MH06 Trace Tools Supportpac
Author – Tim Zielke – CICS/MQ Systems Programmer Alight Solutions

Quick Start Examples:

1) Use MQOptions to find constant values for an Open 34:

```
> 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)  34

open options for decimal value 34 and hex value 0x22 converts to:
0x00000002  MQOO_INPUT_SHARED
0x00000020  MQOO_INQUIRE
```

2) Use mqtrcfrmt to format a Windows trace on the Windows platform:

```
> mqtrcfrmt.win.exe AMQ5488.0.TRC AMQ5488.0.TRC.FMT
```

3) Use mqtrcfrmt to format a unix trace on the Windows platform:

```
> mqtrcfrmt.win.exe AMQ5488.0.FMT AMQ5488.0.FMT2 -o unix -e big
```

4) Use mqtrcfrmt to format a z/OS trace on the Windows platform:

```
> mqtrcfrmt.win.exe ZOS.TRC ZOS.TRC2 -o zos -e big
```

5) Use mqtrcfrmt to format an Application Activity Trace from amqsactz on the Windows platform:

```
> mqtrcfrmt.win.exe amqsactz.out amqsactz.out2 -o aat -p 8
```

6) Use mqtrcfrmt to format a Solaris trace generated from a 32 bit app on the Solaris platform:

```
> mqtrcfrmt.solaris AMQ5488.0.FMT AMQ5488.0.FMT2 -p 4
```

7) Use mqapitrcstats to get stats report from Linux trace on Linux platform:

```
> mqapitrcstats.linux AMQ5488.0.FMT AMQ5488.0.FMT.rpt
```

8) Use mqapitrcstats to get stats report from Windows trace on Linux platform:

```
> mqapitrcstats.linux AMQ5488.0.TRC AMQ5488.0.TRC.rpt -o windows
```

9) Use mqapitrcstats to get stats report from Solaris trace on Windows platform:

```
> mqapitrcstats.win.exe AMQ5488.0.FMT AMQ5488.0.FMT.rpt -o unix -e big
```

MQOPTIONS Manual

Program name: MQOptions
Author: Tim Zielke - CICS/MQ Systems Programmer for Alight Solutions

Description:
MQ diagnostic tool to convert decimal or hex values of options to constant values. The constant resolution used in this program was built against the 8.0 constants.

Programs Provided:
MQOptions.class – compiled Java 1.5 class

Function:
This program will take a decimal or hex options value for either conn, open, get, put, close, cbd, sub, subrq, report and provide the constant values that the value converts to.

ENDIANNESS NOTE: When working with an options field, you do need to make sure you correctly determine the endianness of the field. Endianness is how a processor will order multi byte binary data. A little endian processor (x86) will put the least significant byte in the lowest address. A big endian processor (SPARC) will put the most significant byte in the lowest address. Here are some examples from MQ traces:

Open options field 8208 = 0x2010 in a SPARC (big endian) trace:

Options:
0x0000: 00002010

Open options field 8208 = 0x2010 in a x86 (little endian) trace:

Options:
0x0000: 10200000

Notice in the x86 trace, that the bytes 00 00 20 10 have been byte reversed to 10 20 00 00. This is because the byte 10 is the least significant byte, so it will appear in the lowest address on a little endian processor. If the bytes had been x'12345678', they would appear as 78 56 34 12 on a little endian processor.

The bottom line is that if you are working with an options field that was generated on a little endian (i.e. x86) processor, you will need to reverse the bytes to read it accurately. The MQOptions program will tell you the endianness of the platform that the program is running on in its first line of output. This can help you identify the endianness of your options field, if you are working with a trace that was generated on the same platform that you are using for MQOptions. Another tip is look for an MQ Version field in your data, and see if it looks like a small or large number. The Version field that appears after the StrucId (i.e. GMO) in an MQ data structure will be a very low number like 1 or 2. If instead it looks like a large number like x'01000000', then you are dealing with data in a little endian format.

Usage Examples:

```
>java MQOptions

The current platform you are running on is Big-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


>java MQOptions

The current platform you are running on is Big-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)  0x00002010

open options for hex value 0x00002010 converts to:
0x00000010  MQOO_OUTPUT
0x00002000  MQOO_FAIL_IF_QUIESCING
```

MQTRCFRMT Manual

Program name: mqtrcfrmt
Author: Tim Zielke - CICS/MQ Systems Programmer for Alight Solutions

Description:
MQ diagnostic tool to expand data structures and fields of a distributed MQ trace generated from strmqtrc, a z/OS user parameter (API) trace, or an Application Activity Trace with amqsactz (see note below). For unix platforms, the trace must first be formatted with dspmqtrc, before being used with mqtrcfrmt. Refer to the MQ manual for more information on how to create distributed traces with strmqtrc or how to create z/OS user API traces, if needed.

The trace formatting used in this program was tested against MQ 7.1, 7.5, 8.0, and 9.0 traces. It probably can format early version/releases of MQ traces (i.e. 7.0).

NOTE: Going forward from 2017, the primary focus of the testing and support of this mqtrcfrmt tool will just be with Linux x86. The Windows and Solaris executables will continued to provided (as long as I continue to have tools to build them), but they are being "stabilized" at v8 and will have very little validation at v9 and later.

Executables Provided:
mqoptions.exe - 32 bit exe built and tested on Windows 7
mqoptions.linux - 32 bit exe built and tested on Linux x86 (compiled with large file 2GB support)
mqoptions.solaris - 32 bit exe built and tested on Solaris 10 SPARC (compiled with large file 2GB support)

Function:
This program will take a formatted distributed MQ trace, z/OS user parameter (API) trace, or Application Activity Trace from amqsactz and expand data structures and fields into a more human readable format that includes the structure fields, the numeric values of these fields, and constants that the values represent.

Application Activity Trace Note (from 2014):
Some earlier amqsact programs have a bug where hex fields like D3 get printed out as FFFFFFD3 in the hex dumps of MQ data structures (i.e. MQMD). Therefore, the mqtrcfrmt program can not format correctly an Activity Trace report from these amqsact programs. amqsactz (source code provided in MH06 supportpac) can correctly create the AAT report without this formatting bug, as can later corrected amqsact programs.

Distributed data structures and fields that are expanded:
- Actual Name (message property)
- Bufferlength
- CallbackDesc
- CbContext
- Close Options
- Compcode
- ConnectOpts (Options field only)

Consumer Control Options
CrtMsgHOpts
Datalength
DltMsgHOpts
DltPropOpts
Getmsgopts
HConn
HObj
Hmsg
InqPropOpts
Msgdesc
Objdesc
Open Options
PropDesc
Putmsgopts
Reason
Subdesc
StatusData
SetPropOpts
Type (message property)
Type (status)
Value (message property)

z/OS control blocks that are expanded:
OD
MD
GMO
PMO
CBD
SD

Application Activity Trace data structures expanded:
MQCNO (Options field only)
MQMD
MQGMO
MQPMO
MQCBD
MQCBC
MQSD

Inputs:
(1) input-trace-file: (required and must be first input)
Descripton: Name of input trace file for mqtrcfrmt to format. For Solaris/Linux platform, it should be formatted prior with dspmqtrc.

(2) output-trace-file: (required and must be second input)
Description: Name of formatted output trace file that the program creates.

Following are optional inputs that can appear in any order and must follow the previous two required inputs. They are specified with a switch.

-e endianness:
Description: Endianness of server that the trace was run on. The endianness of the trace is determined to be the server that mqtrcfrmt is running on, if -e is not specified. Valid values are case sensitive. Valid Values: (little, big)

-o operating-system:
Description: Operating system that the trace was run on. Valid values are listed below. The unix value includes linux. Values are case sensitive. unix is the default for the linux or solaris executable. windows is the default for the windows executable. Valid Values: (unix, windows, zos, aat)

-p pointer-length:
Description: The pointer byte length for data type MQPTR. Valid values are 4 and 8. 8 is the default for unix. 4 is the default for windows and zos. Valid Values: (4, 8)

-m message-parsing:
Description: Message parsing will parse any message data that it finds as if it was a certain CCSID, and then produce information on the parsing. It will provide parsing information such as how many ASCII bytes were found, what multi-byte characters were found and their byte position in the message, invalid bytes that were found and their byte position in the message, etc. Valid Values: (819, 1200, and 1208). See Message Parsing section below for more details and examples.

-s message-search:
Description: Allows user to search for message data (hex or string) in a distributed trace or activity trace. This functionality is helpful as it is sometimes difficult to search for hex or string message data due to how the data is formatted or line wrapped in trace output. Examples of the search string can be a hex search like “0x4546” or a string search like “fox”. If a match is found, a line will be inserted in the mqtrcfrmt output trace file (e.g. AMQ5488.0.FMT2) that includes the text “msgSearch: hit at message offset x”. The offset will tell you where in the message the match was found, starting at that offset.

Examples:

1) Use mqtrcfrmt to format a Windows trace on the Windows platform:
```
mqtrcfrmt.win.exe AMQ5488.0.TRC AMQ5488.0.TRC.FMT
```

2) Use mqtrcfrmt to format a Solaris trace on the Windows platform:
```
mqtrcfrmt.win.exe AMQ5488.0.FMT AMQ5488.0.FMT2 -o unix -e big
```

3) Use mqtrcfrmt to format a z/OS trace on the Windows platform:
```
mqtrcfrmt.win.exe ZOS.TRC ZOS.TRC2 -o zos -e big
```

4) Use mqtrcfrmt to format a Solaris trace generated from a 32 bit app on the Solaris platform:
```
mqtrcfrmt.solaris AMQ5488.0.FMT AMQ5488.0.FMT2 -p 4
```

5) Use mqtrcfrmt to format an Application Activity Trace from amqsactz on the Windows platform:
```
mqtrcfrmt.win.exe amqsactz.out amqsactz.out2 -o aat -p 8
```

6) Use mqtrcfrmt to format a Linux trace and message parse message data as 1208 (UTF-8):
```
mqtrcfrmt.linux AMQ5488.0.FMT AMQ5488.0.FM2 –m 1208
```

MISC NOTES:

Endianness:
Endianness is needed to properly read multi-byte binary data in the trace. If you are not sure what endianness is, just run mqtrcfrmt and let it default. It will display in the output what endianness it defaulted to. Now check the output file and see if the Version fields are in the 1-10 range. If they are very large numbers, the default endianness was incorrect. Redo the mqtrcfrmt, but specify the opposite endianness that was used/displayed in the run before with the -e option.

MQPTR:
The following data type lengths are assumed for the trace parsing. The byte size of MQPTR can be adjusted with the -p option.

- MQLONG - 4 bytes
- MQHOBJ - 4 bytes
- MQPTR - 4 bytes (default for windows, zos, and aat)
- MQPTR - 8 bytes (default for unix)

Struc-Id:
Structures are not expanded if the eye-catcher text is not found in the StrucId. For example, Msgdesc: will not be expanded if the text located in the StrucId is not "MD ".

Negative numbers in output of certain fields:
I have seen cases where an Options field like the GMO can have a value that has an extraneous bit set where that bit is the highest bit. For example, the value x'02004004' appeared as x'82004004' in the trace. x'80000000' is not a valid bit for a GMO, so MQ is probably ignoring that high order bit. However, a 4 byte signed binary field like a long in C will consider that a negative number. Options fields are listed as MQLONG which is a signed long field, so technically that is a negative number when it is displayed. This long winded explanation is basically saying that you may see negative numbers for some fields like Options fields, if the program "mistakenly" set the highest 4 bits to 8 - F. However, the constant values should still come out correctly, based on my testing.

Special trace formatting for certain fields:
The following fields are expanded with a special = character after the field name:

- Actual Name (message property)
- Bufferlength

Compcode
Datalength
HConn
HObj
Hmsg
Options (applies for any Options field)
Reason
Type (message property)
Type (status)
Value (message property)

This allows for a user customizable summary trace to be created from the formatted trace. This can be very helpful in reading an API trace.

Here is an example of an egrep command on Linux that can be used to build a summary API trace:

```
egrep '( >>| <<|Hconn=|Hobj=|Compcode=|Reason=|Hmsg=|Actual
Name=|Value=|Options=|Type=|ObjectName |Persistence )' AMQ10913.0.FMT2
```

Here is an example of a findstr command on Windows that can be used to build a summary API trace:

```
findstr ">> << Hconn= Hobj= Compcode= Reason= Hmsg= Actual Name= Value=
Options= Type= ObjectName ResolvedQName Persistence" AMQ10913.0.TRC2
```

Message Parsing:
Message parsing will analyze trace message data at the byte level with the assumption that it has the layout of a given CCSID, and then produce information on that byte analysis. It will provide byte analysis information such as how many ASCII bytes were found, what multi-byte characters were found and their byte position in the message, invalid bytes that were found and their byte position in the message, etc. Valid values to use are 819 (ISO-8859-1), 1200 (UTF-16) and 1208 (UTF-8). For strmqtrc tracing, the message parser will also display the Format, CCSID, and API call that was active when the message Buffer was printed out. For activity trace, the Format, CCSID and API call will not be included. For 1200, the endianness of the trace (endianness determined by -e switch or platform endianness that mqtrcfrmt detected and explicitly reported) is used to parse the 1200 message data. Below are some examples of what you will find with message parsing in the formatted trace output.

819 example:
Here is a strmqtrc trace that included an 819 message that has been message parsed as 819. ISO-8859-1 is a single byte code page, but it does support non-ASCII characters (bytes > x'7F'), and also has an undefined range of bytes between 0x00 - 0x1F and 0x7F - 0x9F. Bytes in the undefined range are flagged under the Inv (Invalid) count. Sometimes, having an 819 encoded message with bytes in the undefined range can cause downstream data conversion issues. For example, having a byte in the message like 0x81 can cause errors with MQ Java (v8 and higher) if you read in that message to a Java String and then try to later write the string message.

```
15:27:48.397087    26644.1      SHCON:1400005      Buffer:
15:27:48.397089    26644.1      SHCON:1400005        0x0000:  61626364
65046667 6869f16a 6b6c6d6e  |abcde.fghi.jklmn|
```

```
15:27:48.397089    26644.1      SHCON:1400005        0x0010:  6f847071
72737475 76777879 7a        |o.pqrstuvwxyz   |
msg-parser  UTF-8 Totals: Line:585 Pid:26644.1 Format:MQSTR    CCSID:819
API:MQGET << Byte:29 ASCII:27 MB2:0 MB3:0 MB4:0 Inv:2
msg-parser Byte Analysis: Line:585 a-MB4,b-INV,11-INV,
```

1200 example:
Here is a strmqtrc trace that included a 1200 message of "fox" or x'0066006F0078' followed by a surrogate pair of x'D801DC37'. The msg-parser UTF-16 line shows that the message had 10 bytes, with 3 ASCII characters, one surrogate pair (SP), and no invalid bytes (Inv). The msg-parser Byte Analysis line says that a surrogate pair was found at byte offset 6 (6-SP) in the message.

```
14:49:27.664265    29003.1       CONN:1400006      Buffer:
14:49:27.664269    29003.1       CONN:1400006        0x0000:  0066006F
0078D801 DC37              |.............|
msg-parser UTF-16 Totals: Line:128 Pid:29003.1 Format:MQSTR    CCSID:0
API:MQPUT >> Byte:10 ASCII:3 SP:1 Inv:0
msg-parser Byte Analysis: Line:128 6-SP,
```

1208 example:
Here is a strmqtrc trace that included a 1208 message of "fox" or x'666f78' followed by a multi-byte encoding of a Korean character x'eab080'. The msg-parser UTF-8 line shows that the message had 6 bytes, where the ASCII characters that were found was three, the multi-byte 2 characters (MB2) was zero, the multi-byte 3 characters (MB3) was one, the multi-byte 4 characters (MB4) was zero, and the invalid byte count (Inv) was zero. The msg-parser Byte Analysis line shows that a multi-byte character of 3 bytes was found at starting byte offset 3 (3-MB3) in the message.

```
14:49:27.664265    29003.1       CONN:1400006      Buffer:
14:49:27.664269    29003.1       CONN:1400006        0x0000:  666f78ea b080
|.............|
msg-parser  UTF-8 Totals: Line:124 Pid:29003.1 Format:MQSTR    CCSID:0
API:MQPUT >> Byte:6 ASCII:3 MB2:0 MB3:1 MB4:0 Inv:0
msg-parser Byte Analysis: Line:124 3-MB3,
```

msg2File:
If you insert a special tag "msg2File-" above a "Buffer:" line in a strmqtrc trace or a "Message Data:" line in an activity trace, the mqtrcfrmt program will write the bytes of the message to a file (max length of file name is 20 and must contain only alphanumeric characters) 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-msg1p29003
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 msg1p29003 will be written out in your current directory that contains the bytes of the message in the Buffer.

NOTE: Make sure when you add your line for the msg2File in the strmqtrc trace, the line contains a timestamp, pid.tid, etc. as in the example above. This is important as the pid.tid appearing in the line is a key piece of text that mqtrcfrmt is looking for to recognize a line as having trace data in a strmqtrc trace.

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. MQFile2Msg will read in a file into a byte array and then PUT the file contents as a message to a specified queue. Optionally, you can provide the queue manager name, format, ccsid, and encoding for the message. The MQFile2Msg is written as a local bindings connection program. Here is an example of running MQFile2Msg on Linux:

```
> export CLASSPATH=/opt/mqm/java/lib/com.ibm.mq.jar:

> /opt/mqm/java/jre64/jre/bin/java -Djava.library.path=/opt/mqm/java/lib64
MQFile2Msg –q TCZ.TEST1 –f msg1p29003 –m QM1 –o MQSTR –c 819 –e 273
```

MQAPITRCSTATS Manual

Program name: mqapitrcstats
Author: Tim Zielke - CICS/MQ Systems Programmer for Alight Solutions

Description:
MQ diagnostic tool to provide a summary report of API statistical data from a distributed MQ trace.

The trace parsing used in this program was built against the 7.1, 7.5, and 8.0 traces. It does not work against a 7.0 trace, as the trace formatting was changed between 7.0 and 7.1.

Executables Provided:
mqapitrcstats.exe - 32 bit exe built and tested on Windows 7
mqapitrcstats.linux - 32 bit exe built and tested on Linux x86 (compiled with large file 2GB support)
mqapitrcstats.solaris - 32 bit exe built and tested on Solaris 10 SPARC (compiled with large file 2GB support)

NOTE: Going forward from 2017, the primary focus of the testing and support of this mqapitrcstats tool will just be with Linux x86. The Windows and Solaris executables will continued to provided (as long as I continue to have tools to build them), but they are being "stabilized" at v8 and will have very little validation at v9 and later.

Function:
This program will take a formatted distributed MQ trace on Solaris SPARC, Linux x86, or Windows, and produce a report that gives statistical information like how many GETs were done by queue for a given thread, what was the average GET response time, summary statistics for the entire process, etc.

It produces statistical information for the following APIs:
MQOPEN
MQCLOSE
MQGET
MQPUT
MQPUT1

By thread id, it gives the following statistical information for each API:

MQOPEN (grouped by queue name and HCONN):
Total Ok
Total Ok Elapsed Time
Avg Ok Elapsed Time
Total Warning/Failed
Total Warning/Failed Elapsed Time
Avg Warning/Failed Elapsed Time

MQCLOSE (grouped by HCONN):
Total Ok
Total Ok Elapsed Time
Avg Ok Elapsed Time

Total Warning/Failed
Total Warning/Failed Elapsed Time
Avg Warning/Failed Elapsed Time

MQGET/MQPUT/MQPUT1 (grouped by queue name):
Total Ok
Total Ok Elapsed Time
Avg Ok Elapsed Time
Total Warning/Failed
Total Warning/Failed Elapsed Time
Avg Warning/Failed Elapsed Time
Total Bytes

For multithreaded traces, a summary for the entire process appears at the bottom of the report which has the following for these 5 APIs:
Total Ok
Total Ok Elapsed Time
Avg Ok Elapsed Time
Total Warning/Failed
Total Warning/Failed Elapsed Time
Avg Warning/Failed Elapsed Time
Total Bytes (if applicable)

How does it determine the API response time from the trace? In an API trace you have the trace records before the entry to the API call and the trace records after the exit from the API call. This program gets the last trace record from the entry and the first (actually it is probably the second) record from the exit and subtracts the values to get a response time.

PARSING/WARNING ERRORS section in the report:
If mqapitrcstats is not able to find an MQ API begin (i.e. MQGET >>) that corresponds to an MQ API end (i.e. MQGET <<) in the API trace, this will be reported as a parsing warning or error. This could happen if the trace was turned on during an MQGET call. Just make note of this section, as it is helping you to know that there may be API calls that will not be accounted for in your report.

Inputs:
(1) input-trace-file: (required and must be first input)
Description: Name of input trace file. For Solaris/Linux platform, it should be formatted prior with dspmqtrc.

(2) output-report-file: (required and must be second input)
Description: Name of statistics report that program creates.

Following are optional inputs that can appear in any order and must follow the previous two required inputs. They are specified with a switch.

-e endianness:

Description: Endianness of server that the trace was run on. The endianness of the trace is determined to be the server that mqtrcfrmt is running, on if -e is not specified. Valid values are case sensitive. Valid Values: (little, big)

-o operating-system:
Description: Operating system that the trace was run on. Valid values are listed below. The unix value includes linux. Values are case sensitive. unix is the default for the linux or solaris executable. windows is the default for the windows executable. Valid Values: (unix, windows)

Examples:
1) Use mqapitrcstats to get stats report from Linux trace on Linux platform:
```
mqapitrcstats.linux AMQ5488.0.FMT AMQ5488.0.FMT.rpt
```

2) Use mqapitrcstats to get stats report from Windows trace on Linux platform:
```
mqapitrcstats.linux AMQ5488.0.TRC AMQ5488.0.TRC.rpt -o windows
```

3) Use mqapitrcstats to get stats report from Solaris trace on Windows platform:
```
mqapitrcstats.win.exe AMQ5488.0.FMT AMQ5488.0.FMT.rpt -o unix -e big
```

Usage Notes:
Endianness is needed to properly read multi-byte binary data in the trace. If you are not sure what endianness is, just run mqapitrcstats and let it default. It will display in the output what endianness it defaulted to. Now check the output file and see if the Version fields are in the 1-10 range. If they are very large numbers, the default endianness was incorrect. Redo the mqapitrcstats, but specify the opposite endianness that was used/displayed in the run before with the -e option.

WARNING!!! - Tracing in general adds significant overhead to performance. It would be unwise to take a distributed trace and extrapolate that in general it is giving an accurate picture of how your application is performing from an MQ API response time stand point.

However, there is a specific type of trace that I call the API program trace (explained below) where the mqapitrcstats program can be leveraged to help shed some light to how the MQ queue manager is performing when gross performance application issues are being reported. Also, based on some testing I have done when you compare the performance numbers between API program tracing and an API exit that is just tracking API performance in memory (i.e. no I/O overhead), I have found the API program tracing numbers to be accurate within millisecond accuracy when compared to the API exit performance numbers.

What is an MQ distributed API program trace? A strmqtrc that is run with the "-t api -p prgm" option. This type of trace reduces the tracing to just the MQ stub code inside the specified program and is a much more light-weight trace than a "-t all" trace.

Here is an example of how to use an API program trace for a local bindings application:

Use Case:

A developer is reporting to the MQ administrator that their application named prgm1 which locally binds to the queue manager is taking 60 seconds to process an application task that mainly does MQ GETs and PUTs from an MQ perspective, and they say the reason for the application slowness is that MQ is slow. The following can be done to help see if the MQ queue manager is playing a part or not in the performance issue. Let's assume this is happening on a Linux server.

Run a 5 minute API program trace on prgm1:

```
strmqtrc -m qmgr -t api -p prgm1
```

Use dspmqtrc to format the trace:

```
dspmqtrc AMQ12345.0.TRC > AMQ12345.0.FMT
```

Use mqapitrcstats to get a summary performance report:

```
mqapitrcstats AMQ12345.0.FMT AMQ12345.0.FMT.rpt
```

The report shows that prgm1's GETs and PUTs are actually in the .000x range. This is enough factual data to get the developer to look into other places in their application and they find out that something outside of MQ was the actual culprit for the application response time issues.

amqsactz

amqsact is an IBM supplied sample for formatting activity trace records. amqsactz takes the amqsact sample and provides some usability enhancements for working with activity trace data. The documentation for how to use amqsactz (also listed below) is located in the top part of the amqsactz source code. There are also some examples of how to build an amqsactz executable, in this documentation. The build examples are just taken from the MQ manual, and the recommendation is just to use the MQ manual to build your executable. A Linux x86_64 executable called amqsactz.linux is included as a sample executable.

NOTE: amqsactz was built against a v8 sample of amqsact. There was a review of amqsactz against the v9 amqsact sample, but it does not include new v9 functionality like the ability to subscribe to topics for activity trace data.

amqsactz user documentation:

```
/*****************************************************************/
/*                                                               */
/*   Example of how to use amqsactz on Linux:                    */
/*   1. amqsactz -r -b > amqsactz.out                            */
/*   2. grep 1LS= amqsactz.out > amqsactz_1LS.out                */
/*   3. amqsactz -v -b > amqsactz_v.out                          */
/*                                                               */
/*   Step 1 will create summary output of each activity trace    */
/*   record, followed by reports (generated by -r) that summarize*/
/*   all of the activity trace information. The -b switch causes */
/*   the activity trace messages to be browsed, and not consumed.*/
/*                                                               */
/*   Step 2 will produce an API trace.  Each line will represent */
/*   one API call. We grep for 1LS=, since each API line will have*/
/*   that piece of text on its line.                             */
/*                                                               */
/*   Step 3 will produce a verbose report for each activity trace*/
/*   record.  The RecordNum that appears in the first two reports*/
/*   for each activity trace record/API call, can be used for    */
/*   indexing into this verbose report.                          */
/*                                                               */
/*   amqsactz has the following parameters                       */
/*                                                               */
/*   amqsactz  [-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 API fields for API line  */
```

```
/*             [-u]              # Abstract Connection Id for       */
/*                                 readability in API line         */
/*             [-r]              # Create summary reports for non-  */
/*                                 verbose run                     */
/*                                                                 */
/*   -c CCSID was put in so I could control how the data was being  */
/*   converted on the GET for something specific I was looking at.  */
/*   You probably can ignore this switch.                          */
/*                                                                 */
/*   -f is a numeric value to represent what fields to show in     */
/*   the API summary line. Add the values below to get the fields  */
/*   to list (i.e. -f 7 would list Pid, Tid, CnId).                */
/*                                                                 */
/*   To display all fields below a certain field (largest field    */
/*   inclusive), multiply the largest field by 2 and subtract 1.   */
/*   ex. (2048 x 2) - 1 = 4095 to see all fields from Pid - Appl   */
/*                                                                 */
/*   The default value is 1023 (the first 10 fields in the list).  */
/*                                                                 */
/*   Value        Short Name   Long Name                           */
/*   ==========   ==========   =========                           */
/*            1   Pid          Pid                                */
/*            2   Tid          Tid                                */
/*            4   CnId         Connection Id                      */
/*            8   Chl          Channel Name                       */
/*           16   Date         Date                               */
/*           32   Time         Time                               */
/*           64   Opr          API Operation                      */
/*          128   RC           Reason Code                        */
/*          256   HObj         Object Handle                      */
/*          512   Obj          Object Name                        */
/*         1024   Qmgr         Queue Manager Name                 */
/*         2048   Appl         Application Name                   */
/*         4096   User         User Id                            */
/*         8192   MgId         Message Id                         */
/*        16384   CrId         Correlation Id                     */
/*        32768   OQNm         Object Queue Manager Name          */
/*        65536   RQNm         Resolved Queue Name                */
/*       131072   RQMN         Resolved Queue Manager Name        */
/*       262144   RLQN         Resolved Local Queue Name          */
/*       524288   RLQM         Resolved Local Queue Manager Name  */
/*      1048576   RToQ         Reply To Queue                     */
/*      2097152   RTQM         Reply To Queue Manager             */
/*      4194304   Prty         Priority                           */
/*      8388608   Prst         Persistence                        */
/*     16777216   Rprt         Report Options                     */
/*     33554432   Expr         Expiry                             */
/*     67108864   MsgL         Message Length                     */
/*    134217728   CCSI         Coded Character Set Id             */
/*    268435456   Encd         Encoding                           */
/*    536870912   MDAB         Message Data As Bytes              */
/*   1073741824   MDAC         Message Data As Character          */
/*                                                                 */
/*   -g is a numeric value to represent what fields to show in     */
/*   the API summary line. Add the values below to get the fields  */
/*   to list (i.e. -g 7 would list CnOp, OpOp, GtOp).              */
/*                                                                 */
```

```
/*   To display all fields below a certain field (largest field     */
/*   inclusive), multiply the largest field by 2 and subtract 1.    */
/*   ex. (128 x 2) - 1 = 255 to see all fields from CnOp - SROp     */
/*                                                                  */
/*   The default value is 0 (no fields).                            */
/*                                                                  */
/*   Value        Short Name   Long Name                            */
/*   ==========   ==========   =========                            */
/*            1   CnOp         Connection Options                   */
/*            2   OpOp         Open Options                         */
/*            4   GtOp         Get Options                          */
/*            8   PtOp         Put Options                          */
/*           16   ClOp         Close Options                        */
/*           32   CbOp         Callback Options                     */
/*           64   SbOp         Subscription Options                 */
/*          128   SROp         Subscription Request options         */
/*          256   OprD         Queue Manager Operation Duration     */
/*                                                                  */
/*   -u makes the Connection Id that appears in the API lines       */
/*   easier to read. Each unique connection id is converted to a    */
/*   a smaller and more readable unique number.                     */
/*                                                                  */
/*   -r summary reports includes the following reports that help    */
/*   summarize the activity trace data. This -r switch is ignored   */
/*   if the -v switch is active.                                    */
/*   Application Summary Report                                     */
/*   Application Objects Referenced Report                          */
/*   Application Objects Detail Report                              */
/*   Application Channels Referenced Report                         */
/*   Application Operations Executed Report                         */
/*   Application Operations Options Report                          */
/*   Application Operations Reason Code Report                      */
/*                                                                  */
/*   amqsactz server build examples (better to use MQ manual):      */
/*   Linux x86 build example when MQ install path is /opt/mqm:      */
/*   gcc -m64  -o amqsactz amqsactz.c -I/opt/mqm/inc -L/opt/mqm/lib64  */
/*   -Wl,-rpath=/opt/mqm/lib64 -Wl,-rpath=/usr/lib64 -lmqm_r -lpthread */
/*                                                                    */
/*   Solaris SPARC build example with MQ install path of /opt/mqm:    */
/*   gcc -m64 -o amqsactz amqsactz.c -I/opt/mqm/inc -L/opt/mqm/lib64  */
/*   -R/opt/mqm/lib64 -R/usr/lib/64 -lmqm -lsocket -lnsl -ldl         */
/*                                                                  */
/*   Windows Visual Studio build example when MQ install path is    */
/*   "c:\Program Files\IBM\WebSphere MQ":                           */
/*   cl -MD amqsactz.c -Famqsactz_win.exe                           */
/*    "c:\Program Files\IBM\WebSphere MQ\tools\Lib\mqm.Lib"         */
/********************************************************************/
```

Activity Trace Diagnostic Aids:

The activity trace can provide a hex dump of some of the API data structures (e.g. GMO. PMO, etc.) with an activity trace. The TraceLevel needs to be HIGH to get these data structures in the activity trace. The mqtrcfrmt program provided in this supportpac can also format most of these API data structures into a more human readable format. See the mqtrcfrmt documentation for more details.

One of the API data structures that is provided by the activity trace is the MQCD (channel definition) for client channels. The mqtrcfrmt program does not format the MQCD, but an example of its layouts with field names and offsets is provided below on the next few pages. This example is for an MQCD generated from a 64 bit queue manager, as the pointers in the MQCD are 8 bytes.

```
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  '                '
 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  '................'
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  '................'
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  '................'
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)
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  '................'
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 [2] (+700 for 8)
    MsgCompList [16] +708 for 64)
00000710:  FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  '................'
00000720:  FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  '................'
00000730:  FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  '................'
00000740:  FFFF FFFF FFFF FFFF 0000 0000 0000 0000  '................'
    CLWLChannelRank (+748 for 4) = 0
    CLWLChannelPriority (+74C for 4) = 0
00000750:  3200 0000 FDFF FFFF FDFF FFFF 0A00 0000  '2...............'
    CLWLChannelWeight (+750 for 4) = 50
    ChannelMonitoring (+754 for 4) = -3
    ChannelStatistics (+758 for 4) = -3
    SharingConversations (+75C for 4) = 10
00000760:  0000 0000 FFC9 9A3B FFC9 9A3B 0000 0000  '.......;...;....'
    PropertyControl (+760 for 4) = 0
    MaxInstances (+764 for 4) = -1 = 999,999,999
    MaxInstacnesPerClient (+768 for 4) = 999,999,999
    ClientChannelWeight (+76C for 4) = 0
00000770:  0100 0000 8813 0000 0200 0000 0000 0000  '................'
    ConnectionAffinity (+770 for 4) = 1
    BatchDataLimit (+774 for 4) = 5000
    UseDLQ (+778 for 4) = 2
    DefReconnect (+77C for 4) = 0
00000780:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
    CertificateLabel (+780 for 64)
00000790:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
000007A0:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
000007B0:  2020 2020 2020 2020 2020 2020 2020 2020  '                '
```

mqsc-qmgrs

mqsc-qmgrs is a Unix bash script that executes “runmqsc -c” commands (runmqsc client mode) against a user defined group of queue managers, and then collects the runmqsc output into two output files. The first output file will have the results from the different queue managers appended together. The second output file will take the first output file, but then flatten all the individual results into one line. The second output file is helpful if you wanted to grep the individual runmqsc results.

The mqsc-qmgrs bash script was tested on a RHEL 6 x86 server.

The mqsc-qmgrs includes more usage documentation and examples at the top of the script.

“runmqsc -c” commands are executed by this script, so you need to be using an MQ client software that supports “runmqsc -c” (e.g. MQ v8).

mqsc-qmgrs has more uses than just as a trace tool, but one trace tool usage would be to use it to turn the activity trace on/off for a group of queue managers.