MQ Problem Determination with Tracing

Tim Zielke

Introduction and Agenda

My Background:

- I have been in IT for 18 years with Hewitt Associates/Aon
- First 13 years mainly on the mainframe COBOL application side
- Last 5 years as a CICS/MQ systems programmer

Session Agenda:

We will cover the following topics that can help with MQ problem determination:

- MQ API tracing with strmqtrc
- Application Activity Trace
- MH06 Trace Tools Supportpac
- Briefly touch on distributed native tools/commands that can be helpful for MQ problem determination

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                            |....            |
```

Signed Binary Integer Data in Trace

- Before we get into reading the API data structures (i.e. Objdesc), let's cover some technical concepts that will help us in that endeavor.

- There are many MQ data fields (Expiry, GMO, etc.) that will appear in the trace that are signed four byte binary integers (MQLONG). These fields can sometimes confuse people when reading a trace, as there are cases where it is easy to misread their data. In order to properly understand how to read these fields, we first need to understand some technical concepts like two's complement and endianness. We will cover these concepts in the next few slides.

2's Complement

- Two's complement is how negative numbers are commonly represented with signed binary integers. For example, a signed four byte binary integer that holds x'FFFFFFFF' = -1.

- Any signed four byte integer with the highest bit set to 1 is a negative number. Therefore, any number that is inclusively between x'80000000' - x'FFFFFFFF' is a negative number, since the highest bit in 8 - F is set to 1.

Ex. x'8' = 1000, x'9' = 1001, x'A' = 1010, . . . , x'D' = 1101, x'E' = 1110, x'F' = 1111

- To find the negative number's absolute value, you flip the bits and add 1. So flipping the bits of x'FFFFFFFF' = x'00000000' and adding 1 gives you 1. So x'FFFFFFFF' = -1.

- Common negative numbers you will see in the traces are:

x'FFFFFFFF' = -1
x'FFFFFFFE' = -2
x'FFFFFFFD' = -3

Endianness – Little Endian (i.e. x86)

Endianness is how a CPU orders the bytes for multi-byte binary data. There is little endian and big endian. Little endian means the littlest byte starts at the lowest address. Big endian means the biggest byte starts at the lowest address. The x86 is a little endian CPU. Most other CPUs are big endian.

Example: Hex value x'01400006', stored at starting address x'0000A010' on x86.

A Little endian CPU (x86) will store this 4 byte value at the starting memory address (i.e. address x'0000A010') from the least significant byte to most significant byte, or little end first.

```
address 0000A010 = x'06'
address 0000A011 = x'00'
address 0000A012 = x'40'
address 0000A013 = x'01'
```

When looking at an MQ trace, this would appear as 06004001. This looks intuitively "reversed" when reading the trace.

Endianness – Big Endian (i.e. SPARC)

Example: Hex value x'01400006', stored at starting address x'0000A010' on a Big endian processor (SPARC).

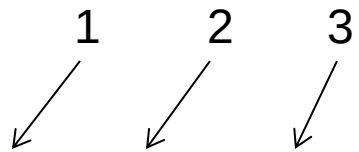
A Big endian CPU (SPARC) will store this 4 byte value at the starting memory address (i.e. address x'0000A010') from the most significant byte to least significant byte, or big end first.

```
address 0000A010 = x'01'
address 0000A011 = x'40'
address 0000A012 = x'00'
address 0000A013 = x'06'
```

When looking at an MQ trace, this would appear as 01400006. This looks intuitively "normal" when reading the trace.

MQ Tracing – Reading a Data Structure

- MQ data structures such as Objdesc, Msgdesc, Putmsgopts, etc. appear in the API trace. The MQ data structures follow a format of a 4 byte character structure id, a 4 byte binary integer version id, and then subsequent fields. The layouts of the data structures can be found in the MQ manual. Here is an example of an Objdesc from our MQ trace which was run on an x86 processor.

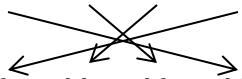



```
51              13:15:37.742885    16884.1     Objdesc:
52              13:15:37.742887    16884.1       0x0000:  4f442020 01000000 01000000 54435a2e
53              13:15:37.742887    16884.1       0x0010:  54455354 31000000 00000000 00000000
```

```
Field 1 is Structure Id (MQCHAR4) = x'4f442020' = "OD  "
Field 2 is Version (MQLONG)       = x'01000000' = 1
Field 3 is Object Type (MQLONG)   = x'01000000' = 1 (MQOT_Q or Queue Object Type)

Remember to reverse bytes for binary fields, since this trace was run on a little endian processor (x86):

Version:
01   00   00   00
```



```
00   00   00   01 = Version is 1
```

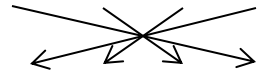
MQ Tracing – Reading an Options Field

- Reading Open Options on line 64

```
 63              13:15:37.742888    16884.1      Options:
 64              13:15:37.742889    16884.1      0x0000:  10200000

Reverse bytes for binary integer fields, since this is a trace generated on a little endian CPU (x86):

Options (MQLONG):
10   20   00   00
```



```
00   00   20   10 = Options is x'00002010' = 8208


To convert 8208 to its open options constant values, find the largest open option value that is closest to or
equal to 8208 and subtract that value.  Continue this process, until you reach 0.

8192 = MQOO_FAIL_IF_QUIESCING

8208 - 8192 = 16

16 = MQOO_OUTPUT

16 - 16 = 0

Therefore, 8208 = MQOO_FAIL_IF_QUIESCING, MQOO_OUTPUT
```

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)
```

MQ Tracing – 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
```

MQ Tracing - 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'
```

MQ Tracing - 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
```

MQ Tracing – API Summary Trace

- A User customizable API summary trace can also be generated from our 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
```

MQ Tracing Other Uses - Performance

1) General performance of API calls

- 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.

```
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.

MQ Tracing Other Uses – MQ programs

2) Investigation of MQ supplied programs (i.e. runmqtrm, runmqdlq, etc.)

```
strmqtrc –m qmgr –t all –p runmqtrm
```

- The MQ supplied programs come with their own built in tracing that can be activated with strmqtrc. The traces can be cryptic, but there is usually enough readable data to provide some helpful diagnostics for the MQ administrator. For example, a trace of the runmqtrm program will record if/when the trigger message was read from the INITQ, if/when it started the application of your process, and the operating system return code from the start call. Also, this does not require that runmqtrm be run in the foreground, to debug it.

MQ Tracing Other Uses - Clients

- Client applications can be traced, as well. You can either run a client trace on the client server (unfortunately, Java clients do not support this type of tracing) or trace the queue manager process that the SVRCONN channel is running on.

- Examples of client traces

1. `strmqtrc -t api -p prog1 (from client server)`

2. `strmqtrc -m qmgr -t api -p amqrmppa (from queue manager server)`

Some Final strmqtrc Comments

- Tracing adds performance overhead and can create large files. Be judicious on the length of time that you run the trace and try and be selective with the options (i.e. -t api -p prog1) to reduce any unneeded output. Also, keep an eye on the size of your trace files and your space available on your trace file system (i.e. /var/mqm). You can also use the strmqtrc -l (MaxSize in MB) option to limit the size of your trace files, but this means that trace data can be overwritten and lost. The -l option keeps a current AMQppppp.qq.TRC and a previous AMQppppp.qq.TRS file.

```
strmqtrc –m qmgr –t api –p amqsput –l 1
```

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.
- “Increasing the visibility of messages using WebSphere MQ Application Activity Trace” by Emma Bushby is an IBM DeveloperWorks article that does a good job in explaining the Application Activity Trace in detail.
- 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.

mqat.ini – Turning on Trace for an Appl

- This is an example of adding an ApplicationTrace stanza to the mqat.ini file to turn on the Activity Trace for any application whose name is amqsput.

```
ApplicationTrace:            # Application specific settings stanza
  ApplClass=ALL              # Application type
                             #   Values:  (USER | MCA | ALL)
                             #   Default: USER
  ApplName=amqsput           # Application name (may be wildcarded)
                             #   (matched to app name without path)
                             #   Default: *
  Trace=ON                   # Activity trace switch for application
                             #   Values:  ( ON | OFF )
                             #   Default: OFF
  ActivityInterval=0         # Time interval between trace messages
                             #   Values: 0-99999999 (0=off)
                             #   Default: 0
  ActivityCount=0            # Number of operations between trace msgs
                             #   Values: 0-99999999 (0=off)
                             #   Default: 0
  TraceLevel=MEDIUM          # Amount of data traced for each operation
                             #   Values: LOW | MEDIUM | HIGH
                             #   Default: MEDIUM
  TraceMessageData=0         # Amount of message data traced
                             #   Values: 0-104857600
                             #   Default: 0
```

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 on Capitalware's Sample WebSphere MQ C Code web site 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 (i.e. 554F108A2061F801) can be hard to read and differentiate in a trace. This feature will abstract each unique connection id to a more readable number (i.e. 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 a one line summary of 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
```

Channels Referenced Report

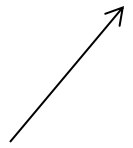
- The Application Channels Referenced report will show what channels were referenced by your application. In our case there were none, but any channels referenced and how many times they were referenced would be displayed in this report.

```
================================================================================================
Application Channels Referenced Report
================================================================================================
Qmgr(XXXXXXXXXXX.QM1)   Pid(6780)  ApplName(amqsput)  UserId(mqm)  referenced the following channels:
Qmgr(XXXXXXXXXXX.QM1)   Pid(6808)  ApplName(amqsput)  UserId(mqm)  referenced the following channels:
Qmgr(XXXXXXXXXXX.QM1)   Pid(6813)  ApplName(amqsput)  UserId(mqm)  referenced the following channels:
Qmgr(XXXXXXXXXXX.QM1)   Pid(28977) ApplName(runmqtrm) UserId(mqm)  referenced the following channels:
```

Operations Executed Report

- The Application Operations Executed report will show what and how many API calls were made, what persistence was used, total message length data, and performance numbers for the API calls (performance numbers currently only available at 8.0.0.2).

```
================================================================================================
Application Operations Executed Report
This report also includes persistence (NonPrst, Prst, DfltPrst) and total message length data, where
applicable
================================================================================================
Qmgr(XXXXXXXXXXX.QM1)  Pid(6780)  ApplName(amqsput)  UserId(mqm)  executed the following operations:
  Operation: MQXF_CLOSE      Count: 1
      Total Duration in Microseconds: 118                  Average Duration in Microseconds: 118
  Operation: MQXF_CONNX      Count: 1
      Total Duration in Microseconds: 417                  Average Duration in Microseconds: 417
  Operation: MQXF_DISC       Count: 1
  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
```



Operations Options Report

- The Application Operations Options report will show what options were used for each operation.

```
================================================================================
Application Operations Options Report
Options tracked include conn, open, get, put, close, callback, sub, subrq
================================================================================
Qmgr(XXXXXXXXXXX.QM1)  Pid(6780)  ApplName(amqsput)  UserId(mqm)  referenced the following options
by operations:
  Operation: MQXF_CLOSE
    Close Options: 0          Count: 1
      MQCO_NONE
      MQCO_IMMEDIATE
  Operation: MQXF_CONNX
    Connect Options: 256        Count: 1
      MQCNO_SHARED_BINDING
  Operation: MQXF_OPEN
    Open Options: 8208       Count: 1
      MQOO_OUTPUT
      MQOO_FAIL_IF_QUIESCING
  Operation: MQXF_PUT
    Put Options: 8260       Count: 1
      MQPMO_NO_SYNCPOINT
      MQPMO_NEW_MSG_ID
      MQPMO_FAIL_IF_QUIESCING
```

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  '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
```

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     <- New at 8.0.0.2!
  Completion Code: MQCC_OK
  Reason Code: 0
  Connect Options: 256    <- Use MQOptions in MH06 supportpac to find constant values for 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     <- New at 8.0.0.2!
  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.*'
```

amqsactz parameters

```
/*   amqsact has the following parameters                                */
/*                                                                       */
/*   amqsact    [-m QMgrName]                                           */
/*              [-q QName]          # Override default queue name       */
/*              [-t TopicString]    # Subscribe to event topic          */
/*              [-b]                # Only browse records               */
/*              [-c CCSID]          # CCSID to use on GET               */
/*              [-v]                # Verbose output                    */
/*              [-d <depth>]        # Number of records to display      */
/*              [-w <timeout>]      # Time to wait (in seconds)         */
/*              [-s <startTime>]    # Start time of record to process   */
/*                                  # i.e. 2015-07-01T11:10:00          */
/*              [-e <endTime>]      # End time of record to process     */
/*                                  # i.e. 2015-07-01T11:11:00          */
/*              [-f <APIFields1>]   # Fields to display for API line    */
/*              [-g <APIFields2>]   # More APID fields for API line     */
/*              [-u]                # Abstract Connection Id for        */
/*                                    readability in API 1 Line         */
/*              [-r]                # Create summary reports for non-   */
/*                                    verbose run                       */
```

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.

- For client connections, you can get the MQCD (channel definition) data structure for the MQCONN/MQCONNX. The mqtrcfrmt program does not expand this data structure. We will also look at an MQCD over the next several slides and see how to read it from an offset and field stand point.

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'
```

MQCD layout

```
MQCD Structure:
 00000000:  434C 4945 4E54 2E54 4F2E 5345 5256 4552  'CLIENT.TO.SERVER'
     ChannelName (+0 for 20) = 'CLIENT.TO.SERVER'
 00000010:  2020 2020 0000 0000 0700 0000 0200 0000  '    ............'
     Version (+14 for 4) = 0
     ChannelType (+18 for 4) = 7 (MQCHT_SVRCONN)
     TransportType (+1C for 4) = 2 (MQXPT_TCP)
 00000020:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
     Desc (+20 for 64)
 00000030:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
 00000040:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
 00000050:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
 00000060:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
     QMgrName (+60 for 48)
 00000070:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
 00000080:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
 00000090:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
     XmitQName (+90 for 48)
 000000A0:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
 000000B0:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
 000000C0:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
     ShortConnectionName (+C0 for 20)
 000000D0:  0000 0000 2020 2020 2020 2020 2020 2020  '....            '
     MCAName (+D4 for 20)
 000000E0:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
     ModeName (+E8 for 8)
 000000F0:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
     TpName (+F0 for 64)
 00000100:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
```

MQCD layout

```
00000110:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
00000120:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
00000130:  3200 0000 0000 0000 0A00 0000 3C00 0000  '2...........<...'
    BatchSize (+130 for 4) = 50 = x'32'
    DiscInterval (+134 for 4) = 0
    ShortRetryCount (+138 for 4) = 10 = x'A'
    ShortRetryInterval (+13C for 4) = 60 = x'3C'
00000140:  FFC9 9A3B B004 0000 2020 2020 2020 2020  '...;....        '
    LongRetryCount (+140 for 4) = 999,999,999 = x'3B9AC9FF'
    LongRetryInterval (+144 for 4) = 1,200 = x'4B0'
    SecurityExit (+148 for 128)
00000150:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
00000160:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
00000170:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
00000180:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
00000190:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
000001A0:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
000001B0:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
000001C0:  2020 2020 2020 2020 0000 0000 0000 0000  '        ........'
    MsgExit (+1C8 for 128)
000001D0:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
000001E0:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
000001F0:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
00000200:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
00000210:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
00000220:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
00000230:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
```

MQCD layout

```
00000240:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
    SendExit (+248 for 128)
00000250:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
00000260:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
00000270:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
00000280:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
00000290:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
000002A0:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
000002B0:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
000002C0:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
    ReceiveExit (+2C8 for 128)
000002D0:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
000002E0:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
000002F0:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
00000300:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
00000310:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
00000320:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
00000330:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
00000340:  0000 0000 0000 0000 FFC9 9A3B A086 0100  '...........;....'
    SeqNumberWrap (+348 for 4) = 999,999,999 = x'3B9AC9FF'
    MaxMsgLgth (+34C for 4) = 100,000 = x'186A0'
00000350:  0100 0000 0000 0000 2020 2020 2020 2020  '........        '
    PutAuthority (+350 for 4) = 1
    DataConversion (+354 for 4) = 0
    SecurityUserData (+358 for 32)
00000360:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
00000370:  2020 2020 2020 2020 0000 0000 0000 0000  '        ........'
    MsgUserData (+378 for 32)
00000380:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
```

MQCD layout

```
00000390:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
    SendUserData (+398 for 32)
000003A0:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
000003B0:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
    ReceiveUserData (+3B8 for 32)
000003C0:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
000003D0:  0000 0000 0000 0000 2020 2020 2020 2020  '........        '
    UserIdentifier (+3D8 for 12)
000003E0:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
    Password (+3E4 for 12)
000003F0:  0000 0000 0000 0000 0000 0000 0100 0000  '................'
    MCAUserIdentifier (+3F0 for 12)
    MCAType (+3FC for 4) = 1
00000400:  3132 372E 302E 302E 3100 0000 0000 0000  '127.0.0.1.......'
    ConnectionName (+400 for 264)
00000410:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
00000420:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
00000430:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
00000440:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
00000450:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
00000460:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
00000470:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
00000480:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
00000490:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
000004A0:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
000004B0:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
000004C0:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
000004D0:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
000004E0:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
```

MQCD layout

```
000004F0:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
00000500:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
    RemoteUserIdentifier (+508 for 12)
00000510:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
    RemotePassword (+514 for 12)
00000520:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
    MsgRetryExit (+520 for 128)
00000530:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
00000540:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
00000550:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
00000560:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
00000570:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
00000580:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
00000590:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
    MsgRetryUserData (+590 for 32)
000005A0:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
000005B0:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
000005C0:  0A00 0000 E803 0000 2C01 0000 0000 0000  '........,.......'
    MsgRetryCount (+5C0 for 4) = 10 = x'A'
    MsgRetryInterval (+5C4 for 4) = 1,000 = x'3E8'
    HeartBeatInterval (+5C8 for 4) = 300 = x'12C'
    BatchInterval (+5CC for 4) = 0
000005D0:  0200 0000 0000 0000 0000 0000 0000 0000  '................'
    NonPersistentMsgSpeed (+5D0 for 4) = 2
    SrucLentgh (+5D4 for 4)
    ExitNameLength (+5D8 for 4)
    ExitDataLength (+5DC for 4)
```

MQCD layout

```
000005E0:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
    MsgExitsDefined (+5E0 for 4)
    SendExitsDefined (+5E4 for 4)
    ReceiveExistDefined (+5E8 for 4)
    !+5EC has structure padding of 4 bytes to align next 8 byte pointer!
000005F0:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
    MsgExitPtr (+5F0 for 8)
    MsgUserDataPtr (+5F8 for 8)
00000600:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
    SendExitPtr (+600 for 8)
    SendUserDataPtr (+608 for 8)
00000610:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
    ReceiveExitPtr (+610 for 8)
    ReceiveUserDataPtr (+618 for 8)
00000620:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
    ClusterPtr (+620 for 8)
    ClustersDefined (+628 for 4)
    NetworkPriority (+62C for 4)
00000630:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
    LongMCAUserIdLength (+630 for 4)
    LongRemoteUserIdLength (+634 for 4)
    LongMCAUserIdPtr (+638 for 8)
    LongRemoteUserIdPtr (+63C for 8)
00000640:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
    LongRemoteUserIdPtr (+640 for 8)
    MCASecurityId (+648 for 40)
00000650:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
00000660:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
```

MQCD layout

```
00000670:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
    RemoteSecurityId (+670 for 40)
00000680:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
00000690:  0000 0000 0000 0000 2020 2020 2020 2020  '........        '
    SSLCipherSpec (+698 for 32)
000006A0:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
000006B0:  2020 2020 2020 2020 0000 0000 0000 0000  '        ........'
    SSLPeerNamePtr (+6B8 for 8)
000006C0:  3400 0000 0000 0000 FFFF FFFF 2020 2020  '4...........    '
    SSLPeerNameLength (+6C0 for 4) = 52 = x'34'
    SSLClientAuth (+6C4 for 4) = 0
    KeepAliveInterval (+6C8 for 4) = -1 = x'FFFFFFFF'
    LocalAddress (+6CC for 48)
000006D0:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
000006E0:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
000006F0:  2020 2020 2020 2020 2020 2020 0000 0000  '            ....'
    BatchHeartbeat (+6FC for 4) = 0
00000700:  0000 0000 FFFF FFFF 0000 0000 FFFF FFFF  '................'
    HdrCompList (+700 for 4) = 0
    MsgCompList (+704 for 4) = -1 = x'FFFFFFFF'
    CLWLChannelRank (+708 for 4) = 0
    CLWLChannelPriority (+70C for 4) = -1 = x'FFFFFFFF'
00000710:  FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  '................'
    CLWLChannelWeight (+710 for 4) = -1 = 'FFFFFFFF'
    ChannelMonitoring (+714 for 4) = -1 = 'FFFFFFFF'
    ChannelStatistics (+718 for 4) = -1 = 'FFFFFFFF'
    SharingConversations (+71C for 4) = -1 = 'FFFFFFFF'
```

MQCD layout

```
00000720:  FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  '................'
    PropertyControl (+720 for 4) = -1 = 'FFFFFFFF'
    MaxInstances (+724 for 4) = -1 = 'FFFFFFFF'
    MaxInstacnesPerClient (+728 for 4) = -1 = 'FFFFFFFF'
    ClientChannelWeight (+72C for 4) = -1 = 'FFFFFFFF'
00000730:  FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  '................'
    ConnectionAffinity (+730 for 4) = -1 = 'FFFFFFFF'
    BatchDataLimit (+734 for 4) = -1 = 'FFFFFFFF'
    UseDLQ (+738 for 4) = -1 = 'FFFFFFFF'
    DefReconnect (+73C for 4) = -1 = 'FFFFFFFF'
00000740:  FFFF FFFF FFFF FFFF 0000 0000 0000 0000  '................'
    CertificateLabel (+740 for 64)
00000750:  3200 0000 FDFF FFFF FDFF FFFF 0A00 0000  '2...............'
00000760:  0000 0000 FFC9 9A3B FFC9 9A3B 0000 0000  '.......;...;....'
00000770:  0100 0000 8813 0000 0200 0000 0000 0000  '................'
```

Final Comments on Tracing

- Application Activity Trace is easier to work with than strmqtrc API tracing. Having the trace data go to a queue (Activity Trace) is easier to manage than having the trace data go to numerous files (strmqtrc). The summary reports and one API line tracing (amqsactz) helps with more easily being able to understand the trace data.

- strmqtrc API tracing provides the most detail. If you need to look into both the input and return data of a field (i.e. CodedCharSetId, MsgId, etc.), strmqtrc tracing is better at showing this data. When looking into GETs with MQGMO_CONVERT, the Activity Trace actually shows you fields like the message data and CodedCharSetId before the convert, which might not be what you expect. For example, if a message was in CCSID 37 (EBCDIC) and being converted to 819 (ASCII) with an MQGET (with convert), the data in the activity trace for this MQGET would show the CCSID as 37 and the message data in EBCDIC.

- It can be helpful to run the strmqtrc API tracing and Activity Trace simultaneously and then compare the trace results. Comparing the two traces can sometimes help in understanding things that appear to be odd in the trace.

mqcpcnvt

- mqcpcnvt on Capitalware's Sample WebSphere MQ C Code web site is a program that helps you work with byte conversion between code pages. You can use it to see how certain bytes will convert between two code pages, decipher a byte string into ASCII characters, etc.

```
> mqcpcnvt C9C5C6C2D9F1F4 37 1208 TCZ.TEST1
mqcpcnvt start
target queue is TCZ.TEST1
The following byte string was PUT with CCSID 37
<C9C5C6C2D9F1F4>
The following byte string was returned with converted GET with CCSID 1208:
<49454642523134>
This is the returned byte string printed as chars (a non isprint char will be shown as '.').
<IEFBR14>
mqcpcnvt end
```

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.

strace on Linux examples

- You have executed a “strmqm QM1” command, and it is taking a long time to complete and appears to be hung. If the pid of the strmqm command is 1234, you can do the following command to see if it is doing any kind of system call activity. If your process is stuck inside a system call, this should show up in the output. If your process is doing work (and more than likely it would also be making system calls), you will see that in the strace output, as well. This can give you some insight into what the process is doing, that appears to be hung.

```
strace -p 1234
```

strace on Linux examples

- You have an MQ application which is having issues, and you suspect it has to do with the files that are being accessed by the application. One option is to use strace to debug this. For example you could run the following to see what files the application is opening. You would look for the open system calls, to find the file names. The -f option will trace any child processes that are created. The strace writes its output to standard error (file descriptor 2).

```
strace –f program 2>strace.out
```

- You have an MQ application or command that is producing a lower level error message that you are having trouble diagnosing. One options is to use strace to debug this. You can run the command listed above, but this time search for a write system call that has the beginning of the error message that is being written out. You can them look right above in the strace to see if there are any clues on why the error message is being produced.

Questions & Answers