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

Quick Start Examples:

Java mqtrcfrmt jar (needs Java 1.8 JRE minimum)
1) Use mqtrcfrmt to format a Windows strmqtrc trace

```
> java -jar mqtrcfrmt.jar -i AMQ5488.0.TRC -t windows
```

2) Use mqtrcfrmt to format a Linux strmqtrc trace

```
> java -jar mqtrcfrmt.jar -i AMQ5488.0.FMT -t unix
```

3) Use mqtrcfrmt to format a list of Linux strmqtrc traces

```
> ls AMQ*.FMT > AMQTRC
> java -jar mqtrcfrmt.jar -i AMQTRC -t unix
```

4) Use mqtrcfrmt to format the verbose (-v) output from the amqsact Application Activity Trace sample.
NOTE: mqat.ini must have TraceLevel=HIGH for there to be any MQ data structures like MQGMO to parse.

```
> java -jar mqtrcfrmt.jar -i amqsact.out –t aat
```

LEGACY C mqtrcfrmt executables removed - v7/v8
NOTE: Legacy C mqtrcfrmt executables for v7/v8 have been removed from the MH06 supportpac, since v7/v8 are no longer supported MQ versions. The java mqtrcfrmt program will probably be able to parse v7/v8 (and probably older) versions of MQ traces, but this has not been validated.

MQTRCFRMT Manual

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

Description:
Mqtrcfrmt is a trace formatting tool that can take an IBM MQ distributed strmqtrc trace, and expand the MQ data structures (e.g. MQGMO) and certain trace fields to make them more readable. It is somewhat self-explanatory how things are expanded, so the user is encouraged to just use the tool to get comfortable with what formatting aids is provided for strmqtrc traces. There are also more details in the manual below.

NOTE: For unix platforms, the strmqtrc 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.

The mqtrcfrmt tool can also provide helpful formatting aids with the verbose (-v) output from the amqsact Application Activity Trace sample. Consult the Application Activity Trace section below for more details on what help is provided.

Mqtrcfrmt can also format non-standard traces like the z/OS user parameter (API) trace that contain MQ data structures (e.g. MQGMO) in a hex dump format. You will need to use the -w and -x options to specify the width of the dump and what parsing prefix should be used for the hex lines of the dump. Consult the -w and -x documentation for more details.

USAGE NOTE:
The trace formatting used in this program was tested against IBM MQ 9.0, 9.1, and 9.2, and mainly on the Linux platform. The mqtrcfrmt tool can probably also format older version/releases of MQ traces (e.g. 7.5, 8.0). The Windows trace formatting has not been recently validated, but the parsing logic is flexible enough where it should be able to handle minor changes to the tracing format. If a user runs across any issues with the trace formatting, you can send an e-mail to the author (the e-mail is listed on the MH06 supportpac web page) and it will be looked at as time is available.

Executables Provided:
mqtrcfrmt.jar - compiled at Java 1.8, so minimum 1.8 JRE is required

NOTE: Source code is provided in jar. Eclipse was used to build the jar, and the source can be viewed in an IDE tool like Eclipse.

mqtrcfrmt inputs:
-i inputFile (required)
Description: Name of input trace file for mqtrcfrmt to format. For Solaris/Linux platform, it should be formatted prior with dspmqtrc. If the file starts with AMQTRC, it can also be a file that includes a list of trace files (one trace file per line) to act upon.

-a printChar (optional)
Description: This is an obscure option that is best left alone for ASCII based systems. It will default to "ASCII", which basically means the string data in the dump will have non-printable ASCII bytes

converted to ".". This is similar to what you see in a strmqtrc dump. You can also set this option to "none". This will cause no conversion of non-printable bytes to ".", and would be appropriate to use if mqtrcfrmt was run on a non-ASCII based system like the z/OS mainframe. There is another supported option of "Linux_x86", but it is mainly for internal use only and should be ignored.

-b statusInterval (optional)
Description: This is a number in seconds for mqtrcfrmt to report what percentage has been completed. This is helpful when mqtrcfrmt runs for a while and you want to have a sense of when it will complete. For example, you could specify "-b 15" to have mqtrcfrmt print out every 15 seconds what percentage has been completed. The percentage is calculated by what percentage of the total input file lines have been processed.

-c inputFileCharset (optional)
Description: Charset of input file. This will default to Windows-1252 if trace type is Windows, and UTF-8 for anything else.

-e endianness (optional)
Description: Endianness of input file. The valid values are big or little. If not provided, mqtrcfrmt will determine the endianness of the platform that it is running on and use that for the endianness setting.

-f constantField (optional)
Description: Constant field to expand. It is a requirement that both -f constantField and -v constantValue are provided together, unless the special "?" option for -f is provided. If "?" is given for -f, then a list of the possible valid values will be printed. If passing in a question mark on Unix shell, you may need to surround it with quotes. The constant field and value will be used to expand to their constants.

Here is an example:

```
> java -jar mqtrcfrmt.jar -f MQPMO.Options -v 66
Field MQPMO.Options for value 66 resolves to:
   Options=MQPMO_SYNCPOINT
   Options=MQPMO_NEW_MSG_ID
```

-g msg2FileGlobal (optional)
Description: Consult msg2File documentation below for more details. This global option causes all messages to be written out in the trace.

-m messageParsing (optional)
Description: This option is supported for a traceType of unix or windows. Consult message parsing documentation below for more details. Valid values are 819, 1200, and 1208.

-o outputFileSuffix (optional)
Description: The output file suffix to append to each input trace file. The default is ".2".

-p pointerLength (optional)
Description: The size of MQPTR in the trace. The default is 8. Valid values are 4 or 8.

-r report (optional)
Description: A report that is provided at the end of each strmqtrc trace output file. The only supported value currently is API. This provides an API report of all the API calls, how many were performed, and average response time in seconds. An API report for the entire process is provided, followed by each thread.

-s messageSearch (optional)
Description: This option is supported for a traceType of unix, windows, or aat. Allows user to search for message data (hex or string) in a strmqtrc trace or amqsact verbose output. This functionality is helpful as it is sometimes difficult to search for hex or string message data due to how the data is broken across multiple lines in the 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.FMT.2, amqsact.out.2) 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.

You can also specify the -s option up to three times followed by a -s AND or OR for a multiple string AND/OR search. If the multi-string AND/OR search was successful, the line "msgSearch: multi-string match" will appear in the formatted trace. Note that the individual matches in the individual strings are also still reported (ex. "msgSearch: string2 hit at message offset 4"), so look for the "multi-string match" line for confirmation that the AND/OR multi-string search was successfully matched.
Examples for -s:
searching for messages that contain the hex bytes 4546 in a trace file
```
java -jar mqtrcfrmt.jar -i AMQ10951.0.FMT -t unix -s 0x4546
```

searching for messages that contain the text "The" in a trace file
```
java -jar mqtrcfrmt.jar -i AMQ10951.0.FMT -t unix -s The
```

searching for messages that contain the text "The" AND "quick" AND "brown" in a trace file
```
java -jar mqtrcfrmt.jar -i AMQ10951.0.FMT -t unix -s The -s quick -s
brown -s AND
```

searching for messages that contain the text "The" OR "quick" OR "brown" in a trace file
```
java -jar mqtrcfrmt.jar -i AMQ10951.0.FMT -t unix -s The -s quick -s
brown -s OR
```

-t traceType (optional)
Description: Type of input trace. Valid values are "windows" or "unix" for strmqtrc traces, and "aat" for the verbose (-v) output from amqsact. Any other value is defaulted to undefined. If -t is not specified, the default is undefined. Note that mqtrcfrmt was tested against Windows, Linux x86, and Solaris SPARC strmqtrc traces. It is possible that the unix format covers more than just Linux x86 and Solaris SPARC, but this was never tested as I only had these Unix platforms to test against.

-v constantValue (optional)
Description: See -f constantField for more details.

-w dumpLineWidth (optional)

Description: This is the width in bytes of the hex dumps in the trace. This field should only be used when you are using a -t type of undefined. A -t of windows, unix, or aat will enforce that this value can only be set to 16. If you were working with an undefined trace type that had a dump width of 32 bytes, you would provide -w 32.

-x dumpLinePrefix (optional)
Description: A prefix (e.g. 0x) of a word that will proceed each line in the hex dump. This field is only be used when you are using a -t type of undefined. In order for the trace parsing to work, there needs to be a field (e.g. 0x0000) that proceeds a hex dump line. For example, you could provide "-x 0x" for the following dump example, since all the dump lines are proceeded with a "0x":

```
0x0000:  43424320 02000000 06000000 02000000  |CBC ............|
0x0010:  00000000 00000000 00000000 00000000  |................|
0x0020:  00000000 00000000 00000000 06000000  |................|
0x0030:  00200000 00000000 cea0f81d 522b0000  |. ..........R+..|
```

Mqtrcfrmt Examples:

1) Use mqtrcfrmt to format a Windows strmqtrc trace:

```
> java –jar mqtrcfrmt.jar –i AMQ1234.0.TRC –t windows
```

2) Use mqtrcfrmt to format a Linux strmqtrc trace:

```
> java –jar mqtrcfrmt.jar –i AMQ1234.0.TRC –t unix
```

3) Use mqtrcfrmt to format the verbose (-v) output from the amqsact Application Activity Trace sample.
NOTE: mqat.ini must have TraceLevel=HIGH for there to be any MQ data structures like MQGMO to parse.

```
> java –jar mqtrcfrmt.jar –i amqsact.out –t aat
```

4) Use mqtrcfrmt to format an undefined trace type file that has MQ data structures with a hex dump width of 32 and with a leading prefix that starts with 0x before each hex line

```
> java –jar mqtrcfrmt.jar –i inputFile –x 0x –w 32
```

TIP: One Line API records in strmqtrc:
The IBM strmqtrc format provides one line API summary records that have the text MQI: in them. You can search for just these lines in a strmqtrc trace to get a one line API summary of your application.

Message Parsing:
Message parsing will analyze strmqtrc 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). The message parser will also display the Format, CCSID, and API call that was active when the message buffer was printed out. 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 "Message Data:" line in an amqsact verbose output, the mqtrcfrmt program will write the bytes of the message to a file 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 working 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. This is obviously not needed when using msg2File in amqsact verbose output, since amqsact output does not have a pid.tid in the beginning of the output lines.

NOTE: You can use the -g msg2FileGlobal switch to cause mqtrcfrmt to write out every message buffer in the trace to output files. The value that you provide with -g will be included in the output file names. If using the -g option with a traceType of unix or windows, the name of the output files will include the relevant pid.tid in the output file names. If using the -g option with a traceType of aat, the name of the files will include the name "NOTFOUND", since amqsact does not have a pid.tid format like strmqtrc.

As a convenience, a java (1.6 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.

You can also list multiple files (one line per file name) into a file named MSGFILES and have MQFile2Msg act on all those files in one invocation of the program. 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
```

mqtrcfrmt Application Activity Trace formatting for verbose output of amqsact
mqtrcfrmt will do the following on the verbose (-v) output from amqsact to make it more helpful to work with the output.

1) A unique recordnumber (e.g. REC(1)) will be placed before each Application Activity Trace record. I define a record as the output data that starts with "MonitoringType: MQI Activity Trace" and the subsequent data to the next "MonitoringType: MQI Activity Trace". This is essentially one message taken from the SYSTEM.ADMIN.TRACE.ACTIVITY.QUEUE, and represents activity trace data for a given application. The reason a unique recordnumber is being inserted into the output is that it allows the user to cross-reference one line API data (mentioned below) back to the more verbose data.

2) After each MQI Operation (e.g. MQXF_PUT) output, a line will be inserted that starts with 1LS= and is followed by relevant data for this activity trace record and MQI operation. This allows the user to grep out 1LS= API summary lines to more easily read the trace. Below is an example a 1LS= line.

```
1LS= Rec(1) Pid(30068) Tid(54) Date('2020-04-21') Time('14:08:58')
Opr(MQXF_CMIT) RC(0) Chl('MQAPP1.TLS.SERVER') CnId(23AD985E0201B32A)
HObj() Obj()
```

3) An "mqtrcfrmt Application Activity Trace report" is provided at the end of the formatted amqsact output. The report is broken out by each unique application connection id and summarizes what MQI operations were performed and how many, the operations average response time (if present in the amqsact output) in microseconds, object names and types, total message length, unique reason codes found for this operation, etc.

Notes:
Prst stands for the persistence used in the PUT/GET. 0 = not-persistent, 1 = persistent, 2 = persistent-as-q-default

Below is an example of the report:

```
mqtrcfrmt Application Activity Trace report:

qmgrName('QM1') applName('amqsput') userId('myid') pid(26150)
channel() connId(23AD985E0DC1D12B):
Opr(MQXF_CLOSE) OprCt(1) avgOprDuration(49) ObjType(MQOT_Q)
ObjName('TCZ.TEST1') reasonCds=0
Opr(MQXF_CONNX) OprCt(1) avgOprDuration(181) reasonCds=0
Opr(MQXF_DISC) OprCt(1) reasonCds=0
Opr(MQXF_OPEN) OprCt(1) avgOprDuration(123) ObjType(MQOT_Q)
ObjName('TCZ.TEST1') reasonCds=0
Opr(MQXF_PUT) OprCt(3) avgOprDuration(1584) ObjType(MQOT_Q)
ObjName('TCZ.TEST1') Prst(2) msgTotalLength(15) reasonCds=0
```

amqsactz

amqsactz (which was a C program that was an enhancement to the amqsact sample) is no longer supplied with the MH06 supportpac. The more helpful functionality of amqsactz has been moved into mqtrcfrmt and can be used with the "-t aat" option. Consult the mqtrcfrmt documentation for more details.

mqsc-qmgrs

mqsc-qmgrs (which was a Unix script that was an aid for working with runmqsc -c) is no longer supplied with the MH06 supportpac. If you are interested in using it, you can find a blog posting for it on the MQ Community web page.