An Introduction to, and Comparison of, the Different MQ APIs

Matthew Whitehead
IBM MQ Development
mwhitehead@uk.ibm.com
Agenda

• Languages, Wire Formats, and APIs
• An Overview of C MQI, JMS, and MQ Light
• Recap of the Key MQ Features
• Features supported in the different MQ APIs
Agenda

• Languages, Wire Formats, and APIs
  • An Overview of C MQI, JMS, and MQ Light
  • Recap of the Key MQ Features
  • Features supported in the different MQ APIs
Languages, Wire Formats, and APIs

QM

Wire Formats

APIs

Languages/Runtimes
Languages, Wire Formats, and APIs

MQ Wire Format (& Local Bindings)

Wire Formats

APIs

Languages/ Runtimes
Languages, Wire Formats, and APIs

MQ Wire Format (& Local Bindings)  MQTT

QM

Wire Formats

APIs

Languages/ Runtimes
Languages, Wire Formats, and APIs

MQ Wire Format (& Local Bindings)  MQTT  HTTP

Wire Formats

Languages/Runtimes

APIs

QM
Languages, Wire Formats, and APIs

QM

MQ Wire Format (& Local Bindings)  MQTT  HTTP  AMQP 1.0

Wire Formats

APIs

Languages/ Runtimes
## Languages, Wire Formats, and APIs

<table>
<thead>
<tr>
<th>Languages/ Runtimes</th>
<th>APIs</th>
<th>Wire Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MQ Wire Format</td>
<td>MQTT</td>
<td>HTTP</td>
</tr>
<tr>
<td>(Local Bindings)</td>
<td></td>
<td>AMQP 1.0</td>
</tr>
</tbody>
</table>

- QM
Languages, Wire Formats, and APIs

**Languages/Runtimes**

- MQ Wire Format (& Local Bindings)
- MQ Object Oriented
- MQTT
- HTTP
- AMQP 1.0
- MQI

**APIs**

- QM

**Wire Formats**
Languages, Wire Formats, and APIs

Wire Formats
- MQ Wire Format
- MQTT
- HTTP
- AMQP 1.0

Languages/Runtimes
- MQI
- MQ Object Oriented
- XMS
- WCF

APIs
- MQM
Languages, Wire Formats, and APIs

- MQ Wire Format (& Local Bindings)
- MQTT
- HTTP
- AMQP 1.0

- MQI
- MQ Object Oriented
- XMS
- WCF
- JMS
Languages, Wire Formats, and APIs

MQ Wire Format (& Local Bindings)
MQI
MQ Object Oriented

Wire Formats
MQTT
HTTP
AMQP 1.0

APIs
XMS
WCF
JMS
MQTT

Languages/
Runtimes
QM
Languages, Wire Formats, and APIs

MQ Wire Format (& Local Bindings)

MQI
MQ Object Oriented

Wire Formats
MQTT
HTTP
AMQP 1.0

APIs
MQI
XMS
WCF
JMS
MQTT
MQ HTTP

Languages/Runtimes

Copyright © IBM 2016
Languages, Wire Formats, and APIs

MQ Wire Format (& Local Bindings)  MQTT  HTTP  AMQP 1.0

MQI  MQ Object Oriented  XMS  WCF  JMS  MQTT  MQ HTTP  MQ Light

Wire Formats  APIs  Languages/Runtimes

Copyright © IBM 2016
Languages, Wire Formats, and APIs

MQ Wire Format (& Local Bindings)

MQI
MQ Object Oriented

Wire Formats
MQTT
HTTP
AMQP 1.0

Languages/Runtimes
MQI
XMS
WCF
JMS
MQTT
MQ HTTP
MQ Light
Languages, Wire Formats, and APIs

- MQ Wire Format (& Local Bindings)
- MQTT
- HTTP
- AMQP 1.0

Wire Formats

- MQI
- MQ Object Oriented
- XMS
- WCF
- JMS
- MQTT
- MQ HTTP
- MQ Light

APIs

Languages/ Runtimes
Languages, Wire Formats, and APIs

Wire Formats
- MQ Wire Format (& Local Bindings)
- HTTP
- AMQP 1.0

Languages/ Runtimes
- MQI
- MQ Object Oriented
- MQTT
- WCF
- JMS
- MQ HTTP
- MQ Light
- XMS
- MQI
Languages, Wire Formats, and APIs

Wire Formats
- MQ Wire Format (& Local Bindings)
- AMQP 1.0

APIs
- MQTT
- HTTP
- MQ Light

Languages/Runtimes
- MQI
- MQ Object Oriented
- XMS
- WCF
- JMS
- MQTT
- MQ HTTP
Languages, Wire Formats, and APIs

MQ Wire Format (& Local Bindings)  MQTT  HTTP  AMQP 1.0

MQI  MQ Object Oriented  XMS  WCF  JMS  MQTT  MQ HTTP  MQ Light

Cobol
Languages, Wire Formats, and APIs

Wire Formats
- MQ Wire Format
  (& Local Bindings)
- MQTT
- HTTP
- AMQP 1.0

APIs
- MQI
- MQ Object Oriented
- XMS
- WCF
- JMS
- MQTT
- MQ HTTP
- MQ Light

Languages/
Runtimes
- PL/1
- Cobol

Copyright © IBM 2016
Languages, Wire Formats, and APIs

MQ Wire Format (& Local Bindings)  MQTT  HTTP  AMQP 1.0

MQI  MQ Object Oriented  XMS  WCF  JMS  MQTT  MQ HTTP  MQ Light

PL/1  Cobol  VB

Wire Formats  APIs  Languages/ Runtimes
Languages, Wire Formats, and APIs

QM

Wire Formats
MQ Wire Format (& Local Bindings) MQTT HTTP AMQP 1.0

APIs
MQI MQ Object Oriented XMS WCF JMS MQTT MQ HTTP MQ Light

Languages/ Runtimes
Cobol PL/1 C++ VB

Copyright © IBM 2016
Languages, Wire Formats, and APIs

Wire Formats
- MQ Wire Format (& Local Bindings)
- MQTT
- HTTP
- AMQP 1.0

APIs
- MQI
- MQ Object Oriented
- XMS
- WCF
- JMS
- MQTT
- MQ HTTP
- MQ Light

Languages/Runtimes
- PL/1
- C++
- Cobol
- VB
- Perl
Languages, Wire Formats, and APIs

MQ Wire Format (& Local Bindings)  MQTT  HTTP  AMQP 1.0

MQI  MQ Object Oriented  XMS  WCF  JMS  MQTT  MQ HTTP  MQ Light

PL/1  C++  C

Cobol  VB  Perl

Languages/ Runtimes

Wire Formats

APIs
### Languages, Wire Formats, and APIs

<table>
<thead>
<tr>
<th>Languages/ Runtimes</th>
<th>MQ Wire Format (&amp; Local Bindings)</th>
<th>Wire Formats</th>
<th>APIs</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQI</td>
<td>MQ Object Oriented</td>
<td>MQTT</td>
<td>HTTP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AMQP 1.0</td>
<td></td>
</tr>
<tr>
<td>Cobol</td>
<td>PL/1</td>
<td>XMS</td>
<td>WCF</td>
</tr>
<tr>
<td>VB</td>
<td></td>
<td>JMS</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MQTT</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MQ HTTP</td>
<td>MQ Light</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MQ Light</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>C++</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Perl</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ActiveX</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Languages, Wire Formats, and APIs

QM

MQ Wire Format (& Local Bindings)
MQI
MQ Object Oriented
XMS
WCF
JMS
MQTT
MQ HTTP
MQ Light

MQI
PL/1
C++
C
WCF
ActiveX

Wire Formats
MQTT
HTTP
AMQP 1.0

APIs
Languages/
Runtimes
Cobol
VB
Perl

Copyright © IBM 2016
Languages, Wire Formats, and APIs

Wire Formats
- MQ Wire Format (& Local Bindings)
- MQTT
- HTTP
- AMQP 1.0

APIs
- MQI
- MQ Object Oriented
- XMS
- WCF
- JMS
- MQTT
- MQ HTTP
- MQ Light

Languages/Runtimes
- PL/1
- C++
- C
- WCF
- Java
- VB
- Perl
- ActiveX
Languages, Wire Formats, and APIs

Wire Formats
- MQ Wire Format (& Local Bindings)
- MQTT
- HTTP
- AMQP 1.0

APIs
- MQI
- MQ Object Oriented
- XMS
- WCF
- JMS
- MQTT
- MQ HTTP
- MQ Light

Languages/Runtimes
- Cobol
- PL/1
- C++
- C
- Perl
- ActiveX
- VB
- WCF
- Java
- .Net
Languages, Wire Formats, and APIs

MQ Wire Format (& Local Bindings)

MQTT  HTTP  AMQP 1.0

MQI  MQ Object Oriented

XMIS  WCF  JMS  MQTT  MQ HTTP  MQ Light

Cobol  PL/1  C++  C  WCF  Java  C#  Ruby

WCF  .Net
Languages, Wire Formats, and APIs

Wire Formats
- MQ Wire Format (& Local Bindings)
- MQTT
- HTTP
- AMQP 1.0

MQI
MQ Object Oriented
XMS
WCF
JMS
MQTT
MQ HTTP
MQ Light

Languages/Runtimes
- Cobol
- PL/1
- VB
- C++
- Perl
- C
- ActiveX
- WCF
- Java
- .Net
- C#
- nodejs
- Ruby
Languages, Wire Formats, and APIs

Languages/Runtimes:
- Cobol
- PL/1
- VB
- Perl
- C
- ActiveX
- WCF
- .Net
- Java
- C#  

APIs:
- MQI
- MQ Object Oriented
- XMS
- WCF
- JMS
- MQTT
- MQ HTTP
- MQ Light

Wire Formats:
- MQTT
- HTTP
- AMQP 1.0

Native C++ is now stabilised. Recommendations are XMS or the C MQI.
Languages, Wire Formats, and APIs

MQ Wire Format (& Local Bindings)

MQ Object Oriented (Base Java)

Languages/Runtimes

Cobol
PL/1
VB
Perl
C++
C
ActiveX
WCF
.Net
C#
nodejs
Ruby
Java
WCF
MQTT
HTTP
AMQP 1.0

Wire Formats

Base Java is now stabilised. JMS the recommended MQ-Java interface

Copyright © IBM 2016

Capitalware's MQ Technical Conference v2.0.1.6
Languages, Wire Formats, and APIs

MQ Wire Format (& Local Bindings)

MQ Object Oriented (Base Java)

Wire Formats

MQI
XMS
MQ Light
MQ HTTP
JMS
MQTT
HTTP
AMQP 1.0

Languages/
Runtimes

Cobol
PL/1
VB
Perl
C++
C
ActiveX
WCF
.Net
C#
nodejs
Ruby

Copyright © IBM 2016
Languages, Wire Formats, and APIs

MQ Wire Format (& Local Bindings)  MQTT  HTTP  AMQP 1.0

MQI  MQ Object Oriented  XMS  WCF  JMS  MQTT  MQ HTTP  MQ Light

Cobol  PL/1  C++  C  WCF  .Net  C#  Ruby  node.js

Languages/ Runtimes

Wire Formats

APIs
Languages, Wire Formats, and APIs

QM

MQ Wire Format (& Local Bindings)  MQTT  HTTP  AMQP 1.0

MQI  MQ Object Oriented  XMS  WCF  JMS  MQTT  MQ HTTP  MQ Light

Cobol  PL/1  C++  C  WCF  Java

VB  Perl  ActiveX  .Net  C#

nodejs  Ruby

Copyright © IBM 2016
Agenda

• Languages, Wire Formats, and APIs
• An Overview of C MQI, JMS, and MQ Light
• Recap of the Key MQ Features
• Features supported in the different MQ APIs
C MQI Overview

**Structures**

- **MQMD** – Message descriptor
- **MQOD** – Object descriptor
- **MQSD** – Subscription descriptor
- **MQPD** – Property descriptor
- **MQCD** – Client channel descriptor
- **MQCNO** – Connect options
- **MQGMO** – Get message options
- **MQPMO** – Put message options
- **MQCSP** – Security parameters

**Verbs**

- **MQCONN** – Connect to QM
- **MQCONNX** – Connect with opts
- **MQDISC** – Disconnect from QM
- **MQOPEN** – Open queue/topic
- **MQCLOSE** – Close queue/topic
- **MQBEGIN** – Start global transaction
- **MQCMIT** – Commit transaction
- **MQBACK** – Rollback transaction
- **MQSUB** – Subscribe to topic
- **MQINQ** – Inquire about object
- **MQCTL** – Control callbacks
- **MQSET** – Set object attribute
**Constants** - Completion codes

MQCC_OK – Verb completed OK  
MQCC_WARNING – Verb completed but with warnings (see MQRC)  
MQCC_FAILED – Verb failed

**Constants** - Return codes (there are many of these), e.g:

MQRC_NONE – Nothing to check  
MQRC_Q_MGR_QUIESCING – Can’t do the action – QM is stopping  
MQRC_NOT_AUTHORIZED – Can’t do the action – you’re not authorized  
MQRC_TRUNCATED_MSG_ACCEPTED – The message is too big for the buffer  
MQRC_MSG_TOO_BIG_FOR_Q – Guess what this means…

**Constants** - Misc (there are loads of these), e.g:

MQIA_CURRENT_Q_DEPTH, MQCACH_CHANNEL_NAME, MQCA_CLUSTER_NAME, …
C MQI Overview

```c
MQOD od = {MQOD_DEFAULT};  /* Object Descriptor */
MQMD md = {MQMD_DEFAULT};  /* Message Descriptor */
MQPMO pmo = {MQPMO_DEFAULT};  /* put message options */
MQCNO cno = {MQCNO_DEFAULT};  /* connection options */
MQCSP csp = {MQCSP_DEFAULT};  /* security parameters */

MQHCONN Hcon;  /* connection handle */
MQHOBJ Obj;  /* object handle */
MQLONG Q_options;  /* MQOPEN options */
MQLONG C_options;  /* MQCLOSE options */
MQLONG CompCode;  /* completion code */
MQLONG OpenCode;  /* MQOPEN completion code */
MQLONG Reason;  /* reason code */
MQLONG CReason;  /* reason code for MQCONNX */
MQLONG messlen;  /* message length */
char buffer[65535];  /* message buffer */
char QMName[50];  /* queue manager name */
char *UserId;  /* UserId for authentication */
char Password[MQ_CSP_PASSWORD_LENGTH + 1] = {0};  /* For auth */
```
C MQI Overview

MQCONNX (QMName, /* queue manager */
           &cno, /* connection options */
           &Hcon, /* connection handle */
           &CompCode, /* completion code */
           &CReason); /* reason code */

if (CompCode == MQCC_FAILED)
{
    // Something went wrong - print out the error and quit
}

if (CompCode == MQCC_WARNING)
{
    // Handle the warning, decide whether to continue
}
O_options = MQOO_OUTPUT               /* open queue for output */
   | MQOO_FAIL_IF_QUIESCING;          /* but not if MQM stopping */

MQOPEN(Hcon,
   &od,
   O_options,
   &Hobj,
   &OpenCode,
   &Reason);

if (OpenCode == MQCC_FAILED)
{
   // Something went wrong - print the error and quit
}

if (Reason != MQRC_NONE)
{
   // Something went wrong - print the error and quit
}
if (fgets(buffer, sizeof(buffer), stdin) != NULL) /* read a message from keyboard input */{
    messlen = (MQLONG)strlen(buffer); /* get the length of the message */
    if (buffer[messlen-1] == '\n') /* remove the last newline character */{
        buffer[messlen-1] = '\0';
        --messlen;
    }
}

memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId) );

MQPUT(Hcon, /* connection handle */
    Hobj, /* object handle */
    &md, /* message descriptor */
    &pmo, /* default options (datagram) */
    messlen, /* message length */
    buffer, /* message buffer */
    &CompCode, /* completion code */
    &Reason); /* reason code */

if (Reason != MQRC_NONE)
{
    // Print out reason code and quit
}
}
JMS Overview

Objects

ConnectionFactory – from which connections are created
Connection – from which sessions are created
Session – from which producers, consumers, and destinations are created
Destination – to which messages are sent (Queue and Topic sub-types)

MessageProducer – with which messages are sent
MessageConsumer – with which messages are received
QueueBrowser – with which messages are browsed from a queue
TextMessage/BytesMessage/ObjectMessage/MapMessage/StreamMessage

JMSContext – JMS 2 – combination of connection and session
JMSProducer and JMSConsumer – JMS 2

Key Actions

connection.start(…) ← (not required in JMS 2)
session.createProducer(…)
producer.send(…)
message.acknowledge()
connection.close()
Connection connection = null;
Session session = null;
Destination destination = null;
MessageProducer producer = null;

try {
    // Create a connection factory
    JmsFactoryFactory ff =
        JmsFactoryFactory.getInstance(WMQConstants.WMQ_PROVIDER);
    JmsConnectionFactory cf = ff.createConnectionFactory();

    // Set up connection properties
    cf.setStringProperty(WMQConstants.WMQ_HOST_NAME, host);
    cf.setIntProperty(WMQConstants.WMQ_PORT, port);
    cf.setStringProperty(WMQConstants.WMQ_CHANNEL, channel);
    cf.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, queueManagerName);
    cf.setStringProperty(WMQConstants.USERID, user);
    cf.setStringProperty(WMQConstants.PASSWORD, password);
    cf.setBooleanProperty(WMQConstants.USER_AUTHENTICATION_MQCSP, true);
} catch (JMSException e) {  // Handle exceptions here
}
// Create JMS objects
connection = cf.createConnection(); // MQCONN is made under the covers

session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);

if (isTopic) {
    destination = session.createTopic(destinationName);
} else {
    destination = session.createQueue(destinationName);
}

// MQOPEN is performed under the covers
producer = session.createProducer(destination);
try {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));

    do {
        System.out.print("Enter some text to be sent <ENTER to finish>: ");
        System.out.flush();

        String line = in.readLine();
        if (line != null && line.trim().length() != 0) {
            TextMessage message = session.createTextMessage(line);
            producer.send(message); // MQPUT happens under the covers

            System.out.println("Sent message: \n" + message);
        }
    } while (line != null);
} catch (Exception JMSException | IOException e) {
    // Handle exceptions here
}
Library Import

```javascript
var mqlight = require('mqlight');
```

Client instantiation

```javascript
var opts = {service: "amqp://localhost", id: "matt"};
var client = mqlight.createClient(opts);
```

Client-connected callback

```javascript
client.on('started', function() {<code to run when the client has connected>});
```

Client-error callback

```javascript
client.on('error', function() {<code to run when the client has an error>});
```
Message-received callback

client.on('message', function(data, delivery) {<code to run when a message is received>});

Malformed-message-received callback

client.on('malformed', function(data, delivery) {<code to run when a malformed message is received>});

Client subscribe, plus client-subscribed callback

client.subscribe(pattern, share, options, function(err, pattern) {<code to run when the subscribe returns>});

Send message, plus message-sent or message-not-sent callback

var buffered = !client.send(topic, body, options, callback);
var mqlight = require('mqlight'); // Import the MQ Light client module
var nopt = require('nopt');
var uuid = require('node-uuid');
var fs = require('fs');

var opts = { // Define configuration options
  service: "amqp://localhost:5672",
  id: "client-01"
};

var client = mqlight.createClient(opts); // Create the connection
// Once we have connected we can start sending messages
client.on('started', function() {

  var sendNextMessage = function() { setImmediate(sendMessage); };

  var sendMessage = function() {

    var options = { qos: mqlight.QOS_AT_LEAST_ONCE };

    var callback = function(err, topic, data, options) {
      if (err) {
        // The send failed - print the error and quit
      } else {
        // Nothing to do - send took place successfully
      }
    }

    client.send(topic, body, options, callback);

  };

});
Agenda

• A refresher on C, JMS, and MQ Light
• An Overview of C MQI, JMS, and MQ Light
• Recap of the Key MQ Features
• Features supported in the different MQ APIs
### MQ features

- Point-to-Point messaging
- Publish/Subscribe messaging
- Shared Subscriptions
- Message persistence
- Message expiry
- Message grouping
- Message segmentation
- Message selection
- Local transactions
- XA/Global transactions
- At least once/at most once/exactly once delivery
- HA failover

- Message properties
- Asynchronous Consume
- Message browsing
Point-to-Point Messaging

<table>
<thead>
<tr>
<th>MQ Wire Format</th>
<th>MQTT</th>
<th>HTTP</th>
<th>AMQP 1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td></td>
<td></td>
<td>~</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MQI</th>
<th>MQ OO</th>
<th>MQTT</th>
<th>MQ HTTP</th>
<th>MQ Light</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>~</td>
</tr>
</tbody>
</table>

Notes:
- ~ can be set indirectly

Examples

**C/MQI**

```c
strncpy(od.ObjectName, “Q1”, (size_t)MQ_Q_NAME_LENGTH);
MQOPEN(Hcon, &od, O_options,&Hobj, &OpenCode, &Reason);
```

**JMS**

```java
destination = session.createQueue(“Q1”);
producer = session.createProducer(destination);
```

**nodejs/ MQ Light**

```javascript
client.send(“Q1”, body, options, callback);
```
Publish/Subscribe Messaging

<table>
<thead>
<tr>
<th>MQ Wire Format</th>
<th>MQTT</th>
<th>HTTP</th>
<th>AMQP 1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td></td>
<td>~</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MQI</th>
<th>MQ OO</th>
<th>MQTT</th>
<th>MQ HTTP</th>
<th>MQ Light</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

**Notes:**

- ~ can be set indirectly

**Examples**

**C/MQI**

```c
od.ObjectString.VSPtr="my/first/messaging/topic";
od.ObjectString.VSLength=(MQLONG)strlen(argv[1]);
od.ObjectType = MQOT_TOPIC;
MQOPEN(Hcon, &od, O_options,&Hobj, &OpenCode, &Reason);
```

**JMS**

```java
destination = session.createTopic("my/first/topic");
producer = session.createProducer(destination);
```

**nodejs/MQ Light**

```javascript
client.send("my/first/messaging/topic", body, options, callback);
```
## Shared Subscriptions

<table>
<thead>
<tr>
<th>MQ Wire Format</th>
<th>MQTT</th>
<th>HTTP</th>
<th>AMQP 1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td></td>
<td>✓</td>
<td>~</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MQI</th>
<th>MQ OO</th>
<th>MQTT</th>
<th>MQ HTTP</th>
<th>MQ Light</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMS 2</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

### Notes:
- ~ can be set indirectly
- Not supported by C MQI
- Primarily added to MQ for JMS 2 support
- MQI has other ways of achieving similar pattern

### Examples

**C/MQI** → **N/A**

```java
MessageConsumer messageConsumer = session.createSharedConsumer(topic,"mySubscription");
```

**JMS 2** →

```java
MessageConsumer messageConsumer = session.createSharedDurableConsumer(topic,"myDurableSub");
```

**nodejs/MQ Light** →

```javascript
client.subscribe(pattern, "myShare", options, function(err, pattern) { // Handle errors });
```
# Message Persistence

<table>
<thead>
<tr>
<th>MQ Wire Format</th>
<th>MQTT</th>
<th>HTTP</th>
<th>AMQP 1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>~</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MQI</th>
<th>MQ OO</th>
<th>MQTT</th>
<th>MQ HTTP</th>
<th>MQ Light</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>~</td>
<td>✓</td>
<td>~</td>
</tr>
</tbody>
</table>

**Examples**

**C/MQI**

```c
MQMD md = {MQMD_DEFAULT};
    md.Persistence = MQPER_PERSISTENT;
...
MQPUT(Hcon, Hobj, &md, &pmo, messlen, buffer, &CC, &RC);
```

**JMS**

```java
MessageProducer producer = session.createProducer(dest);
producer.setDeliveryMode(DeliveryMode.PERSISTENT);
```

**nodejs/ MQ Light**

```javascript
var options = { qos: mqlight.QOS_AT_LEAST_ONCE };
client.send(topic, body, options, callback);
```

**Notes:**

- ~ can be set indirectly
Message Expiry

<table>
<thead>
<tr>
<th>MQ Wire Format</th>
<th>MQTT</th>
<th>HTTP</th>
<th>AMQP 1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MQI</th>
<th>MQ OO</th>
<th>MQTT</th>
<th>MQ HTTP</th>
<th>MQ Light</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Notes:
- MQ expiry value is ‘tenths of a second’

Examples

**C/MQI**

```c
MQMD md = {MQMD_DEFAULT};
md.Expiry = 3000;   // Expire in 5 minutes
...
MQPUT(Hcon, Hobj, &md, &pmo, messlen, buffer, &CC, &RC);
```

**JMS**

```java
MessageProducer producer = session.createProducer(dest);
producer.setTimeToLive(300000);  // Expire in 5 minutes
```

**nodejs/MQ Light**

```javascript
var options = { ttl: 300000 };  // Expire in 5 minutes
client.send(topic, body, options, callback);
```
# Message Grouping

<table>
<thead>
<tr>
<th>MQ Wire Format</th>
<th>MQTT</th>
<th>HTTP</th>
<th>AMQP 1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MQI</th>
<th>MQ OO</th>
<th>MQTT</th>
<th>MQ HTTP</th>
<th>MQ Light</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Examples

### C/MQI

```c
pmo.Options = MQPMO_LOGICAL_ORDER;
...
md.MsgFlags = MQMF_MSG_IN_GROUP;       // For all but last msg
md.MsgFlags = MQMF_LAST_MSG_IN_GROUP;  // For last msg
MQPUT(Hcon, Hobj, &md, &pmo, messlen, buffer, &CC, &RC);
```

### JMS

```java
message.setStringProperty("JMSXGroupId", groupId);
message.setIntProperty("JMSXGroupSeq", 1);
message.setBooleanProperty("JMS_IBM_Last_Msg_In_Group", true);
```

### nodejs/MQ Light

N/A
### Message Segmentation

<table>
<thead>
<tr>
<th>MQ Wire Format</th>
<th>MQTT</th>
<th>HTTP</th>
<th>AMQP 1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MQI</th>
<th>MQ OO</th>
<th>MQTT</th>
<th>MQ HTTP</th>
<th>MQ Light</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>Some langs</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Notes:
- Not supported in most OO APIs
- Not supported on MQ z/OS
- Really a feature of the queue manager, not the API/lang

#### Examples

**C/MQI**

```c
md.MsgFlags = MQMF_SEGMENTATION_ALLOWED;
memcpy(md.GroupId, MQGI_NONE, MQ_GROUP_ID_LENGTH);
MQPUT(Hcon, Hobj, &md, &pmo, messlen, buffer, &CC, &RC);

gmo.Options = MQGMO_COMPLETE_MSG | <other options>;
MQGET(Hcon, Hobj, &md, &gmo, bufflen, buff, &msglen, &CC, &RC);
```

**JMS**

N/A

**nodejs/MQ Light**

N/A
Message Selection (1)

<table>
<thead>
<tr>
<th>MQ Wire Format</th>
<th>MQTT</th>
<th>HTTP</th>
<th>AMQP 1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MQI</th>
<th>MQ OO</th>
<th>MQTT</th>
<th>MQ HTTP</th>
<th>MQ Light</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notes:

- Use on MQOPEN to filter queued messages
- Applies to message properties

Examples

C/MQI

```c
od.SelectionString.VSPtr="Colour='red'";
od.SelectionString.VSLength=MQVS_NULL_TERMINATED;
MQOPEN(Hcon, &od, O_options, &Hobj, &CC, &RC);
```

JMS

```java
String selector = "NewsType = 'Sports'");
MessageConsumer consumer = 
    session.createConsumer(queue, selector);
```
	nodejs/

MQ Light → N/A
Message Selection (2)

<table>
<thead>
<tr>
<th>MQ Wire Format</th>
<th>MQTT</th>
<th>HTTP</th>
<th>AMQP 1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MQI</th>
<th>MQ OO</th>
<th>MQTT</th>
<th>MQ HTTP</th>
<th>MQ Light</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notes:
- Use on MQSUB to filter published messages
- Applies to message properties

Examples

C/MQI

```c
sd.SelectionString.VSPtr="Colour='red'";
sd.SelectionString.VSLength=MQVS_NULL_TERMINATED;
MQSUB(Hcon, &sd, &Hobj, &Hsub, &CompCode, &Reason);
```

JMS

```java
String selector = "NewsType = 'Sports'";
MessageConsumer consumer = 
    session.createConsumer(queue, selector);
```

nodejs/MQ Light

N/A
Local transactions

- Ability to put or get multiple messages to/from a destination, either all of them or none at all

- Messages are hidden from other applications until the transaction is committed.

- If a failure occurs mid-sequence, the transaction is rolled back and none of the messages are delivered. Other applications are never aware of the messages.

- If everything completes successfully, the messages are made available to other applications.
### Local transactions

<table>
<thead>
<tr>
<th>MQ Wire Format</th>
<th>MQTT</th>
<th>HTTP</th>
<th>AMQP 1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MQI</th>
<th>MQ OO</th>
<th>MQTT</th>
<th>MQ HTTP</th>
<th>MQ Light</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Notes:**
- Only queue manager resources involved
- JEE containers manage JMS transactions for you

**Examples**

**C/MQI**

```c
pmo.Options = MQPMO_SYNCPOINT;
MQPUT(Hcon, Hobj, &md, &pmo, messlen, buffer, &CC, &RC);
MQCMIT(Hcon, &CC, &RC);
```

**JMS**

```java
session = connection.createSession(true, Session.AUTO_ACKNOWLEDGE);
// Get and put messages
session.commit();
```

**node.js/MQ Light**

N/A
Global/XA transactions

- Similar to local transactions, but with multiple resources to update, e.g.
  - Application consumes 5 related messages from a queue and writes them all to a database
  - If everything happens successfully, the messages are guaranteed to be removed from the queue and written to the database
  - If a failure occurs, the messages are rolled back to the queue and none are in the database
    - Potential for a transaction to go in-doubt needing manual resolution
Global/XA transactions

<table>
<thead>
<tr>
<th>MQ Wire Format</th>
<th>MQTT</th>
<th>HTTP</th>
<th>AMQP 1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MQI</th>
<th>MQ OO</th>
<th>MQTT</th>
<th>MQ HTTP</th>
<th>MQ Light</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Examples

C/MQI

```c
MQBEGIN(Hcon, &bo, &CC, &RC);
// Read from database
pmo.Options = MQPMO_SYNCPOINT;
MQPUT(Hcon, Hobj, &md, &pmo, messlen, buffer, &CC, &RC);
MQCMIT(Hcon, &CC, &RC);
```

JMS

```java
session = connection.createXASession();
// Read from database, put messages
xaResource = session.getXAResource();
xaResource.prepare(xaId, false);
xaResource.commit(xaId, false);
```

nodejs/

MQ Light

N/A

Notes:

- Multiple resources coordinated together (e.g. MQ and Database)
- In this example the queue manager is coordinating the transaction
- Can be coordinated by DB/WAS etc.
- Can result in in-doubt transactions if a resource fails mid-transaction
- In JMS/JEE the container does the work
- MQ must be configured with the DB XA libraries
Quality of service

- Describes the level of assurance given that a message will be delivered

  - At most once delivery
    
    The message might not arrive, but it is guaranteed that you won’t receive duplicates

  - At least once delivery
    
    The message will definitely arrive, but you may receive duplicates

  - Exactly once delivery
    
    The message is guaranteed to be delivered once and exactly once.

    Often considered to be the ‘best’ option but brings with it various additional costs and complexities

    Where are you measuring from?
**At-most-once delivery**

<table>
<thead>
<tr>
<th>MQ Wire Format</th>
<th>MQTT</th>
<th>HTTP</th>
<th>AMQP 1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MQI</th>
<th>MQ OO</th>
<th>MQTT</th>
<th>MQ HTTP</th>
<th>MQ Light</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

**Examples**

**C/MQI**

```c
md.Persistence = MQPER_NOT_PERSISTENT;
...
MQPUT(Hcon, Hobj, &md, &pmo, messlen, buffer, &CC, &RC);
```

**JMS**

```javascript
producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);
session = connection.createSession(transacted, Session.AUTO_ACKNOWLEDGE);
producer.send(destination, message)
```

**nodejs/MQ Light**

```javascript
var options = { qos: mqlight.QOS_AT_MOST_ONCE };
client.send(topic, body, options, callback);
```
### At-least-once delivery

<table>
<thead>
<tr>
<th>MQ Wire Format</th>
<th>MQTT</th>
<th>HTTP</th>
<th>AMQP 1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MQI</th>
<th>MQ OO</th>
<th>MQTT</th>
<th>MQ HTTP</th>
<th>MQ Light</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

#### Notes:
- Can’t always just turn at-least-once ON with a single parameter
- Relates to several areas:
  - Reliability of network protocol
  - Persistence of messages
  - Durability of subscriptions
  - ACK features in client libraries

#### Examples

**C/MQI**

```c
md.Persistence = MQPER_PERSISTENT;
pmo.Options = MQPMO_SYNCPOINT;
MQPUT(Hcon, Hobj, &md, &pmo, messlen, buffer, &CC, &RC);
MQCMIT(Hcon, &CC, &RC);
```

**JMS**

```java
producer.setDeliveryMode(DeliveryMode.PERSISTENT);
session = connection.createSession(false,
                                      Session.CLIENT_ACKNOWLEDGE);
message.acknowledge();
```

**nodejs/ MQ Light**

```javascript
var options = { qos: mqlight.QOS_AT_LEAST_ONCE };
client.send(topic, body, options, callback);
```
Exactly-once delivery

<table>
<thead>
<tr>
<th>MQ Wire Format</th>
<th>MQTT</th>
<th>HTTP</th>
<th>AMQP 1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MQI</th>
<th>MQ OO</th>
<th>MQTT</th>
<th>MQ HTTP</th>
<th>MQ Light</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Examples

C/MQI

```
MQBEGIN(Hcon, &bo, &CC, &RC);
pmo.Options = MQPMO_SYNCPOINT;
MQPUT(Hcon, Hobj, &md, &pmo, messlen, buffer, &CC, &RC);
MQCMIT(Hcon, &CC, &RC);
```

JMS

```
session = connection.createXASession();
// Put messages
xaResource = session.getXAResource();
xaResource.prepare(xaId, false);
xaResource.commit(xaId, false);
```

nodejs/
MQ Light → N/A

Notes:
- Requires XA/Global transactions
- Useful if messages **MUST** arrive and **MUSTN’T** be duplicated, but:
  - Network flow more complicated
  - Might require a human to resolve in-doubt transactions
### HA Failover/Auto-Reconnect

<table>
<thead>
<tr>
<th>MQ Wire Format</th>
<th>MQTT</th>
<th>HTTP</th>
<th>AMQP 1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MQI</th>
<th>MQ OO</th>
<th>MQTT</th>
<th>MQ HTTP</th>
<th>MQ Light</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

#### Notes:
- Really a feature of the client library rather than the wire format or API

#### Client

<table>
<thead>
<tr>
<th>Client</th>
<th>Built-in HA reconnect?</th>
</tr>
</thead>
<tbody>
<tr>
<td>C MQI Client</td>
<td>✓</td>
</tr>
<tr>
<td>JMS MQ Client</td>
<td>✓</td>
</tr>
<tr>
<td>Cobol MQI Client</td>
<td>✓</td>
</tr>
<tr>
<td>MQ Base Java Client</td>
<td>×</td>
</tr>
<tr>
<td>C++ MQ Client</td>
<td>✓</td>
</tr>
<tr>
<td>XMS C/C++.Net</td>
<td>✓</td>
</tr>
<tr>
<td>VB</td>
<td>✓</td>
</tr>
<tr>
<td>.Net WCF</td>
<td>×</td>
</tr>
<tr>
<td>MQ Light Node/Ruby/Java/Python clients</td>
<td>✓</td>
</tr>
</tbody>
</table>
### Message properties

<table>
<thead>
<tr>
<th>MQI</th>
<th>MQ OO</th>
<th>MQTT</th>
<th>MQ HTTP</th>
<th>MQ Light</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>Not base java</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

### Asynchronous consume

<table>
<thead>
<tr>
<th>MQI</th>
<th>MQ OO</th>
<th>MQTT</th>
<th>MQ HTTP</th>
<th>MQ Light</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>Not base java or .Net/C#</td>
<td>some langs</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

### Message browsing

<table>
<thead>
<tr>
<th>MQI</th>
<th>MQ OO</th>
<th>MQTT</th>
<th>MQ HTTP</th>
<th>MQ Light</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✔</td>
<td></td>
</tr>
</tbody>
</table>
Thank You - Questions?
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.
Trademark Statement

- IBM and the IBM logo are trademarks of International Business Machines Corporation, registered in many jurisdictions. Other marks may be trademarks or registered trademarks of their respective owners.
- Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.
- Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.
- Red Hat Enterprise Linux is a registered trademark of Red Hat, Inc. in the United States and other countries.
- Ubuntu and Canonical are registered trademarks of Canonical Ltd.
- SUSE and SLES are registered trademarks of SUSE LLC in the United States and other countries.
- Mac and OS X are trademarks of Apple Inc., registered in the U.S. and other countries.
- Other company, product and service names may be trademarks, registered marks or service marks of their respective owners.
- References in this publication to IBM products and services do not imply that IBM intends to make them available in all countries in which IBM operates.