MQ Tools For Your MQ Toolkit

Tim Zielke

Introduction and Agenda

My Background:

- I have been in IT for 20 years with Hewitt Associates/Aon Hewitt/Alight Solutions
- First 13 years mainly on the mainframe COBOL application side
- Last 7 years as a CICS/MQ systems programmer

Session Agenda:

We will cover the following topics that include free tools from the MH06 Trace Tools supportpac:

- Client mode of runmqsc and mqsc-qmgrs Unix script
- MQ strmqtrc API tracing and mqtrcfrmt
- Application Activity Trace and amqsactz

runmqsc Client Mode

- Starting at MQ v8, IBM has provided a client mode enhancement to runmqsc that allows you to make a client connection to an MQ queue manager.

```
Example:
server1 - MQ v8 client is installed
server2 - MQ queue manager QM2 is running

On server1:
- CCDT file CCDT.TAB is located in /home/mqm/client and has an entry for an
encrypted channel connection to QM2 on server2
- SSL keystore is in /home/mqm/client/ssl.

Execute on server1:
> export MQCHLLIB=/home/mqm/client
> export MQCHLTAB=CCDT.TAB
> export MQSSLKEYR=/home/mqm/client/ssl/key
> runmqsc -c QM2
5724-H72 (C) Copyright IBM Corp. 1994, 2015.
Starting MQSC for queue manager QM2.

<- Prompt here! We are connected to QM2 on server2 with runmqsc!
```

mqsc-qmgrs Unix script in MH06

- This runmqsc client mode enhancement is a nice enhancement, but wouldn't it also be helpful if you could do a "runmqsc -c" against a group of queue managers and have the output for each resource be on one line for easy data mining with a tool like grep? This is where mqsc-qmgrs comes in.

- mqsc-qmgrs is a Unix script that comes with the MH06 (Trace Tools) supportpac that has a wrapper around "runmqsc -c" to accomplish the above enhancements. Here are the main benefits of using mqsc-qmgrs:

1) It can be run against a group of queue managers.
2) Each invocation of "runmqsc -c" runs asynchronously on its own pid for better performance.
3) The runmqsc results for the different queue managers are collected into one result file.
4) Another one line summary result file is also created that contains each multi-line resource result (e.g. DIS QL(*) ALL) collapsed into one line for easy data mining with a tool like grep. Each result line also has the queue manager it is associated with prepended to the line, again for ease of searching the result data.

mqsc-qmgrs – One Line Summary

If you run the runmqsc command “DIS CHL(*) ALTDATE ALTTIME BATCHHB BATCHINT” against QM1, you would get back results formatted onto multiple lines like below:

```
AMQ8414: Display Channel details.
   CHANNEL(CL.2S.SERVER1)                  CHLTYPE(CLUSRCVR)
   ALTDATE(2016-03-11)                     ALTTIME(02.08.00)
   BATCHHB(1000)                           BATCHINT(0)
```

The one line summary result file would take a multi-line channel result like above, and parse it into one line with the queue manager prepended to the line:

```
QM1. AMQ8414: Display Channel details.  CHANNEL(CL.2S.SERVER1)
CHLTYPE(CLUSRCVR)  ALTDATE(2016-03-11)
ALTTIME(02.08.00)  BATCHHB(1000)
```

Again, this aids in the ability to grep the result data for runmqsc.

mqsc-qmgrs - Usage Notes

1) mqsc-qmgrs requires an input of a queue manager group and a unique identifier text.

A queue manager group contains a list of queue managers. The queue manager groups must also be defined near the top of the script.

```
# Define qmgr groups here
ALL="QM1 \
     QM2 \
     QM3"
```

The unique identifier text allows multiple users to run mqsc-qmgrs in the same working directory. For this doc, we will assume TIMZ is used.

```
Example: mqsc-qmgrs ALL TIMZ
```

2) mqsc-qmgrs must be executed in a directory that includes the file mqsc-qmgrs-TIMZ-input. This file will include the runmqsc commands to execute.

```
Example: mqsc-qmgrs-TIMZ-input might include the following:
DIS Q(*) ALL
```

mqsc-qmgrs - Usage Notes

3) mqsc-qmgrs will produce a file in the CWD called mqsc-qmgrs-TIMZ-output-all, which includes all the runmqsc output from the queue manager group appended together.

4) mqsc-qmgrs will produce a file in the CWD called mqsc-qmgrs-TIMZ-output-all-1LS, which has the individual results put into one line with the queue manager name prepended to the line.

mqsc-qmgrs-TIMZ-output-all contains multi-line result output like runmqsc:

```
AMQ8414: Display Channel details.
   CHANNEL(CL.2S.SERVER1)                  CHLTYPE(CLUSRCVR)
   ALTDATE(2016-03-11)                     ALTTIME(02.08.00)
   BATCHHB(1000)                           BATCHINT(0)
```

mqsc-qmgrs-TIMZ-output-all-1LS has each result summarized onto one line with qmgr prepended:

```
QM1. AMQ8414: Display Channel details.  CHANNEL(CL.2S.SERVER1)
CHLTYPE(CLUSRCVR)  ALTDATE(2016-03-11)                    ALTTIME(02.08.00)
BATCHHB(1000)
```

mqsc-qmgrs – Setting It Up

- You need a Unix server that mqsc-qmgrs can run on that has remote connectivity to the queue manager servers that you want to access.
- MQ v8 or higher MQ Client software installed on this Unix server.
- CCDT for the queue managers that you will be accessing with mqsc-qmgrs. At MQ v8 and higher, “runmqsc -n” option now supports the ability to edit CCDT files.
- Set up equivalent SVRCONN channels on the queue managers you want to access.
- Set up environment variables similar to the following, and you are now ready to use mqsc-qmgrs!

```
> export MQCHLLIB=/home/mqm/client
> export MQCHLTAB=CCDT.TAB
> export MQSSLKEYR=/home/mqm/client/ssl/key
```

mqsc-qmgrs - Use Cases

- Helpful for MQ administrative analysis. For example, you are moving a queue manager from server1 to server2, and you need to know all the queue managers that know about a queue named APP.Q1. You can use mqsc-qmgrs to run the commands “DIS Q(APP.Q1)” and “DIS QC(APP.Q1)” against the group of relevant queue managers to find out this information.

- Helpful for MQ administration. For example, you want all your queue managers to have the setting MONQ(LOW), you can run the command “ALTER QMGR MONQ(LOW)” against all of your queue managers with mqsc-qmgrs, and get back all the results summarized into one result file.

- Helpful for MQ monitoring. For example, a user has reported performance issues with a queue named APP.Q1, and you want to iteratively run a “DIS QS(APP.Q1) TYPE(QUEUE) ALL” against the 4 queue managers that this queue is defined on to see how this queue is performing. mqsc-qmgrs gives you a single location to perform this and also get the results back in a summarized fashion.

MQ strmqtrc API Tracing - Overview

Overview:

strmqtrc tracing is a troubleshooting tool that comes with distributed WebSphere MQ. A strmqtrc API trace (strmqtrc -t api) of an MQ application will include all of the API calls (i.e. MQOPEN, MQPUT, etc.) that the application makes, including the before and after of each API call. The before (denoted with the >> symbol in the trace) has the input data that application is passing into MQ. The after (denoted with the << symbol in the trace) has the return data that MQ is returning to the application.

This API data is very helpful in MQ problem determination, as it allows you to see in detail all of the input data that your application is passing to MQ and what return data your application is getting back from MQ. The MQ API trace can be cryptic to read, but we will cover a MH06 supportpac trace tool (mqtrcfrmt) that can significantly aid in reading MQ API traces.

MQ API Tracing – amqsput on Linux x86

- Turn on an API trace for the amqsput program

```
strmqtrc -m qmgr -t api -p amqsput
```

- Run the amqsput program on a TCZ.TEST1 queue, and do two PUTs to the queue, and then end the program.

- NOTE: By default, trace writes out on Linux to a file like:

```
/var/mqm/trace/AMQ16884.0.TRC (where 16884 = pid)
```

- Turn off the tracing

```
endmqtrc -a
```

- Format the trace (this step is not needed for Windows traces)

```
dspmqtrc AMQ16884.0.TRC > AMQ16884.0.FMT
```

Reading a strmqtrc API Trace

- We will now look at the AMQ16884.0.FMT trace. The following slides will contain pieces of that trace, that we will look at in further detail. Note that some of the extraneous trace data has been edited, so that it can fit on the slide.

AMQ16884.0.FMT – Header

- Lines 3 – 27 have the trace header information, with key environmental and application information.

```
1    WebSphere MQ Formatted Trace - Formatter V3
2
3     +-----------------------------------------------------------------------+
4     |                                                                       |
5     | WebSphere MQ Formatted Trace V3                                       |
6     | ==============================                                        |
7     |                                                                       |
8     | Date/Time          :- 06/30/14  13:15:37 CST                          |
9     | UTC Time           :- 1404152137.740853                               |
10    | UTC Time Offset    :- 5 (CST)                                         |
11    | Host Name          :- MYSERVER123                                     |
12    | Operating System   :- Linux 2.6.32.59-0.7-default                     |
13    | LVLS               :- 7.5.0.3                                         |
14    | Product Long Name  :- WebSphere MQ for Linux (x86-64 platform)        |
15    | Build Level        :- p750-003-140123                                 |
16    | Installation Path  :- /opt/mqm                                        |
17    | Installation Name  :- Installation1     (1)                           |
18    | License Type       :-                                                 |
19    | Effective UserID   :- 244 (mqm)                                       |
20    | Real UserID        :- 244 (mqm)                                       |
21    | Program Name       :- amqsput                                         |
22    | Addressing Mode    :- 64-bit                                          |
23    | Process            :- 16884                                           |
24    | QueueManager       :- MYSERVER123!MQTEST1                             |
25    | Reentrant          :- 1                                               |
26    |                                                                       |
27    +-----------------------------------------------------------------------+
```

AMQ16884.0.FMT – Columns/API after

- Line 33 shows the column headers which include a microsecond time stamp, process.thread, and API trace data.
- Lines 36 – 46 show an MQCONN after. Remember that after means that this is the data being returned from MQ to the application, when the API call has ended. This is denoted by the << on line 36. From the end of this call, we can see that it was successful (Compcode and Reason were zero), and that an Hconn or connection handle was returned (x'06004001'). Use this Hconn value for the subsequent API calls to follow the connection activity for this specific connection.

```
33   Timestamp         Process.Thread  Trace Ident  Trace Data
34   ============================================================
35   *13:15:37.742821   16884.1       CONN:1400006  __________
36    13:15:37.742829   16884.1       CONN:1400006  MQCONN <<
37    13:15:37.742831   16884.1       CONN:1400006  Name          : Input  Parm
38    13:15:37.742832   16884.1       CONN:1400006  Hconn:
39    13:15:37.742834   16884.1       CONN:1400006    0x0000:  06004001                      |..@.            |
40    13:15:37.742835   16884.1       CONN:1400006  ConnectOpts:
41    13:15:37.742837   16884.1       CONN:1400006    0x0000:  434e4f20 01000000 00010000   |CNO ........    |
42    13:15:37.742838   16884.1       CONN:1400006  Compcode:
43    13:15:37.742840   16884.1       CONN:1400006    0x0000:  00000000                      |....            |
44    13:15:37.742841   16884.1       CONN:1400006  Reason:
45    13:15:37.742842   16884.1       CONN:1400006    0x0000:  00000000                      |....            |
46    13:15:37.742846   16884.1       CONN:1400006  MQI:MQCONN HConn=01400006 rc=00000000
```

AMQ16884.0.FMT – MQOPEN before

- Lines 48 – 67 are an MQOPEN before. Remember that before means that this is the data being passed from the application to MQ, when the call was initiated. This is denoted by the >> on line 48. The inputs being passed in are the Hconn, Objdesc, Options, Hobj, Compcode, Reason. Note that some data (i.e. ObjDesc) is both input and output data. Options is just input data. Compcode is just output data.

- Note for the Objdesc (lines 51 - 62), this MQ API data structure is printed in the raw hex data format, with each 16 byte line formatted to ASCII directly to the right.

```
48   13:15:37.742881    16884.1   MQOPEN >>
49   13:15:37.742882    16884.1   Hconn:
50   13:15:37.742884    16884.1     0x0000:  06004001                          |..@.            |
51   13:15:37.742885    16884.1   Objdesc:
52   13:15:37.742887    16884.1     0x0000:  4f442020 01000000 01000000 54435a2e  |OD  ........TCZ.|
53   13:15:37.742887    16884.1     0x0010:  54455354 31000000 00000000 00000000  |TEST1...........|
54   13:15:37.742887    16884.1     0x0020:  00000000 00000000 00000000 00000000  |................|
55   13:15:37.742887    16884.1     0x0030:  00000000 00000000 00000000 00000000  |................|
56   13:15:37.742887    16884.1     0x0040:  00000000 00000000 00000000 00000000  |................|
57   13:15:37.742887    16884.1     0x0050:  00000000 00000000 00000000 00000000  |................|
58   13:15:37.742887    16884.1     0x0060:  00000000 00000000 00000000 414d512e  |............AMQ.|
59   13:15:37.742887    16884.1     0x0070:  2a000000 00000000 00000000 00000000  |*...............|
60   13:15:37.742887    16884.1     0x0080:  00000000 00000000 00000000 00000000  |................|
61   13:15:37.742887    16884.1     0x0090:  00000000 00000000 00000000 00000000  |................|
62   13:15:37.742887    16884.1     0x00a0:  00000000 00000000                    |........        |
63   13:15:37.742888    16884.1   Options:
64   13:15:37.742889    16884.1     0x0000:  10200000                          |. ..            |
65   13:15:37.742891    16884.1   Hobj          : Output Parm
66   13:15:37.742892    16884.1   Compcode      : Output Parm
67   13:15:37.742894    16884.1   Reason        : Output Parm
```

AMQ16884.0.FMT – MQOPEN after

- Lines 69 – 89 are an MQOPEN after. Note that we now have a returned Hobj, Compcode, and Reason from the MQOPEN call.
- The API trace would then contain the rest of our amqsput API calls (i.e. MQPUTs, MQCLOSE, etc.)

```
69   13:15:37.743138   16884.1   MQOPEN <<
70   13:15:37.743141   16884.1   Hconn          : Input  Parm
71   13:15:37.743142   16884.1   Objdesc:
72   13:15:37.743144   16884.1     0x0000:  4f442020 01000000 01000000 54435a2e  |OD  ........TCZ.|
73   13:15:37.743144   16884.1     0x0010:  54455354 31000000 00000000 00000000  |TEST1...........|
74   13:15:37.743144   16884.1     0x0020:  00000000 00000000 00000000 00000000  |................|
75   13:15:37.743144   16884.1     0x0030:  00000000 00000000 00000000 00000000  |................|
76   13:15:37.743144   16884.1     0x0040:  00000000 00000000 00000000 00000000  |................|
77   13:15:37.743144   16884.1     0x0050:  00000000 00000000 00000000 00000000  |................|
78   13:15:37.743144   16884.1     0x0060:  00000000 00000000 00000000 414d512e  |............AMQ.|
79   13:15:37.743144   16884.1     0x0070:  2a000000 00000000 00000000 00000000  |*...............|
80   13:15:37.743144   16884.1     0x0080:  00000000 00000000 00000000 00000000  |................|
81   13:15:37.743144   16884.1     0x0090:  00000000 00000000 00000000 00000000  |................|
82   13:15:37.743144   16884.1     0x00a0:  00000000 00000000                    |........        |
83   13:15:37.743145   16884.1   Options        : Input  Parm
84   13:15:37.743151   16884.1   Hobj:
85   13:15:37.743153   16884.1     0x0000:  02000000                            |....            |
86   13:15:37.743154   16884.1   Compcode:
87   13:15:37.743156   16884.1     0x0000:  00000000                            |....            |
88   13:15:37.743157   16884.1   Reason:
89   13:15:37.743158   16884.1     0x0000:  00000000                            |....            |
```

mqtrcfrmt tool in MH06

- mqtrcfrmt is a trace tool that comes with the MH06 supportpac. It will help you read a trace by expanding the MQ data structures and fields in the trace into human readable formats with MQ constant expansions included. Executables from mqtrcfrmt are provided for Linux x86, Solaris SPARC, and Windows.

- Example of using the mqtrcfrmt tool to expand our trace:

```
mqtrcfrmt.linux AMQ16884.0.FMT AMQ16884.0.FMT2
```

mqtrcfrmt - AMQ16884.0.FMT2

```
59   13:15:37.742885    16884.1   Objdesc:
60   13:15:37.742887    16884.1     0x0000:  4f442020 01000000 01000000 54435a2e  |OD  ........TCZ.|
61   13:15:37.742887    16884.1     0x0010:  54455354 31000000 00000000 00000000  |TEST1...........|
62   13:15:37.742887    16884.1     0x0020:  00000000 00000000 00000000 00000000  |................|
63   13:15:37.742887    16884.1     0x0030:  00000000 00000000 00000000 00000000  |................|
64   13:15:37.742887    16884.1     0x0040:  00000000 00000000 00000000 00000000  |................|
65   13:15:37.742887    16884.1     0x0050:  00000000 00000000 00000000 00000000  |................|
66   13:15:37.742887    16884.1     0x0060:  00000000 00000000 00000000 414d512e  |............AMQ.|
67   13:15:37.742887    16884.1     0x0070:  2a000000 00000000 00000000 00000000  |*...............|
68   13:15:37.742887    16884.1     0x0080:  00000000 00000000 00000000 00000000  |................|
69   13:15:37.742887    16884.1     0x0090:  00000000 00000000 00000000 00000000  |................|
70   13:15:37.742887    16884.1     0x00a0:  00000000 00000000                    |........        |
71                      16884.1        Objdesc expanded (all fields):
72                      16884.1        StrucId (CHAR4)              : 'OD  '
73                      16884.1                                       x'4f442020'
74                      16884.1        Version (MQLONG)             : 1
75                      16884.1                                       x'01000000'
76                      16884.1        ObjectType (MQLONG)          : 1
77                      16884.1                                       x'01000000'
78                      16884.1           ObjectType MQOT_Q
79                      16884.1        ObjectName (MQCHAR48)        : 'TCZ.TEST1'
80                      16884.1                                       x'54435a2e544553543100 . . . 00'
81                      16884.1        ObjectQMgrName (MQCHAR48)    : '........'
82                      16884.1                                       x'0000000000000000000 . . . 00'
83                      16884.1        DynamicQName (MQCHAR48)      : 'AMQ.*
84                      16884.1                                       x'414d512e2a0000000000 . . . 00'
85                      16884.1        AlternateUserId (MQCHAR12)   : '............'
86                      16884.1                                       x'000000000000000000000000'
```

mqtrcfrmt - AMQ16884.0.FMT2 - cont

```
347   13:15:40.064569   16884.1   Putmsgopts:
348   13:15:40.064570   16884.1     0x0000:  504d4f20 01000000 04200000 ffffffff  |PMO ..... ......|
349   13:15:40.064570   16884.1     0x0010:  00000000 01000000 00000000 00000000  |................|
350   13:15:40.064570   16884.1     0x0020:  54435a2e 54455354 31202020 20202020  |TCZ.TEST1       |
351   13:15:40.064570   16884.1     0x0030:  20202020 20202020 20202020 20202020  |                |
352   13:15:40.064570   16884.1     0x0040:  20202020 20202020 20202020 20202020  |                |
353   13:15:40.064570   16884.1     0x0050:  4d595345 52564552 3132332e 4d515445  |MYSERVER123.MQTE|
354   13:15:40.064570   16884.1     0x0060:  53543120 20202020 20202020 20202020  |ST1             |
355   13:15:40.064570   16884.1     0x0070:  20202020 20202020 20202020 20202020  |                |
356                     16884.1        Putmsgopts expanded (all fields):
357                     16884.1        StrucId (CHAR4)              : 'PMO '
358                     16884.1                                       x'504d4f20'
359                     16884.1        Version (MQLONG)             : 1
360                     16884.1                                       x'01000000'
361                     16884.1        MQPMO.Options= (MQLONG)      : 8196
362                     16884.1                                       x'04200000'
363                     16884.1           Options=MQPMO_NO_SYNCPOINT
364                     16884.1           Options=MQPMO_FAIL_IF_QUIESCING
365                     16884.1        Timeout (MQLONG)             : -1
366                     16884.1                                       x'ffffffff'
367                     16884.1        Context (MQLONG)             : x'00000000'
368                     16884.1        KnownDestCount (MQLONG)      : 1
369                     16884.1                                       x'01000000'
370                     16884.1        UnknownDestCount (MQLONG)    : 0
371                     16884.1                                       x'00000000'
372                     16884.1        InvalidDestCount (MQLONG)    : 0
373                     16884.1                                       x'00000000'
374                     16884.1        ResolvedQName (MQCHAR48)     : 'TCZ.TEST1'
375                     16884.1                                       x'54435a2e544553543120 . . . 20'
376                     16884.1        ResolvedQMgrName (MQCHAR48)  : 'MYSERVER123.MQTEST1
```

mqtrcfrmt – API Summary Trace

- A User customizable API summary trace can also be generated from our mqtrcfrmt expanded AMQ16884.0.FMT2 trace by pulling out key lines from the trace.

```
egrep '( >>$| <<$|Hconn=|Hobj=|Compcode=|Reason=|Hmsg=|Actual Name=|Value=|Options=|Type=|ObjectName
|ResolvedQName |Persistence )' AMQ16884.0.FMT2


 13:15:37.742829    16884.1       CONN:1400006      MQCONN <<
                    16884.1                              Hconn=06004001
                    16884.1                              MQCNO.Options= (MQLONG)      : 256
                    16884.1                                 Options=MQCNO_SHARED_BINDING
                    16884.1                              Compcode=0
                    16884.1                              Reason=0
 13:15:37.742881    16884.1       CONN:1400006      MQOPEN >>
                    16884.1                              Hconn=06004001
                    16884.1                              ObjectName (MQCHAR48)        :
'TCZ.TEST1.......................................'
                    16884.1                              MQOO.Options= (MQLONG)       : 8208
                    16884.1                                 Options=MQOO_OUTPUT
                    16884.1                                 Options=MQOO_FAIL_IF_QUIESCING
 13:15:37.743138    16884.1       CONN:1400006      MQOPEN <<
                    16884.1                              ObjectName (MQCHAR48)        :
'TCZ.TEST1.......................................'
                    16884.1                              Hobj=02000000
                    16884.1                              Compcode=0
                    16884.1                              Reason=0
 13:15:37.743176    16884.1       CONN:1400006      MQI:MQOPEN HConn=01400006 HObj=00000002 rc=00000000
ObjType=00000001 ObjName=TCZ.TEST1
```

dspmqtrc - API Summary Trace

dspmqtrc is also inserting one line summary API lines with an “MQI:” text. You can grep lines that have “MQI:” out of a formatted strmqtrc to get an API summary.

At MQ v8:

```
16394.1 MQI:MQCONN HConn=01400006 rc=00000000
16394.1 MQI:MQOPEN HConn=01400006 HObj=00000002 rc=00000000 ObjType=00000001 ObjName=TCZ.TEST1
16394.1 MQI:MQPUT HConn=01400006 HObj=00000002 BufLen=00000032 rc=00000000
16394.1 MQI:MQCLOSE HConn=01400006 HObj=00000002
16394.1 MQI:MQDISC HConn=01400006
```

At MQ v9 (Notice SYNCP and PERS have now been added at v9!):

```
23284.1  MQI:MQCONN HConn=01400006 rc=00000000
23284.1  MQI:MQOPEN HConn=01400006 HObj=00000002 rc=00000000 ObjType=00000001 ObjName=TCZ.TEST1
23284.1  MQI:MQPUT HConn=01400006 HObj=00000002 BufLen=00000005 rc=00000000 SYNCP(NO) PERS(NO)
23284.1  MQI:MQCLOSE HConn=01400006 HObj=00000002
23284.1  MQI:MQDISC HConn=01400006
```

mqtrcfrmt – Message Parsing

mqtrcfrmt program has the ability to message parse or analyze an MQ message in a strmqtrc (or amqsact activity trace) as if it was 1208 (UTF-8) or 1200 (UTF-16). This can be helpful to validate if a message is being accurately labeled with the CCSID or Encoding, investigating data conversion issues, etc. See my MQTC 2016 Data Conversion session for more details about message parsing.

```
# run trace with -d all option to capture message data
> strmqtrc -m QM1 -t api -d all -p mypgmname

# For some platforms, use dspmqtrc to format the trace
> dspmqtrc AMQ12345.0.TRC > AMQ12345.0.FMT

# mqtrcfrmt program with -m message parsing option to byte analyze message as 1208
> mqtrcfrmt.linux AMQ12345.0.FMT AMQ12345.0.FMT2 -m 1208

Inside AMQ12345.0.FMT2 (we have a message of "niño niño" in UTF-8):
08:06:08.803334    23401.1     Buffer:
08:06:08.803338    23401.1       0x0000:  6e69c3b1 6f206e69 c3b16f        |ni..o ni..o     |
msg-parser  UTF-8 Totals: Line:366 Pid:23401.1 Format:MQSTR    CCSID:1208 API:MQGET <<
Byte:11 ASCII:7 MB2:2 MB3:0 MB4:0 Inv:0
msg-parser Byte Analysis: Line:366 2-MB2, 8-MB2
```

mqtrcfrmt – Message Search

mqtrcfrmt provides the ability to search for a text (in hex or string) of a message in a strmqtrc trace (or amqsact activity trace), and report the offset of where the match was detected. The message body in a trace appears 16 characters per trace line, so this provides a way to match on search strings that are broken over one or many lines.

In a trace (with the strmqtrc -d option), you could have a message traced as follows. A grep search for the string “dog” would be a miss, since “dog” is broken up over two lines.

```
Buffer:
  0x0000:  54686520 71756963 6b206272 6f776e20  |The quick brown |
  0x0010:  666f7820 6a756d70 6564206f 76657220  |fox jumped over |
  0x0020:  74686520 736c6f77 206c617a 7920646f  |the slow lazy do|
  0x0030:  67                                   |g               |
```

Using mqtrcfrmt, you can run the following command to match on the string “dog”:

```
./mqtrcfrmt.linux AMQ12345.0.FMT AMQ12345.0.FMT2 -s “dog”
```

There would then be the following text inserted after the message in the FMT2 file:

```
msgSearch: hit at message offset 2e
```

mqtrcfrmt – msg2File

If you insert a special tag “msg2File-” above a “Buffer:” line in a strmqtrc trace or a “Message Data:” line in an amqsact activity trace, the mqtrcfrmt program will write the bytes of the message to a file (max length of file name is 20) whose name follows the “msg2File-“ tag.

For example, if you add this msg2File line before a message Buffer in strmqtrc:

```
14:49:27.664265 29003.1 CONN:1400006 msg2File-file1
14:49:27.664265 29003.1 CONN:1400006 Buffer:
14:49:27.664269 29003.1 CONN:1400006 0x0000: 0066006F 0078D801 DC37 |.............|
```

then a file called file1 will be written out in your current directory that contains the bytes of the message in the Buffer.

As a convenience, a Java (1.5 compiled) MQFile2Msg.class executable is provided to be able to take a file like the one that msg2File will produce and PUT it back to a queue.

This functionality allows you to capture and reuse messages without having to stop running MQ applications.

mqapitrcstats - Trace Performance Tool

- API tracing provides microsecond timings in the trace record. By finding the API begin (i.e. MQGET >>) and the API end (i.e. reason field of MQGET <<) you can roughly calculate the time it took for the API MQGET to complete. Do note that tracing does add overhead to the timings, but this can still be helpful for diagnosing gross performance issues.

```
48      ------------>  13:15:37.742881    16884.1       CONN:1400006      MQOPEN >>

69                     13:15:37.743138    16884.1       CONN:1400006      MQOPEN <<

88                     13:15:37.743157    16884.1       CONN:1400006      Reason:
89      ------------>  13:15:37.743158    16884.1       CONN:1400006        0x0000:  00000000

13:15:37.743158 - 13:15:37.742881 = 0.000277 seconds to complete for the MQOPEN
```

- mqapitrcstats tool in the MH06 Trace Tools supportpac will read an entire API trace and create a summary report of the response times of the open, close, get, put, and put1 API calls. Executables are provided for Linux x86, Solaris Sparc, and Windows.

MQOptions and mqidecode

- The MH06 supportpac has a Java tool called MQOptions that can help with deciphering many MQ option fields.

```
>java MQOptions

The current platform you are running on is Little-endian.
IMPORTANT: You may need to first reverse the bytes of your options value, depending on the endianness of this field!
Refer to the MQOptions manual for more information on endianness, if needed.

Enter your options field (conn, open, get, put, close, cbd, sub, subrq, report)  open
Enter your value (i.e. 8208 or 0x00002010)  8208

open options for decimal value 8208 and hex value 0x2010 converts to:
0x00000010  MQOO_OUTPUT
0x00002000  MQOO_FAIL_IF_QUIESCING
```

- Another tool that can do this is the mqidecode tool in the MS0P (WebSphere MQ Explorer Extended Management Plug-ins) supportpac. mqidecode can also decode many other fields, besides option fields.

```
>mqidecode -p MQOO -v 8208
MQOO_OUTPUT (0x00000010)
MQOO_FAIL_IF_QUIESCING (0x00002000)
```

Application Activity Trace

- The Application Activity Trace was first introduced in 7.1. It provides detailed information of the behavior of applications connected to a queue manager, including their MQI (or API) call details.

- The Activity Trace is another tool that can be helpful in MQ problem determination or application review, by giving you visibility to the inputs and outputs of your application API calls. It is also more user friendly than strmqtrc tracing.

Activity Trace – Usage Notes

- Applications write Activity Trace records to the SYSTEM.ADMIN.TRACE.ACTIVITY.QUEUE.

- There is a hierarchy to turning ON/OFF the Activity Trace:

1) Globally with ACTVTRC queue manager attribute (ON/OFF) (overridden by 2)
2) MQCNO_ACTIVITY_TRACE connection options specified in an MQCONNX. ACTVCONO queue manager attribute must be ENABLED for this to be checked, and the default value is DISABLED. (overridden by 3)
3) Settings in a matching stanza in mqat.ini (located in qm.ini directory)

- For example, you could have the ACTVTRC(OFF), but that is overridden to ON by the application specifying the MQCNO_ACTIVITY_TRACE_ENABLED option on the MQCONNX, but that is overridden back to OFF with the mqat.ini having a stanza to turn off the Activity Trace for this application. The net result is that the Activity Trace if OFF for this application.

- At MQ v9, the Activity Trace also supports the ability to subscribe to system topics to get activity trace data. For example, you can use the amqsact program to subscribe for activity trace messages for a given application name, channel, or connection id. However, this session will focus on the global approach, as described above.

mqat.ini

```
#*******************************************************************#
#* Module Name: mqat.ini                                           *#
#* Type       : IBM MQ queue manager configuration file            *#
#  Function   : Define the configuration of application activity   *#
#*              trace for a single queue manager.                  *#
#*******************************************************************#

# Global settings stanza, default values
AllActivityTrace:
  ActivityInterval=1
  ActivityCount=100
  TraceLevel=MEDIUM
  TraceMessageData=0
  StopOnGetTraceMsg=ON
  SubscriptionDelivery=BATCHED

# Prevent the sample activity trace program from generating data
ApplicationTrace:
  ApplName=amqsact*
  Trace=OFF
```

mqat.ini – Usage Note

- In order to pick up an mqat.ini change dynamically in a running program, you need to alter a queue manager attribute (i.e. alter the DESCR field) for the running program to pick up the change in the mqat.ini file.

- Use Case Example:

You turn on activity tracing for a program named mqapp1 by updating the appropriate stanza in the mqat.ini file, and then your start the mqapp1 program. After you have collected your desired activity trace data for mqapp1, you update the mqat.ini file to have the activity trace turned off for mqapp1. The mqapp1 program also continues to run. However, what you observe is that the mqapp1 program continues to write out activity trace messages, even though you turned it off in the mqat.ini file! You then do an alter of a queue manager attribute, and now the activity trace turns off for the mqapp1 program.

Activity Trace – Viewing the Data

- MS0P supportpac (WebSphere MQ Explorer Extended Management Plug-ins) has an Application Activity Trace viewer.

- amqsact is an IBM supplied command line tool (sample code also provided) that can read the messages from the SYSTEM.ADMIN.TRACE.ACTIVITY.QUEUE and format them into a human readable text file.

- amqsactz in the MH06 supportpac is a program that takes the amqsact sample code and provides some of the following enhancements:

1. Includes -r option for application summary reports that show what objects were used, what channels, what API operations were performed, what reason codes were returned, etc.
2. Generate a trace of one line API calls. This allows you to more easily follow the API flow of an application. The API data field items that appear in this output can also be customized.
3. Abstraction of connection id for improved readability. The connection id (e.g. 554F108A2061F801) can be hard to read and differentiate in a trace. This feature will abstract each unique connection id to a more readable number (e.g. 1 instead of 554F108A2061F801) in the trace of one line API calls.

amqsactz - Usage Example

1) Turn on the Activity Trace globally by doing ALTER QMGR ACTVTRC(ON).
2) Let Activity Trace data collect on the SYSTEM.ADMIN.TRACE.ACTIVITY.QUEUE for several minutes. We will also use amqsput to generate some PUTs to a TCZ.TEST1 queue.
3) Turn off the Activity Trace by doing ALTER QMGR ACTVTRC(OFF).

Now we will use amqsactz to view the Activity Trace data in three files.

1. amqsactz.out – standard output file with summary reports
2. amqsactz_1LS.out - API one line trace summary file
3. amqsactcz_v.out - verbose file

amqsactz.out

File #1 - Browse messages to create the standard activity trace output file that includes one line API calls and also various application summary reports at the bottom of the file:

```
amqsactz -r -b > amqsactz.out
```

In the amqsactz.out file, there is one record that is printed out per message from the SYSTEM.ADMIN.TRACE.ACTIVITY.QUEUE. Each record will contain all the API calls for a given application's connection and for a given time interval.

amqsactz.out – Record Example

```
MonitoringType: MQI Activity Trace RecordNum: 0
Correl_id:
00000000:  414D 5143 5858 5858 5858 5858 5858 582E  'AMQCXXXXXXXXXX.'
00000010:  9863 6055 C0EC 0520                      '.c`U...        '
QueueManager: 'XXXXXXXXXX.QM1'
Host Name: `xxxxxxxxxx'
IntervalStartDate: '2015-06-02'
IntervalStartTime: '11:50:29'
IntervalEndDate: '2015-06-02'
IntervalEndTime: '11:50:29'
CommandLevel: 800
SeqNumber: 0
ApplicationName: 'amqsput'
Application Type: MQAT_UNIX
ApplicationPid: 6780
UserId: 'mqm'
API Caller Type: MQXACT_EXTERNAL
API Environment: MQXE_OTHER
Application Function: ''
Appl Function Type: MQFUN_TYPE_UNKNOWN
Trace Detail Level: 2
Trace Data Length: 0
Pointer size: 8
Platform: MQPL_UNIX
========================================================================================================
1LS= Rec(0) Pid(6780) Tid(1) Date(2015-06-02) Time(11:50:29) Opr(MQXF_CONNX) RC(0) Chl() CnId(98636055C0EC0520)
1LS= Rec(0) Pid(6780) Tid(1) Date(2015-06-02) Time(11:50:29) Opr(MQXF_OPEN) RC(0) Chl() CnId(98636055C0EC0520)
HObj(2) Obj(TCZ.TEST1)
========================================================================================================
```

Application Summary Reports

- Following the SYSTEM.ADMIN.TRACE.ACTIVITY.QUEUE data in the amqsactz.out file are the application summary reports (-r option).

1. Application Summary Report
2. Application Objects Referenced Report
3. Application Objects Detail Report
4. Application Channels Referenced Report
5. Application Operations Executed Report
6. Application Operations Options Report
7. Application Operations Reason Code Report

Summary Report

- The Application Summary Report will show how many applications were found in the Activity Trace data. An application is determined by each unique pid, ApplicationName, and UserId that is found. This report will also show the how many different threads were detected, and the overall MQI calls that were made by the application.

```
================================================================================================
Application Summary Report
================================================================================================
Queue Manager          Pid       ApplicationName   UserId     Tid Count    MQI Count
XXXXXXXXXXX.QM1        6780      amqsput           mqm        1            10
XXXXXXXXXXX.QM1        6808      amqsput           mqm        1            15
XXXXXXXXXXX.QM1        6813      amqsput           mqm        1            13
XXXXXXXXXXX.QM1        28977     runmqtrm          mqm        1            7
```

Objects Referenced Report

- With the Application Objects Referenced report, you can see what objects were referenced by each application, and how many times.

```
================================================================================================
Application Objects Referenced Report
================================================================================================
Qmgr(XXXXXXXXXXX.QM1)   Pid(6780)   ApplName(amqsput)   UserId(mqm)   referenced objects:
  ObjName: TCZ.TEST1                                                 Count: 8
Qmgr(XXXXXXXXXXX.QM1)   Pid(6808)   ApplName(amqsput)   UserId(mqm)   referenced objects:
  ObjName: TCZ.TEST1                                                 Count: 13
Qmgr(XXXXXXXXXXX.QM1)   Pid(6813)   ApplName(amqsput)   UserId(mqm)   referenced objects:
  ObjName: TCZ.TEST1                                                 Count: 11
Qmgr(XXXXXXXXXXX.QM1)   Pid(28977)   ApplName(runmqtrm)   UserId(mqm)   referenced objects:
  ObjName: SYSTEM.DEFAULT.INITIATION.QUEUE                           Count: 7
```

Objects Detail Report

```
================================================================================================
Application Objects Detail Report
Details included by object are:
1. Operations found, including persistence (NonPrst, Prst, DfltPrst) and total message length data,
where applicable
2. Options found, which include conn, open, get, put, close, callback, sub, subrq
================================================================================================
Qmgr(XXXXXXXXXXX.QM1)  Pid(6780)  ApplName(amqsput)  UserId(mqm)  referenced the following
operations and options by object:
  Object Name: TCZ.TEST1
    Operation: MQXF_CLOSE      Count: 1
      Total Duration in Microseconds: 118                  Average Duration in Microseconds: 118
    Operation: MQXF_OPEN       Count: 1
      Total Duration in Microseconds: 227                  Average Duration in Microseconds: 227
    Operation: MQXF_PUT        Count: 6
      Total Duration in Microseconds: 10813                Average Duration in Microseconds: 1802
      DfltPrstCount: 6                    TotalMessageLength: 30
    Open Options: 8208       Count: 1
      MQOO_OUTPUT
      MQOO_FAIL_IF_QUIESCING
    Put Options: 8260       Count: 6                                  New at 8.0.0.2!
      MQPMO_NO_SYNCPOINT
      MQPMO_NEW_MSG_ID
      MQPMO_FAIL_IF_QUIESCING
    Close Options: 0          Count: 1
      MQCO_NONE
      MQCO_IMMEDIATE
```

Operations Reason Code Report

- The Application Operations Reason Code report will show the different reason codes and counts for each operation that the application executed.

```
================================================================================
Application Operations Reason Code Report
================================================================================
Qmgr(XXXXXXXXXXX.QM1)  Pid(6780)  ApplName(amqsput)  UserId(mqm)  referenced the following reason
codes by operations:
  Operation: MQXF_CLOSE
    Reason Code: 0          Count: 1
  Operation: MQXF_CONNX
    Reason Code: 0          Count: 1
  Operation: MQXF_DISC
    Reason Code: 0          Count: 1
  Operation: MQXF_OPEN
    Reason Code: 0          Count: 1
  Operation: MQXF_PUT
    Reason Code: 0          Count: 6
```

amqsactz_1LS.out - API trace

Reminder:
File #1 – We browsed messages to create the standard activity trace output file that included one line API calls and application summary reports at the bottom of the file:

```
amqsactz -r -b > amqsactz.out
```

File #2 - Now we will create a one line API trace file from this amqsactz.out file:

```
grep 1LS= amqsactz.out > amqsactz_1LS.out
```

amqsactz_1LS.out – API trace file

- NOTE: You can customize the fields that appear here with the -f and -g switches to amzsactz. There are currently 40 fields (i.e. MsgId, Expiry, etc.) to choose from.

1LS= Rec(0) Pid(6780) Tid(1) Date(2015-06-02) Time(11:50:29) Opr(MQXF_CONNX) RC(0) Chl() CnId(98636055C0EC0520)

1LS= Rec(0) Pid(6780) Tid(1) Date(2015-06-02) Time(11:50:29) Opr(MQXF_OPEN) RC(0) Chl() CnId(98636055C0EC0520) HObj(2) Obj(TCZ.TEST1)

1LS= Rec(1) Pid(6780) Tid(1) Date(2015-06-02) Time(11:50:30.596369) Opr(MQXF_PUT) RC(0) Chl() CnId(98636055C0EC0520) HObj(2) Obj(TCZ.TEST1)

1LS= Rec(2) Pid(6780) Tid(1) Date(2015-06-02) Time(11:50:31.728515) Opr(MQXF_PUT) RC(0) Chl() CnId(98636055C0EC0520) HObj(2) Obj(TCZ.TEST1)

1LS= Rec(3) Pid(6780) Tid(1) Date(2015-06-02) Time(11:50:33.400317) Opr(MQXF_PUT) RC(0) Chl() CnId(98636055C0EC0520) HObj(2) Obj(TCZ.TEST1)

1LS= Rec(4) Pid(6780) Tid(1) Date(2015-06-02) Time(11:50:34.424399) Opr(MQXF_PUT) RC(0) Chl() CnId(98636055C0EC0520) HObj(2) Obj(TCZ.TEST1)

1LS= Rec(5) Pid(6780) Tid(1) Date(2015-06-02) Time(11:50:36.760347) Opr(MQXF_PUT) RC(0) Chl() CnId(98636055C0EC0520) HObj(2) Obj(TCZ.TEST1)

1LS= Rec(6) Pid(6780) Tid(1) Date(2015-06-02) Time(11:50:37.656314) Opr(MQXF_PUT) RC(0) Chl() CnId(98636055C0EC0520) HObj(2) Obj(TCZ.TEST1)

1LS= Rec(6) Pid(6780) Tid(1) Date(2015-06-02) Time(11:50:38) Opr(MQXF_CLOSE) RC(0) Chl() CnId(98636055C0EC0520) HObj(2) Obj(TCZ.TEST1)

amqsactz_1LS.out – API trace with -u

- NOTE: The -u switch will make the connection id more readable, by changing it from 98636055C0EC0520 to a unique smaller number like 1.

1LS= Rec(0) Pid(6780) Tid(1) Date(2015-06-02) Time(11:50:29) Opr(MQXF_CONNX) RC(0) Chl() CnId(1)

1LS= Rec(0) Pid(6780) Tid(1) Date(2015-06-02) Time(11:50:29) Opr(MQXF_OPEN) RC(0) Chl() CnId(1) HObj(2) Obj(TCZ.TEST1)

1LS= Rec(1) Pid(6780) Tid(1) Date(2015-06-02) Time(11:50:30.596369) Opr(MQXF_PUT) RC(0) Chl() CnId(1) HObj(2) Obj(TCZ.TEST1)

1LS= Rec(2) Pid(6780) Tid(1) Date(2015-06-02) Time(11:50:31.728515) Opr(MQXF_PUT) RC(0) Chl() CnId(1) HObj(2) Obj(TCZ.TEST1)

1LS= Rec(3) Pid(6780) Tid(1) Date(2015-06-02) Time(11:50:33.400317) Opr(MQXF_PUT) RC(0) Chl() CnId(1) HObj(2) Obj(TCZ.TEST1)

1LS= Rec(4) Pid(6780) Tid(1) Date(2015-06-02) Time(11:50:34.424399) Opr(MQXF_PUT) RC(0) Chl() CnId(1) HObj(2) Obj(TCZ.TEST1)

1LS= Rec(5) Pid(6780) Tid(1) Date(2015-06-02) Time(11:50:36.760347) Opr(MQXF_PUT) RC(0) Chl() CnId(1) HObj(2) Obj(TCZ.TEST1)

1LS= Rec(6) Pid(6780) Tid(1) Date(2015-06-02) Time(11:50:37.656314) Opr(MQXF_PUT) RC(0) Chl() CnId(1) HObj(2) Obj(TCZ.TEST1)

1LS= Rec(6) Pid(6780) Tid(1) Date(2015-06-02) Time(11:50:38) Opr(MQXF_CLOSE) RC(0) Chl() CnId(1) HObj(2) Obj(TCZ.TEST1)

amqsactz_v.out - verbose file

File #3 - Browse messages to create a formatted activity trace output file with verbose expansion of each API call:

```
amqsactz -v -b > amqsactz_v.out
```

amqsactz_v.out – Record Example

- First part of record 0 is similar to the amqsactz.out file (non-verbose).

```
MonitoringType: MQI Activity Trace RecordNum: 0
Correl_id:
00000000:  414D 5143 5858 5858 5858 5858 5858 582E  'AMQCXXXXXXXXXXX.'
00000010:  9863 6055 C0EC 0520                       '.c`U...          '
QueueManager: 'XXXXXXXXXXX.QM1'
Host Name: 'xxxxxxxxxxx'
IntervalStartDate: '2015-06-02'
IntervalStartTime: '11:50:29'
IntervalEndDate: '2015-06-02'
IntervalEndTime: '11:50:29'
CommandLevel: 800
SeqNumber: 0
ApplicationName: 'amqsput'
Application Type: MQAT_UNIX
ApplicationPid: 6780
UserId: 'mqm'
API Caller Type: MQXACT_EXTERNAL
API Environment: MQXE_OTHER
Application Function: ''
Appl Function Type: MQFUN_TYPE_UNKNOWN
Trace Detail Level: 2
Trace Data Length: 0
Pointer size: 8
Platform: MQPL_UNIX
```

amqsactz_v.out – Record Example

- However, instead of getting a one line data summary of the API call, now each call is expanded into all the fields that were included in the Activity Trace data for that API call. Here is the verbose data for the MQCONNX API call.

```
MQI Operation: 0
  Operation Id: MQXF_CONNX
  ApplicationTid: 1
  OperationDate: '2015-06-02'
  OperationTime: '11:50:29'
  ConnectionId:
  00000000:  414D 5143 5858 5858 5858 5858 5858 582E  'AMQCXXXXXXXXXX.'
  00000010:  9863 6055 C0EC 0520                      '.c`U...         '
  QueueManager: 'XXXXXXXXXXX.QM1'
  QMgr Operation Duration: 417
  Completion Code: MQCC_OK
  Reason Code: 0
  Connect Options: 256
```

amqsactz_v.out – Record Example

- Here is the verbose data for the MQOPEN API call.

```
MQI Operation: 1
  Operation Id: MQXF_OPEN
  ApplicationTid: 1
  OperationDate: '2015-06-02'
  OperationTime: '11:50:29'
  Object_type: MQOT_Q
  Object_name: 'TCZ.TEST1'
  Object_Q_mgr_name: ''
  Hobj: 2
  QMgr Operation Duration: 227
  Completion Code: MQCC_OK
  Reason Code: 0
  Open_options: 8208     <- Use MQOptions in MH06 supportpac to find constant values for 8208
  Object_type: MQOT_Q
  Object_name: 'TCZ.TEST1'
  Object_Q_mgr_name: ''
  Resolved_Q_Name: 'TCZ.TEST1’
  Resolved_Q_mgr: 'XXXXXXXXXXX.QM1'
  Resolved_local_Q_name: 'TCZ.TEST1‘     <- This would be the XMITQ name for a remote queue
  Resolved_local_Q_mgr: 'XXXXXXXXXXX.QM1'
  Resolved_type: MQOT_Q
  Dynamic_Q_name: 'AMQ.*'
```

Activity Trace – API Data Structures

- You can get a hex dump of some of the API data structures (i.e. GMO) with an Activity Trace. The TraceLevel needs to be HIGH to get these data structures in the Activity Trace. The mqtrcfrmt program in the MH06 supportpac can also format most of these API data structures into a more human readable format. We will look at a formatted MQGMO data structure, over the next few slides.

MQGMO expansion with mqtrcfrmt

```
MQGMO Structure:
 00000000:  474D 4F20 0000 0004 0200 0005 0000 7530  'GMO ..........u0'
 00000010:  0000 0000 0000 0000 5359 5354 454D 2E4D  '........SYSTEM.M'
 00000020:  414E 4147 4544 2E4E 4455 5241 424C 452E  'ANAGED.NDURABLE.'
 00000030:  3535 4246 4233 3635 3230 3030 3443 3033  '55BFB36520004C03'
 00000040:  2020 2020 2020 2020 0000 0003 2020 2000  '        ....   .'
 00000050:  55BF B364 0000 0041 0000 0000 0000 0001  'U..d...A........'
 00000060:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
     Getmsgopts expanded (all fields):
     StrucId (CHAR4)              : 'GMO '
                                    x'474D4F20'
     Version (MQLONG)             : 4
                                    x'00000004'
     MQGMO.Options= (MQLONG)      : 33554437
                                    x'02000005'
        Options=MQGMO_WAIT
        Options=MQGMO_NO_SYNCPOINT
        Options=MQGMO_PROPERTIES_FORCE_MQRFH2
     WaitInterval (MQLONG)        : 30000
                                    x'00007530'
     Signal1 (MQLONG)             : 0
                                    x'00000000'
     Signal2 (MQLONG)             : 0
                                    x'00000000'
     ResolvedQName (MQCHAR48)     : 'SYSTEM.MANAGED.NDURABLE.55BFB36520004C03        '
```

MQGMO expansion with mqtrcfrmt

```
MQMO.MatchOptions= (MQLONG) : 3
                                x'00000003'
      MatchOptions=MQMO_MATCH_MSG_ID
      MatchOptions=MQMO_MATCH_CORREL_ID
   GroupStatus (MQCHAR)        : ' '
                                 x'20'
      GroupStatus MQGS_NOT_IN_GROUP
   SegmentStatus (MQCHAR)      : ' '
                                 x'20'
      SegmentStatus MQSS_NOT_A_SEGMENT
   Segmentation (MQCHAR)       : ' '
                                 x'20'
      Segmentation MQSEG_INHIBITED
   Reserved1 (MQCHAR)          : '.'
                                 x'00'
   MsgToken (MQBYTE16)         : x'55BFB364000000410000000000000001'
   ReturnedLength (MQLONG)     : 0
                                 x'00000000'
   Reserved2 (MQLONG)          : 0
                                 x'00000000'
   MsgHandle (MQHMSG)          : x'0000000000000000'
```

Note on Tracing the Message

- Data conversion happens outside of the queue manager. For a bindings (local) connection, the data conversion happens in the application process. For a client connection, it could happen in the amqrmppa (channel pooling) process on Linux, as an example.

- As a consequence, the Application Activity Trace does not show the converted message on an MQGET, since the data conversion is happening outside of the queue manager and the Activity Trace runs inside the queue manager.

- strmqtrc “-d all” does show the converted message on an MQGET in its trace, since it also traces in the application process or the SVRCONN channel process (e.g. amqrmppa).

Activity Trace – WARNING!

- On three separate occasions over the past several years of using the Activity Trace, I have run into issues (in Production) where turning on the Activity Trace has caused the queue manager to hit an internal error and become unstable. The queue manager had to be restarted to restore service.

- This is a rare occurrence, but it is something to be aware of.

- This type of issue just happened to me recently on Linux (June 2017), and there is an APAR IT09496 to correct the issue (tentative release schedule for APAR IT09496 below):

Version	Maintenance Level
v7.5	7.5.0.9
v8.0	8.0.0.8
v9.0 CD	9.0.4
v9.0 LTS	9.0.0.3

Distributed Native Tools

- When you have a difficult MQ application problem to solve, sometimes using some of the native problem determination tools for your operating system environment can be helpful. They allow you to “look under the hood” of what the application is doing, and can sometimes provide some helpful insight.

- UNIX/Linux

1. strace or truss - records system calls of running processes
2. lsof or pfiles - display what files that process has open

- Windows

1. Process Explorer (Windows Sysinternals) - find what files, DLLs, a process has open
2. Process Monitor (Windows Sysinternals) - records real time file system, registry and process/thread activity.

Questions & Answers