define MQPMO_SET_IDENTITY_CONTEXT     0x00000400
#define MQPMO_SET_ALL_CONTEXT          0x00000800
#define MQPMO_ALTERNATE_USER_AUTHORITY 0x00001000
#define MQPMO_FAIL_IFQUIESCING         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_RETAIR                   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

• **GOOD: 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

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 structure
  – Message Descriptor
  – Get Message Options

QMGR
Getting Application

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

```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);

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)

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

- **Define MQGMO_WAIT** 0x00000001
- **Define MQGMO_NO_WAIT** 0x00000000
- **Define MQGMO_SET_SIGNAL** 0x00000008
- **Define MQGMO_FAIL_IFQUIESCING** 0x00000200
- **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

• **GOOD: Avoid using default syncpoint setting**
  - Defaults are not the same on z/OS and Distributed
  - Generally
    • MQGMO_SYNCPOINT_IF_PERSISTENT

• **UGLY: Not using MQGMO_FAIL_IF_QUIESCING**
  - Ensure your application ends promptly

• **GOOD: Generally use MQGMO_CONVERT**
  - Even if you ‘think’ you don’t need it

• **GOOD: Remember to reset MsgId & CorrelId fields**
  - These fields are used for selection and are returned

• **BAD: Forgetting to handle ‘poison messages’**
  - Look at BackoutCount in MQMD

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

MQCONNX
MQOPEN
MQPUT

QMGR

Connection Handle
Object Handle
Message Descriptor
Put Message Options
Message Data

Completion Code
Reason Code

• Updates structure
  – Message Descriptor
  – Put Message Options

• Very similar to a normal P2P Put
Publishing Application

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

```c
MQOD ObjDesc = {MQOD_DEFAULT};

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

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

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

• **Choose topic string carefully**
  – GOOD: Use sensible topic hierarchy
    • Based on context of published data
  – UGLY: Using different topic for each publish
    • This is probably meta data, use message property
  – BAD: Using long topic strings when not necessary
    • Topic strings can be up to 10K bytes

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

- Updates structure
  - Subscription Descriptor
- Very similar to MQOPEN
Subscribing Application

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

```c
MQSD  SubDesc = {MQSD_DEFAULT};
SubDesc.ObjectString.VSPtr    = "Price/Fruit/Apples";
SubDesc.ObjectString.VSLength = MQVS_NULL_TERMINATED;
SubDesc.Options               = MQSO_CREATE
                             | MQSO_MANAGED
                             | MQSO_FAIL_IF_QUIESCING;
```
## Subscription Descriptor (MQSD)

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

- `#define MQSO_NON_DURABLE` 0x00000000
- `#define MQSO_READ_AHEAD` 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_IFQUIESCING` 0x00002000
- `#define MQSO_ALTERNATE_USER_AUTHORITY` 0x00004000
- `#define MQSO_WILDCARD_CHAR` 0x00100000
- `#define MQSO_WILDCARD_TOPIC` 0x00200000
- `#define MQSO_SET_CORREL_ID` 0x00400000
- `#define MQSO_SCOPE_QMGR` 0x04000000
- `#define MQSO_NO_READ_AHEAD` 0x08000000
- `#define MQSO_READ_AHEAD` 0x10000000

**Options can be ‘ORed’ together as required.**
Subscribing Tips

• **GOOD:** Managed handles make things simpler

• **BAD:** Using durable subscriptions when not necessary
  – Avoid build up of messages

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

Connection Handle
Object Handle Close Options
Completion Code
Reason Code

• Updates Object Handle
Closing Application

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

```c
MQCONN hConn;
MQOBJ hObj = MQHO_UNUSABLE_HOBJ;
MQOD ObjDesc = {MQOD_DEFAULT};

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

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

< Issue some MQI calls here > 

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

- Options available depending on object type

| 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**
  – **GOOD:** Only close triggered queue if application ending

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

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

MQCONNX
MQOPEN
MQPUT
MQOPEN
MQGET
MQCLOSE
MQDISC

Completion Code
Reason Code

Connection Handle

• Updates connection handle
Disconnecting Application

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

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

cno.Options |= MQCNOHANDLE_SHARE_BLOCK | MQCNO_RECONNECT
```

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

< Issue some MQI calls here >

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

- **GOOD**: 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
    • do not try and monitor channels for example

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

• Check reason codes and log failures
  – MQ trace can be useful
<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>Are you running too many queue managers or brokers?</td>
<td></td>
</tr>
<tr>
<td>09:30</td>
<td>What's New in WebSphere Message Broker</td>
<td></td>
<td></td>
<td>Diagnosing Problems for MQ</td>
<td>CICS and WMQ - The Resurrection of Useful</td>
</tr>
<tr>
<td>11:00</td>
<td>Extending IBM WebSphere MQ and WebSphere Message Broker to the Cloud</td>
<td>WMQ - Introduction to Dump Reading and SMF Analysis - Hands-on Lab</td>
<td>BIG Data Sharing with the cloud - WebSphere eXtreme Scale and WebSphere Message Broker integration</td>
<td>Getting the best availability from MQ on z/OS by using Shared Queues</td>
<td></td>
</tr>
<tr>
<td>12:15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01:30</td>
<td>Introduction to MQ</td>
<td>MQ on z/OS – Vivisection</td>
<td>Migration and maintenance, the necessary evil</td>
<td>The Dark Side of Monitoring MQ - SMF 115 and 116 Record Reading and Interpretation</td>
<td></td>
</tr>
<tr>
<td>03:00</td>
<td>First Steps With WebSphere Message Broker: Application Integration for the Messy</td>
<td>BIG Connectivity with WebSphere MQ and WebSphere Message Broker</td>
<td>WebSphere MQ CHINIT Internals</td>
<td>Using IBM WebSphere Application Server and IBM WebSphere MQ Together</td>
<td></td>
</tr>
<tr>
<td>04:30</td>
<td>WebSphere MQ application design, the good, the bad and the ugly</td>
<td>What’s New in the WebSphere MQ Product Family</td>
<td>MQ &amp; DB2 – MQ Verbs in DB2 &amp; Q-Replication</td>
<td>WebSphere MQ Channel Authentication Records</td>
<td></td>
</tr>
<tr>
<td>06:00</td>
<td></td>
<td></td>
<td>Clustering - The Easier Way to Connect Your Queue Managers</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Please fill in evaluations at share.org/SFEval   #12624