Using Application Activity Trace

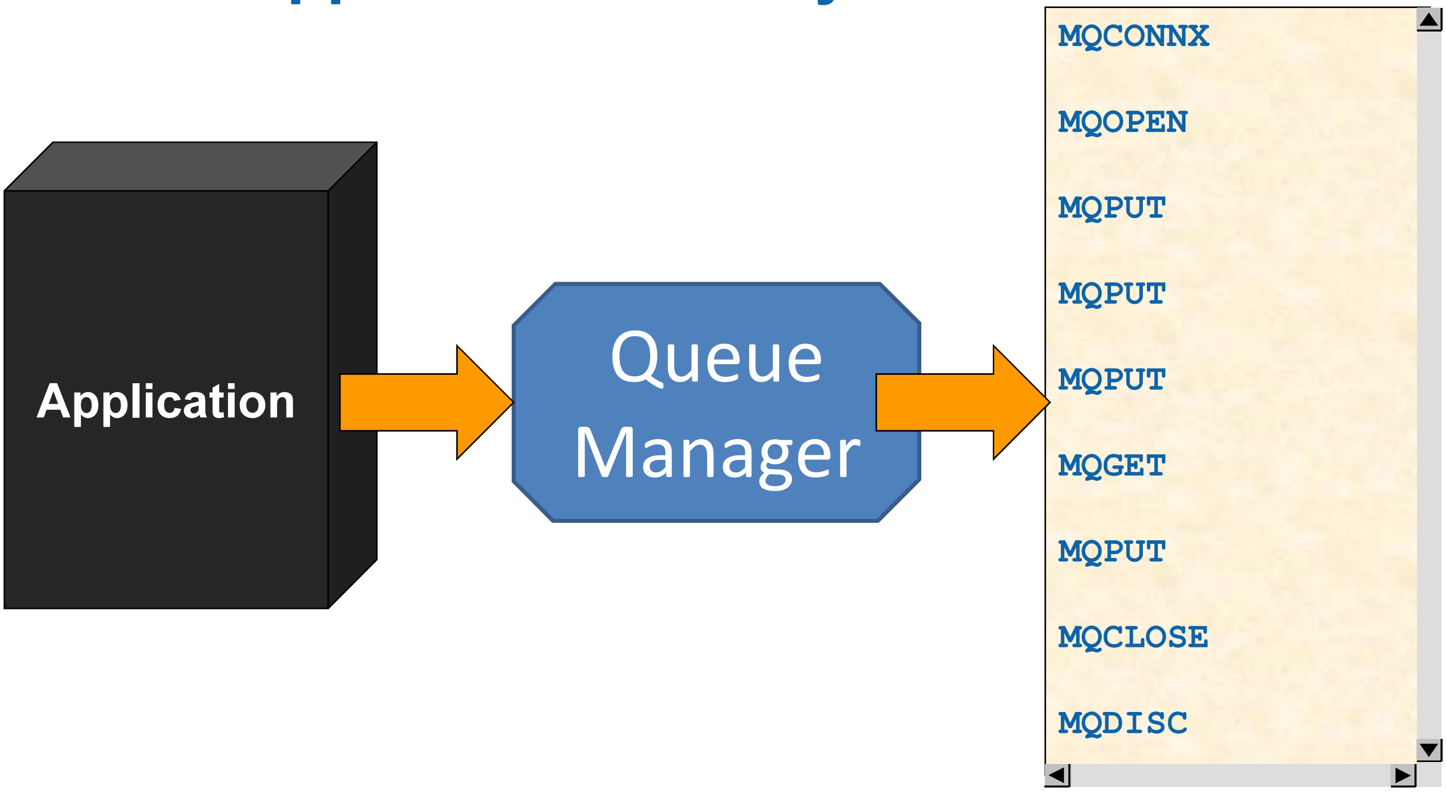
Morag Hughson – morag@mqgem.com

MQGem Software

Agenda

- What is Application Activity Trace?
- How do you enable it?
- What do you get out?
- What can you do with it?
- Activity Trace on the MQ Appliance
- Using dynamic mode subscriptions
- Tools to view the output

What is Application Activity Trace?



```
MQCONNX
MQOPEN
MQPUT
MQPUT
MQPUT
MQGET
MQPUT
MQCLOSE
MQDISC
```

What is Application Activity Trace? - Notes

N
O
T
E
S

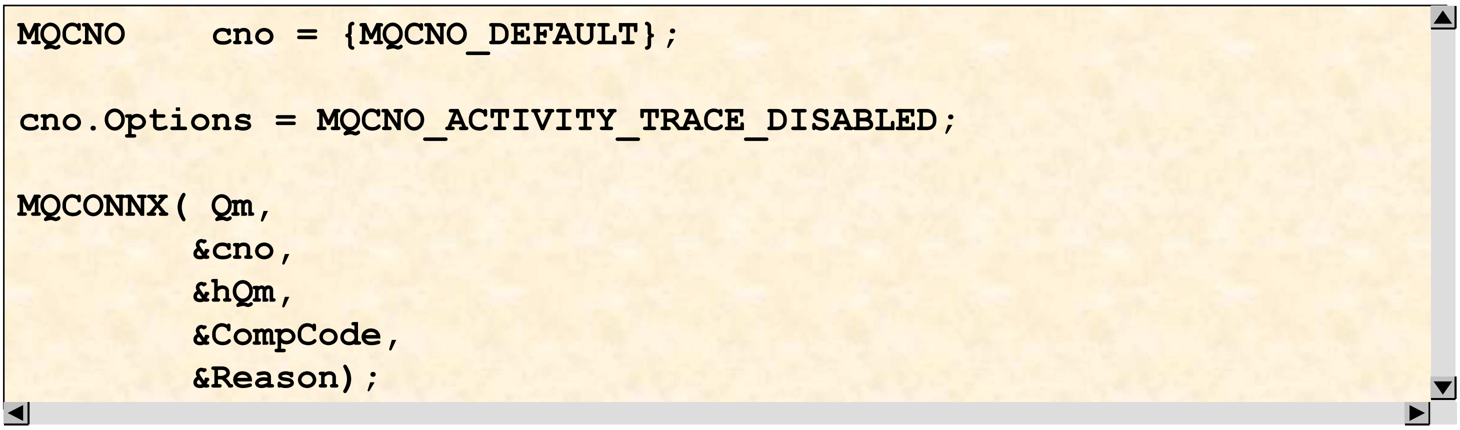
- Application Activity Trace is a feature of IBM MQ that allows you to discover exactly what the applications connected to your queue manager are doing. You can see the object names that they open and the options they use on the various verbs they call. You can find out about the size, persistence, priority and more, of your messages. Please note, this feature is only available on the Distributed platforms.
- Application activity trace produces detailed information about the behaviour of applications connected to a queue manager. It traces the behaviour of an application and provides a detailed view of the parameters used by an application as it interacts with IBM MQ resources. It also shows the sequence of MQI calls issued by an application.

Configuring the trace

- **Queue manager attributes**

ALTER QMGR ACTVTRC(ON | OFF) ACTVCONO(ENABLED | DISABLED)

- **Application Over-ride**



```
MQCNO    cno = {MQCNO_DEFAULT};

cno.Options = MQCNO_ACTIVITY_TRACE_DISABLED;

MQCONNX( Qm,
        &cno,
        &hQm,
        &CompCode,
        &Reason);
```

- **`mqat.ini` file**

Configuring the trace - Notes

N

O

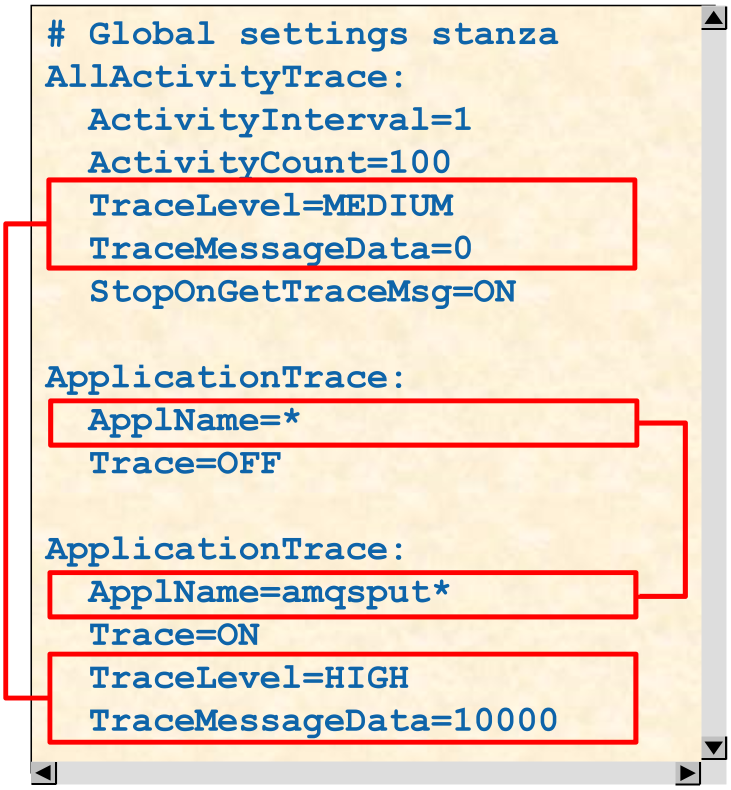
T

E

S

- To turn on Application Activity Trace, there is a control on the queue manager, the ACTVTRC attribute which can be set to ON or OFF. If this attribute is set to ON it does not mean that every application is traced however. You can further tune which applications it applies to.
- Applications can opt out of Application Activity Trace. For example if you are writing an application to collect and view the output of the trace, it is a good idea to opt-out so that you don't clutter up the trace you're trying to collect with records of the application reading the output! However, the queue manager has the final say about whether applications are allowed to opt-out with the ACTVCONO attribute which can be set to ENABLED or DISABLED.
- Should an application wish to opt-out, it uses the `MQCNO_ACTIVITY_TRACE_DISABLED` option on MQCONNX to indicate so. There is also a second connection option, `MQCNO_ACTIVITY_TRACE_ENABLED` for the opposite effect.
- The detailed configuration of Application Activity Trace takes place in the `mqat.ini` file which we'll look at on the next page.

mqat.ini file



```
# Global settings stanza
AllActivityTrace:
  ActivityInterval=1
  ActivityCount=100
  TraceLevel=MEDIUM
  TraceMessageData=0
  StopOnGetTraceMsg=ON

ApplicationTrace:
  ApplName=*
  Trace=OFF

ApplicationTrace:
  ApplName=amqsput*
  Trace=ON
  TraceLevel=HIGH
  TraceMessageData=10000
```

- **One global stanza**
- **Several per application stanzas**
  - Can over-ride global settings
  - Most specific match on `ApplName` applies

mqat.ini file - Notes

N O T E S

- The `mqat.ini` file is location in the same place as the `qm.ini` file. It is a stanza based file just like other MQ ini files such as `mqs.ini` and `qm.ini`.
- It can contain two different types of stanza. It can contain a single `AllActivityTrace` stanza, and then multiple `ApplicationTrace` stanzas allowing specific configuration for different application connections.
- Each of the `ApplicationTrace` stanzas provides configuration for how to treat specific application connections. If required you can over-ride settings in the global stanza to be different for specific application connections.
- You can use wildcards in the `ApplName`. Also, for those of you on Windows who are used to `ApplName` fields containing part of the path, you only need to provide the name from the last '\' onwards. Where several stanza could match a particular application, such as in this example, the most specific match is the one that applies. If there is more than one matching, most specific, the last matching one will be used.

Picking up `mqat.ini` changes

- **Changes take effect when…**
- **The application next connects**
- **A change is made to the queue manager object**
  - Connected applications will pick up change

Picking up `mqat.ini` changes - Notes

N O T E S

- When changes are made to the `mqat.ini` file, applications that subsequently connect after the change was made will pick up and run using the settings in the mqat.ini file. An application that is already running and will not make another connection can also be forced to pick up the changes by making an alteration to the queue manager object.
- This alteration doesn’t have to make an actual change to the queue manager, changing an attribute to the value it is already set to will suffice. Some people like to use `DESCR` for this.

Configuration Interaction

- **ACTVTRC(ON) ACTVCONO(DISABLED) + MQCNO_ACTIVITY_TRACE_DISABLED**
  - This connection still traced
- **ACTVTRC(ON) + ApplicationTrace Stanza Trace=OFF**
  - This connection not traced
- **ACTVTRC(ON) ACTVCONO(ENABLED) + MQCNO_ACTIVITY_TRACE_DISABLED + ApplicationTrace Stanza Trace=ON**
  - This connection is traced

- **Precedence**
  - `mqat.ini`
  - `MQCNO_ACTIVITY_*`
  - `ACTVTRC`

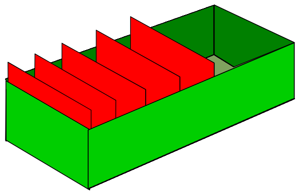
Configuration Interaction - Notes

N O T E S

- These various configuration settings interact. For example, although you've used the `MQCNO_ACTIVITY_TRACE_DISABLED` option, if `ACTVCONO(DISABLED)` is set, your connection will still be traced. You were not allowed to opt-out.
- Perhaps an obvious one, but if you have a stanza saying tracing is off, then that over-rides the `ACTVTRC(ON)` setting for your application.
- And finally a less obvious one. If you have opted-out by using `MQCNO_ACTIVITY_TRACE_DISABLED`, and you've been allowed to do so with `ACTVCONO(ENABLED)`, but there is also a stanza for the application in the `mqat.ini` file that says `Trace=ON`, then tracing will be on. The `mqat.ini` file setting wins out.
- In essence there is a precedence order of the various configuration settings. The `ACTVTRC` attribute can be overridden by the `MQCNO_ACTIVITY_TRACE_*` option settings, which again can be overridden by the settings in the `mqat.ini` file.

Application Activity Trace

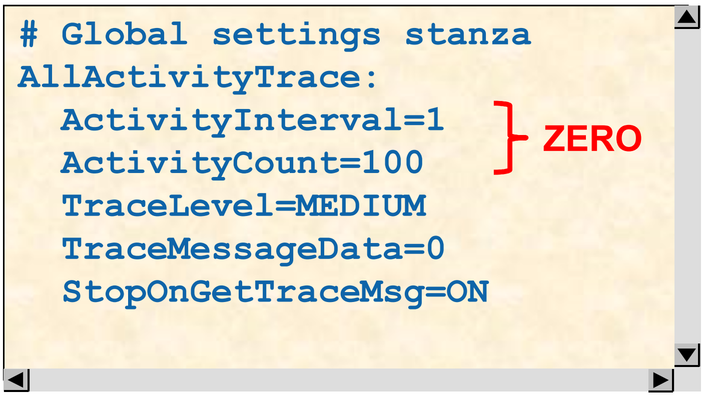
- PCF Messages written to a SYSTEM queue.



Default MAXDEPTH(3000)

SYSTEM.ADMIN.TRACE.ACTIVITY.QUEUE

- New message when...



```
# Global settings stanza
AllActivityTrace:
  ActivityInterval=1
  ActivityCount=100
  TraceLevel=MEDIUM
  TraceMessageData=0
  StopOnGetTraceMsg=ON
```

} ZERO

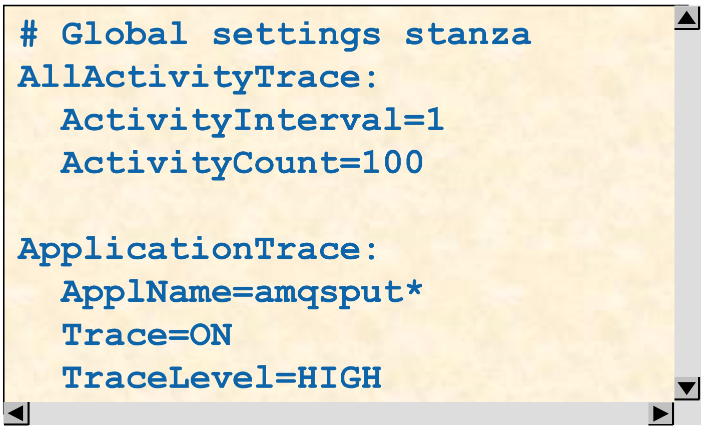
- Operations = `ActivityCount`
- Lifetime = `ActivityInterval`
- Message data = `MAXMSGL`
- Application Disconnects

Application Activity Trace - Notes

N O T E S

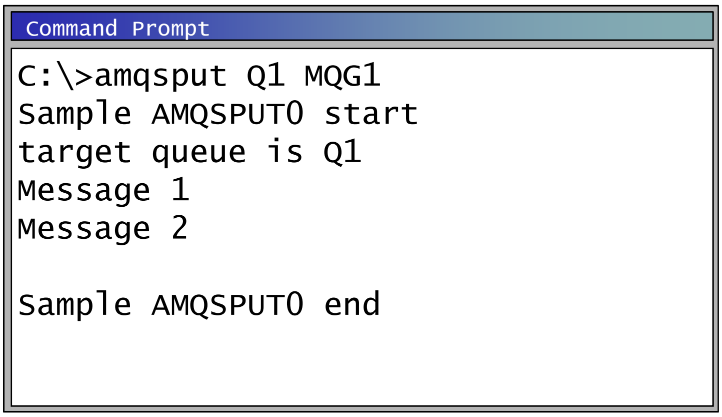
- The output from Application Activity Trace is a series of messages written to the SYSTEM.ADMN.TRACE.ACTIVITY.QUEUE.
- The messages will be written to the queue when one of the following statements is true.
  - The number of operations collected into the message exceeds the value of `ActivityCount` in the `mqat.ini` file stanza (could be taken from the global stanza)
  - The connection has been alive for longer than the value of `ActivityInterval` in the `mqat.ini` stanza (again, this could be taken from the global stanza)
  - The amount of data exceeds the maximum message size defined for the output queue.
- If `ActivityCount` is set to zero, the trace message is written when one of the other points is reached.
- If `ActivityInterval` is set to zero, the trace message is written when one of the other points is reached.
- Beware the definition of this queue comes with a MAXDEPTH(3000) (less than the MAXDEPTH(5000) on the SYSTEM.DEFAULT.LOCAL.QUEUE. If you're trying to generate a lot of trace, this may limit you. Consider making bigger messages so you get fewer of them with the above settings.

A simple example



```
# Global settings stanza
AllActivityTrace:
  ActivityInterval=1
  ActivityCount=100

ApplicationTrace:
  ApplName=amqsput*
  Trace=ON
  TraceLevel=HIGH
```



```
C:\>amqsput Q1 MQG1
Sample AMQSPUT0 start
target queue is Q1
Message 1
Message 2

Sample AMQSPUT0 end
```

```
11:27:11  1048(  1) [  284us] C:\mqm8004\bin64\amqsput.exe MQCONNX
11:27:11  1048(  1) [  543us] C:\mqm8004\bin64\amqsput.exe MQOPEN Q1
11:27:14  1048(  1) [   80us] C:\mqm8004\bin64\amqsput.exe MQPUT Q1
11:27:16  1048(  1) [   50us] C:\mqm8004\bin64\amqsput.exe MQPUT Q1
11:27:17  1048(  1) [   83us] C:\mqm8004\bin64\amqsput.exe MQCLOSE Q1
11:27:17  1048(  1) [   83us] C:\mqm8004\bin64\amqsput.exe MQDISC Q1
```

A simple example - Notes

NOTES

- Now that we've enabled Application Activity Trace, let's try a simple example. As a reminder we show the active values for this application, as we're going to run the simple amqsput sample.
- Depending on how fast you can type, you'll probably get 4 messages on your SYSTEM.ADMIN.TRACE.ACTIVITY.QUEUE as a result of running this. The first message contains the details of the MQCONNX and the MQOPEN, the next two contain an MQPUT each, and the final message contains the MQCLOSE and MQDISC. This is because we have `ActivityInterval` set to 1 second, so after 1 second it will write what has happened so far by this application as a trace message. Unless you are planning on parsing the messages yourself, this breakdown shouldn't really matter to you.
- It is worth noting that the contents of any one Application Activity Trace PCF message are all about a single MQ connection.
- With this output you can immediately see the general flow of the application. However, there is much more data in the Application Activity Trace than this.

* Example output on previous slide from MO71 Activity Trace viewer

A simple example – application data

```
Correl_id:
00000000:  414D 5143 4D51 4731 2020 2020 2020 2020  'AMQCMQG1........'
00000010:  33BA 6259 2000 2C01                      '3.bY..,.        '
QueueManager: 'MQG1'
Host Name: ‘MQGWIN1'
IntervalStartDate: '2017-07-10'
IntervalStartTime: '11:43:13'
IntervalEndDate: '2017-07-10'
IntervalEndTime: '11:43:13'
CommandLevel: 800
SeqNumber: 1
ApplicationName: 'C:\mqm8004\bin64\amqsput.exe'
Application Type: MQAT_WINDOWS_NT
ApplicationPid: 8228
UserId: ‘mqgemusr'
API Caller Type: MQXACT_EXTERNAL
API Environment: MQXE_OTHER
Application Function: ''
Appl Function Type: MQFUN_TYPE_UNKNOWN
Trace Detail Level: 3
Trace Data Length: 10000
Pointer size: 8
Platform: MQPL_WINDOWS_NT
```

EXTERNAL | INTERNAL

OTHER | MCA | MCA_SVRCONN
COMMAND_SERVER | MQSC

IBM i platform only

Application data – Notes

N

O

T

E

S

- Each message contains a set of data in it to identify the application being traced.
- Since each individual message only contains data about one MQ connection, this is included at the start of every message written for the application, so even where our short run of amqsput was broken up into 4 messages this information is included in each message. This allows applications parsing and viewing this information to correlate it as being for the same application.
- Within this set of information are things like the application name and type, and the user id running it, the process ID and “thread” (more on thread identifiers later), the environment and caller type, and a little about the queue manager involved.

* Example output on previous slide from amqsact sample

A simple example – API calls

```
MQOPEN(
   Hobj          :2 QUEUE(MQG1/Q1)
   Open Options :00002010
                 00002000 MQOO_FAIL_IF_QUIESCING         (Fail if quiescing)
                 00000010 MQOO_OUTPUT                    (Output)
   CompCode      :0
   Reason        :0     OK.
  )
MQPUT(
   Hobj          :2 QUEUE(MQG1/Q1)
     [  128 bytes] Put Options (MQPMO)
     StrucId       :'PMO '
     Version       :1
     PMO Options  :00002044
                   00002000 MQPMO_FAIL_IF_QUIESCING        (Fail if quiescing)
                   00000040 MQPMO_NEW_MSG_ID               (New message id)
                   00000004 MQPMO_NO_SYNCPOINT             (No syncpoint)
     Resolved Q    :'Q1                                           '
     Resolved Qmgr:'MQG1                                          '
   Data Length   :9
   Message       :'Message 1'
   CompCode      :0
   Reason        :0     OK.
  )
```

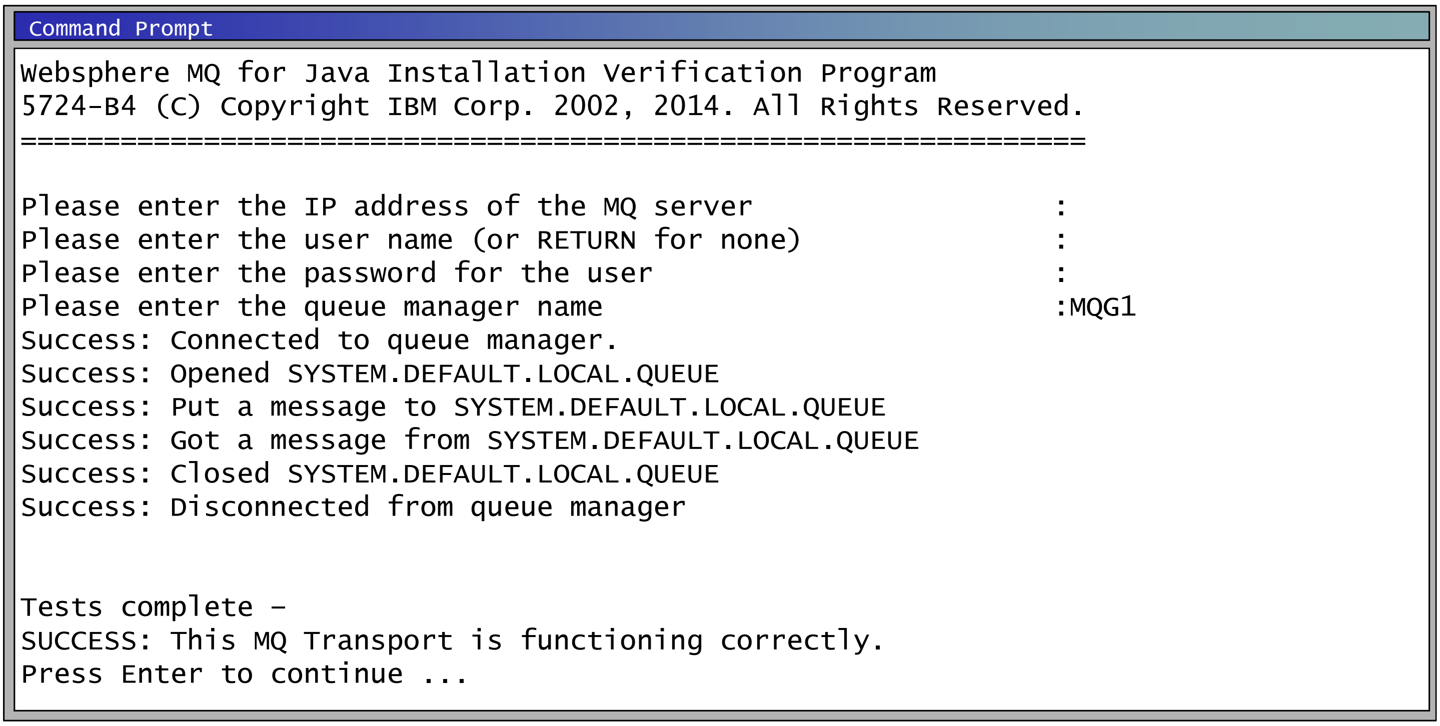
A simple example – API calls – Notes

NOTES

- And then of course, there is all the data about each API call. You can see all the options used, and in the case of some of the verbs, the entire option structure. For example here, we don't get the entire Object Descriptor (MQOD) as a blob but we get each of the relevant fields individually, but for the MQPUT both the MQMD and MQPMO structures are provided as a blob as well as some of the pertinent fields being provided individually.
- You get the Completion Code and Reason Code for each verb, and the object handle when that's applicable. You don't however, get the connection handle (more on that later).

* Example output on previous slide from MO71 Activity Trace viewer

Activity Trace always shows MQI calls

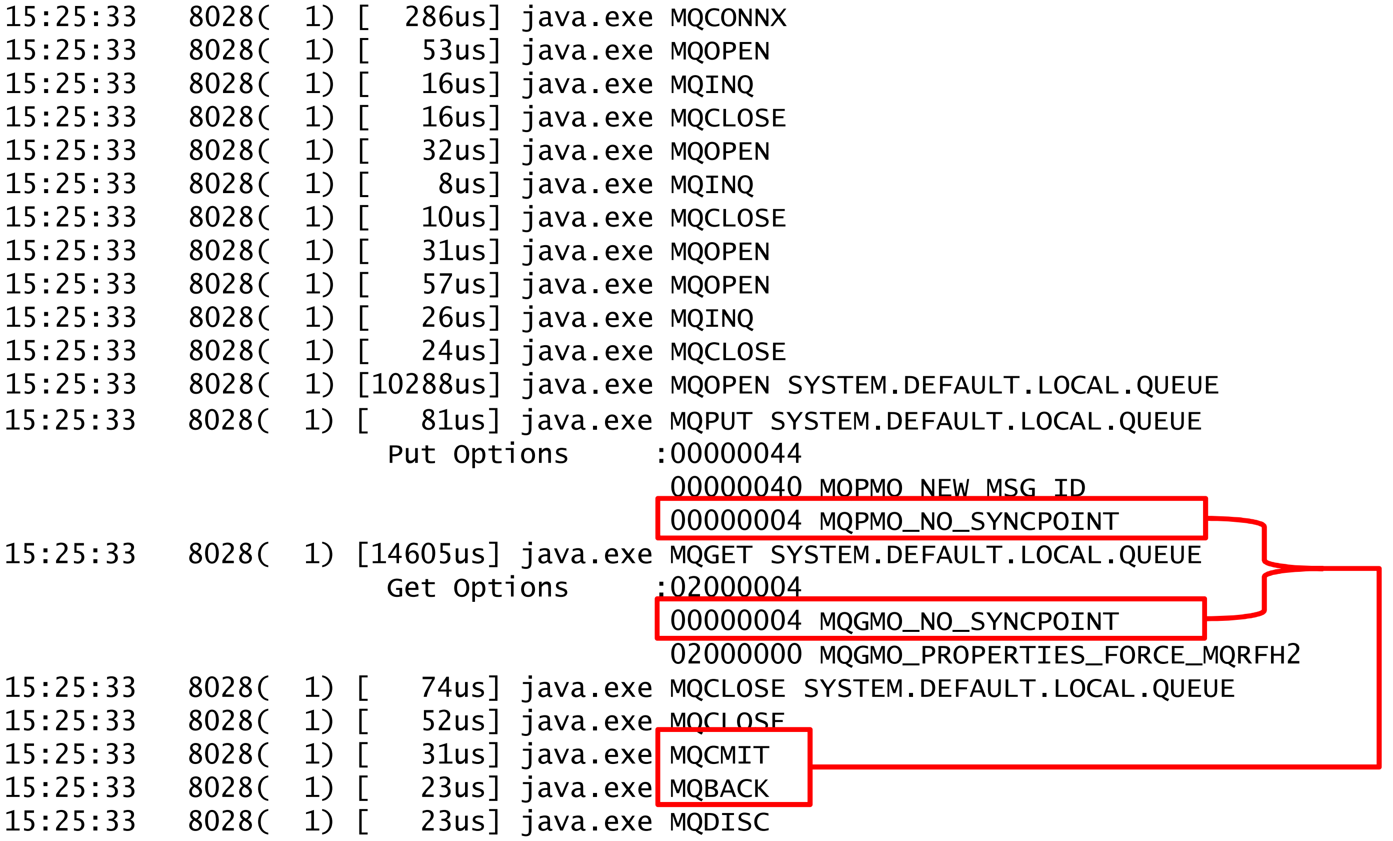


```
Websphere MQ for Java Installation Verification Program
5724-B4 (C) Copyright IBM Corp. 2002, 2014. All Rights Reserved.
================================================================

Please enter the IP address of the MQ server              :
Please enter the user name (or RETURN for none)           :
Please enter the password for the user                    :
Please enter the queue manager name                       :MQG1
Success: Connected to queue manager.
Success: Opened SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Put a message to SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Got a message from SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Closed SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Disconnected from queue manager


Tests complete -
SUCCESS: This MQ Transport is functioning correctly.
Press Enter to continue ...
```

MQIVP – shown as MQ API calls



```
15:25:33   8028(  1) [  286us] java.exe MQCONNX
15:25:33   8028(  1) [   53us] java.exe MQOPEN
15:25:33   8028(  1) [   16us] java.exe MQINQ
15:25:33   8028(  1) [   16us] java.exe MQCLOSE
15:25:33   8028(  1) [   32us] java.exe MQOPEN
15:25:33   8028(  1) [    8us] java.exe MQINQ
15:25:33   8028(  1) [   10us] java.exe MQCLOSE
15:25:33   8028(  1) [   31us] java.exe MQOPEN
15:25:33   8028(  1) [   57us] java.exe MQOPEN
15:25:33   8028(  1) [   26us] java.exe MQINQ
15:25:33   8028(  1) [   24us] java.exe MQCLOSE
15:25:33   8028(  1) [10288us] java.exe MQOPEN SYSTEM.DEFAULT.LOCAL.QUEUE
15:25:33   8028(  1) [   81us] java.exe MQPUT SYSTEM.DEFAULT.LOCAL.QUEUE
                       Put Options     :00000044
                                        00000040 MQPMO_NEW_MSG_ID
                                        00000004 MQPMO_NO_SYNCPOINT
15:25:33   8028(  1) [14605us] java.exe MQGET SYSTEM.DEFAULT.LOCAL.QUEUE
                       Get Options     :02000004
                                        00000004 MQGMO_NO_SYNCPOINT
                                        02000000 MQGMO_PROPERTIES_FORCE_MQRFH2
15:25:33   8028(  1) [   74us] java.exe MQCLOSE SYSTEM.DEFAULT.LOCAL.QUEUE
15:25:33   8028(  1) [   52us] java.exe MQCLOSE
15:25:33   8028(  1) [   31us] java.exe MQCMIT
15:25:33   8028(  1) [   23us] java.exe MQBACK
15:25:33   8028(  1) [   23us] java.exe MQDISC
```

MQIVP – zoom in on MQINQ calls

```
15:25:33   8028(  1) [  286us] java.exe MQCONNX
15:25:33   8028(  1) [   53us] java.exe MQOPEN
15:25:33   8028(  1) [   16us] java.exe MQINQ
                           Selectors    :[0]    32 MQIA_PLATFORM
                                         [1]    31 MQIA_COMMAND_LEVEL
15:25:33   8028(  1) [   16us] java.exe MQCLOSE
15:25:33   8028(  1) [   32us] java.exe MQOPEN
15:25:33   8028(  1) [    8us] java.exe MQINQ
                           Selectors    :[0]    31 MQIA_COMMAND_LEVEL
15:25:33   8028(  1) [   10us] java.exe MQCLOSE
15:25:33   8028(  1) [   31us] java.exe MQOPEN
15:25:33   8028(  1) [   57us] java.exe MQOPEN
15:25:33   8028(  1) [   26us] java.exe MQINQ
                           Selectors    :[0]    31 MQIA_COMMAND_LEVEL
                                         [1]    32 MQIA_PLATFORM
                                         [2]     2 MQIA_CODED_CHAR_SET_ID
                                         [3]  2015 MQCA_Q_MGR_NAME
                                         [4]  2032 MQCA_Q_MGR_IDENTIFIER
15:25:33   8028(  1) [   24us] java.exe MQCLOSE
```

Trace always shows MQI calls – Notes

N

O

T

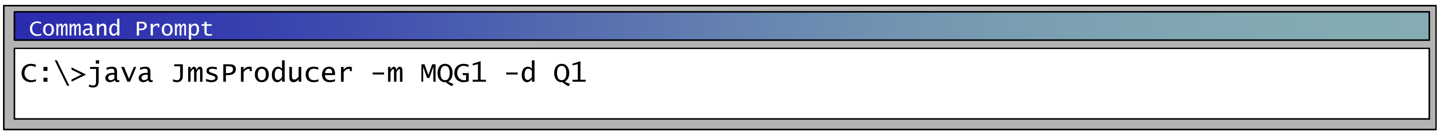
E

S

- The previous example was a very simple one. `amqsput` is a very simple MQ API sample, and you learned nothing about what it did that couldn't just as easily have been gleaned by eyeballing the code.
- Of course, you don't always have access to the code of an application, nor the logic decision points it makes. And at other times your application may not be written in the native MQ API, and so even having the code doesn't always tell you exactly what it is doing.
- Let's run another very simple example, this time of the MQIVP java sample.
- With just the straight summary of the MQ API calls you can immediately see that it's doing a little more under the covers than it said it was doing. It does an MQINQ of the queue manager – in fact several (slightly redundant) MQINQs.
- But then it continues as expected, with an MQOPEN, MQPUT, MQGET and MQCLOSE of the SYSTEM.DEFAULT.LOCAL.QUEUE.
- Oddly it finishes up with an MQCMIT and then an MQBACK call, even though it uses the *_NO_SYNCPOINT options explicitly throughout.

* Example output on previous slide from MO71 Activity Trace viewer

JmsProducer – shown as MQ API calls



```
Command Prompt
C:\>java JmsProducer -m MQG1 -d Q1
```

```
===========================================================
  Tid Time      Operation   CompCode  MQRC  HObj (ObjName)

  001 15:01:11  MQXF_CONNX  MQCC_OK   0000  -                   JMSConnection
  001 15:01:11  MQXF_OPEN   MQCC_OK   0000  2 (   )
  001 15:01:11  MQXF_INQ    MQCC_OK   0000  2 (   )
  001 15:01:11  MQXF_CLOSE  MQCC_OK   0000  2 (   )
  001 15:01:13  MQXF_DISC   MQCC_OK   0000  -
===========================================================
===========================================================
  Tid Time      Operation   CompCode  MQRC  HObj (ObjName)

  001 15:01:11  MQXF_CONNX  MQCC_OK   0000  -                   JMSSession
  001 15:01:11  MQXF_OPEN   MQCC_OK   0000  2 (Q1)
  001 15:01:12  MQXF_PUT    MQCC_OK   0000  2 (Q1)
  001 15:01:12  MQXF_PUT    MQCC_OK   0000  2 (Q1)
  001 15:01:13  MQXF_CLOSE  MQCC_OK   0000  2 (Q1)
  001 15:01:13  MQXF_DISC   MQCC_OK   0000  -
===========================================================
```

JmsProducer –MQ API calls – Notes

N

O

T

E

S

- In this sample, one of the JMS supplied samples, we can again see another interface translated into native MQ API calls in the trace.
- Those of you with a working knowledge of JMS, will be aware of the concept of a JMSConnection and a JMSSession, both of which translated into an MQ connection under the covers. We can see that in action in this example. The first MQCONNX is the JMSConnection, and the second, within which the MQPUT occurs is the JMSSession.

* Example output on previous slide from amqsact sample

What can you do with it?

- Spot applications making more MQCONN(X) calls than MQDISC calls
- Spot redundant code, like an application making multiple MQINQ calls
- Spot incorrect, or inadvisable uses of certain options on MQ API calls
  - More examples in a moment
- Spot incorrect uses of attributes in the MQMD of messages
  - More examples in a moment

What can you do with it? – Notes

N O T E S

- We've already seen a few examples of how, even just in the basic summary showing the MQ API invocations made, you can spot what might be redundancies, inefficiencies or downright errors in the application code.
- For example, you could spot an application making more MQCONN(X) calls than it makes MQDISC calls, and thus solve that MAXCHLS problem you've been seeing.
- You have all the options used on each MQ API call and can spot where there are problems in the code with the options being used. More examples to follow.
- You get to see the MQMD fields at MQPUT and MQGET time so you can see whether your application messages are conforming to your company's best practices. More examples to follow.

Option checking

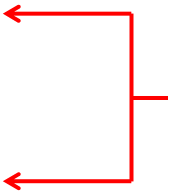
```
MQGMO
Get Options       :00002004
                   00002000 MQGMO_FAIL_IF_QUIESCING
                   00000004 MQGMO_NO_SYNCPOINT

MQGMO
Get Options       :00002001
                   00002000 MQGMO_FAIL_IF_QUIESCING
                   00000001 MQGMO_WAIT

MQOPEN(
   Hobj           :8 QUEUE(MQG1/Q1)
   Open Options   :00000410
                   00000010 MQOO_OUTPUT
                   00000400 MQOO_SET_IDENTITY_CONTEXT
MQPUT(
   Hobj           :8 QUEUE(MQG1/Q1)
   MQPMO
   Put Options    :00000004
                   00000004 MQPMO_NO_SYNCPOINT
```

No MQGMO_CONVERT

No Syncpoint option



Same Hobj ties
Together APIs

Option checking – Notes

N
O
T
E
S

- Sometimes when you use incorrect options, you will get an MQRC_OPTIONS_ERROR from MQ. However, there are some options or option combinations that, while not an error, are not necessarily advisable.
- Doing an MQGET without using the MQGMO_CONVERT option. It'll work fine until your platform coverage extends to include new codepages.
- Doing an MQGET without being explicit about syncpoint. It'll work fine until you connect the same application to a z/OS queue manager.
- You can spot problems that are as result of incorrect options being used across two different MQ APIs. For example, when an MQOPEN uses the MQOO_SET_IDENTITY_CONTEXT option but then the MQPUT doesn't then use MQPMO_SET_IDENTITY_CONTEXT. This *might* be OK if it uses the PMO option for only some MQPUTs. Of course, this could be worse, if the application does use the option at MQPUT time but then doesn't fill anything into the identity fields.
- There are no doubt plenty of other examples you can think of. See later for a few things that unfortunately cannot be checked with this trace.

* Example output on previous slide from MO71 Activity Trace viewer

Message attributes

```
Report Options: 0
Msg_type: MQMT_DATAGRAM
Expiry: -1
Format_name: 'MQSTR'
Priority: -1
Persistence: 2
Msg_id:
00000000:  414D 5120 4D51 4731 2020 2020 2020 2020  'AMQ MQG1        '
00000010:  2365 6559 2000 9B02                      '#eeY .ø.        '
Correl_id:
00000000:  0000 0000 0000 0000 0000 0000 0000 0000  '................'
00000010:  0000 0000 0000 0000                      '........        '
Reply_to_Q : ''
Reply_to_Q_Mgr: ''
Coded_char_set_id: 0
Encoding: 546
Put_date: '20170713'
Put_time: '02311968'
```

MQPRI_PRIORITY_AS_Q_DEF

MQPER_PERSISTENCE_AS_Q_DEF

Message attributes – Notes

N O T E S

- The fields from the MQMD are traced out on MQ API calls that make use of it. For an MQPUT, these are mainly the input fields from the application, so you can see applications that are not filling in a specific priority or persistence and using the defaults.
- You can check the format field is correctly supplied, and expiry matches company policy for example.
- You cannot unfortunately check for applications that are forgetting to the null out the message id for each MQPUT as the message id in the trace is the resultant message id that was assigned to the message. This is probably more useful for most cases to be able to track where that message went next.

* Example output on previous slide from amqsact sample

What can’t you do with it?

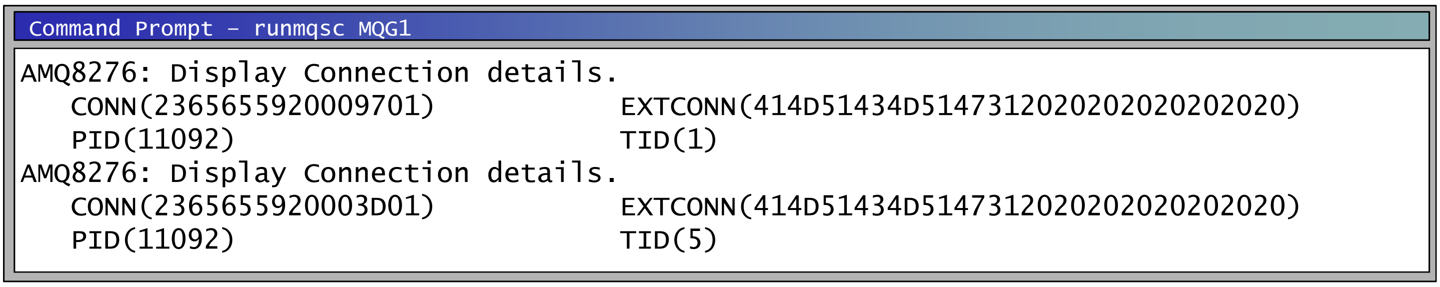
- Detect MQCONN(X) calls that result in MQRC_ALREADY_CONNECTED (2002)
- Detect calls that fail with MQRC_OPTIONS_ERROR (2046)
- Detect MQPUT calls that don’t use a syncpoint option!
- Detect MQPUT calls that forget to reset the MsgId to nulls

What can’t you do with it? - Notes

N O T E S

- An MQCONN(X) call that results in MQRC_ALREADY_CONNECTED (2002) does not get traced – doesn’t get far enough inside the queue manager to be detected and traced. In the case of a client connection, doesn’t even make it over the socket to the queue manager.
- Any API call that results in MQRC_OPTIONS_ERROR (2046) does not get traced.
- Unfortunately, if you omit to provide a syncpoint option on MQPUT, it has been filled in as MQPMO_NO_SYNCPOINT for you before the API call is traced, so you cannot detect the case where an application is not coding it.
- Unfortunately, if you are trying to detect an application which is not nulling out the Message ID before calling MQPUT, you can’t because the Message ID that was created for the message is traced rather than the input value. You could still compare it with the previous MQPUT to see if they were the same to detect the problem – just not quite so easy.

Multi-threaded applications



```
AMQ8276: Display Connection details.
   CONN(2365655920009701)              EXTCONN(414D51434D51473120202020202020200)
   PID(11092)                          TID(1)
AMQ8276: Display Connection details.
   CONN(2365655920003D01)              EXTCONN(414D51434D51473120202020202020200)
   PID(11092)                          TID(5)
```

- **MQ's notion of a thread is not the same as the O/S thread**
  - Can be seen in output from DISPLAY CONN
- **This thread ID is also what is seen in Application Activity Trace**

```
11:27:11  11092(   5) [   284us] C:\MQGem\mqmonntp.exe MQCONNX
```

- **It is the thread that the connection was made on**
- **It is not the thread that the hConn might subsequently be used on**
  - Note APAR IT22390
- **hConn is also not supplied in Application Activity Trace**

Multi-threaded applications – Notes

N

O

T

E

S

- When an application makes a connection to MQ, you can see it's details in the output from a `DISPLAY CONN` command. This shows the process id (which is an O/S construct) and the thread id (which is an MQ identifier).
- The thread id shown in this output is the thread id upon which the `MQCONN(X)` call was made.
- This `TID` which you see in `DISPLAY CONN`, is also the thread ID reported in Application Activity Trace.
- The MQ thread ID is the thread the `hConn` was created on. If you have an application that creates several connections on the main thread, say, and then starts it's various worker threads and uses the connections on those other threads, that is not the thread the connection was created on, the thread ID shown in the Application Activity Trace is not the thread that the connection is being used on, it is the thread the connection was created on. This may change in the future as a result of APAR IT22390 which has been raised to address this).
- You'll have noticed in the examples so far, that the `hConn` is also not something that is traced – `hObj`'s are, but not `hConn`.
- So that poses the question, how do I tie together the operations done on a single `hConn` if I can't use the thread or the `hConn` to do this?

Tying together operations on an hConn

- Thread ID not suitable; `hConn` not supplied; How to tie things together?
- Connection ID returned on `MQCONN(X)` trace

```
MQI Operation: 0
  Operation Id: MQXF_CONNX
  ApplicationTid: 8
  OperationDate: '2017-07-13'
  OperationTime: '15:56:38'
  ConnectionId:
  00000000:  414D 5143 4D51 4731 2020 2020 2020 2020  'AMQCMQG1        '
  00000010:  2365 6559 2000 B301                      '#eeY ...        '
  QueueManager: 'MQG1'
```

- Connection ID is the Correlation ID for ALL Application Activity Trace messages for that connection

```
MonitoringType: MQI Activity Trace
Correl_id:
00000000:  414D 5143 4D51 4731 2020 2020 2020 2020  'AMQCMQG1        '
00000010:  2365 6559 2000 B301                      '#eeY ...        '
QueueManager: 'MQG1'
```

Tying together operations – Notes

N

O

T

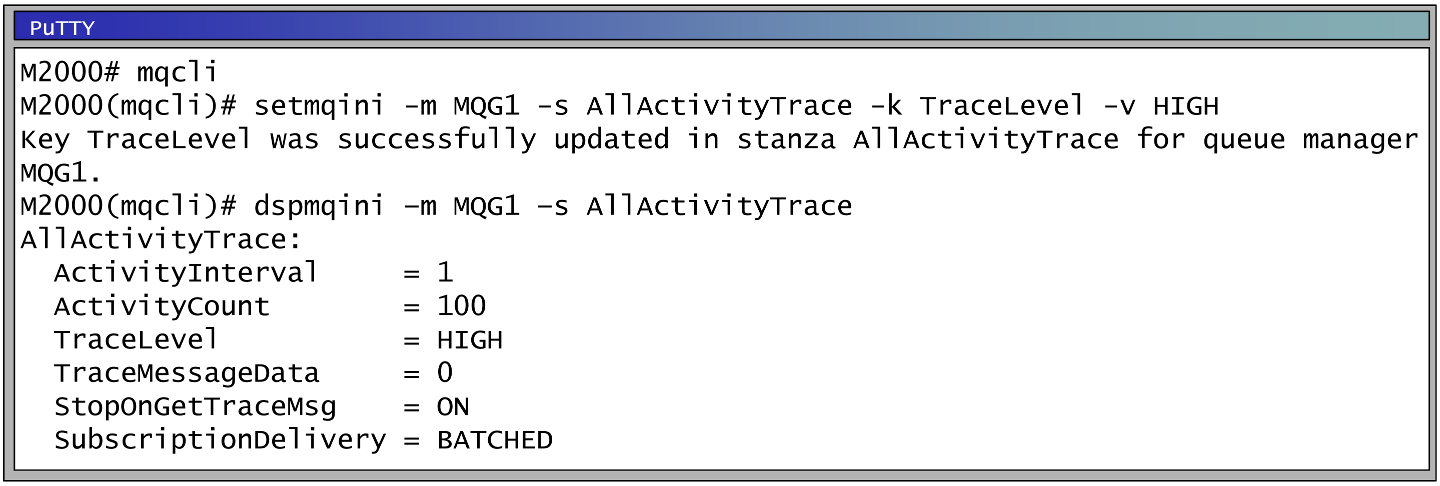
E

S

- Since the thread ID may not suffice for tying together the operations made by a single `hConn`, either because your connections were all made on the same thread, or because you have several `hConns` on a single thread, and because you are not provided with the `hConn` value in Application Activity Trace, how then can you tie together operations made on the same `hConn`?
- The answer is to use the Connection ID.
- You'll have noticed that the Connection ID is a field returned to you in the trace for an `MQCONN(X)` call.
- This value is also the Correlation ID for all Application Activity Trace messages for this connection. It is a unique value and can therefore be use to correlate all the data for a particular connection handle.

MQ Appliance - Configuration

- Can't edit files, so how to configure `mqat.ini`?
- Use `setmqini` (and `dspmqini`) to manipulate the contents of the `mqat.ini` file



```
M2000# mqcli
M2000(mqcli)# setmqini -m MQG1 -s AllActivityTrace -k TraceLevel -v HIGH
Key TraceLevel was successfully updated in stanza AllActivityTrace for queue manager
MQG1.
M2000(mqcli)# dspmqini -m MQG1 -s AllActivityTrace
AllActivityTrace:
  ActivityInterval     = 1
  ActivityCount        = 100
  TraceLevel           = HIGH
  TraceMessageData     = 0
  StopOnGetTraceMsg    = ON
  SubscriptionDelivery = BATCHED
```

- Can only edit `AllActivityTrace` stanza – not `ApplicationTrace` stanzas

MQ Appliance – Configuration – Notes

N

O

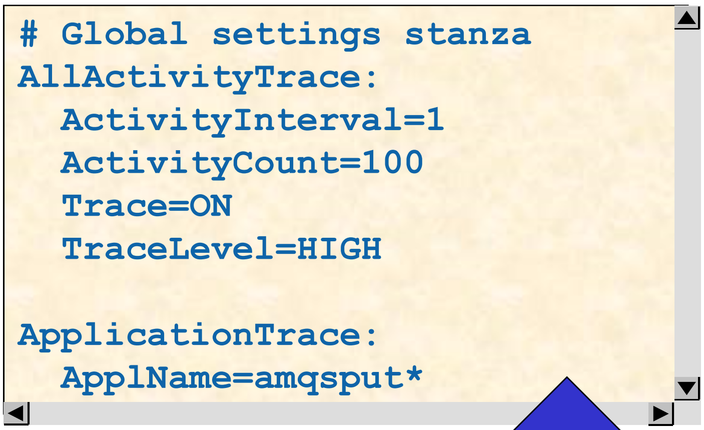
T

E

S

- The astute among you may have spotted an issue with using Application Activity Trace on the MQ Appliance. You can configure the `ACTVTRC` attribute on the queue manager, but how can you change the other settings without the ability to edit files. The `mqat.ini` file is off limits on the MQ Appliance just as with any other file.
- Well, that's not entirely true. While you cannot hand craft the `mqat.ini` file, it is possible to configure it's contents with an MQ Appliance command, `setmqini`. It's partner command `dspmqini` allows you to look at the contents.
- These MQ Appliance only commands are designed for editing ini files for MQ, and can be used to configure values in the `qm.ini` file and the `mqat.ini` file. You don't need to know which file you're directing the command to, the stanza you specify on the command tells it what you're changing.

MQ Appliance – Tracing specific Application



```
# Global settings stanza
AllActivityTrace:
  ActivityInterval=1
  ActivityCount=100
  Trace=ON
  TraceLevel=HIGH

ApplicationTrace:
  ApplName=amqsput*
```

- **Subscribe to SYSTEM topic**
- **Wildcard scheme**
  - CHAR to use ‘*’ or ‘?’
  - TOPIC to use ‘#’ or ‘+’

Subscription to:-

`$SYS/MQ/INFO/QMGR/MQG1/ActivityTrace/ApplName/amqsput*`

Using MQSO_WILDCARD_CHAR

Tracing specific Application – Notes

NOTES

- In order to tune Application Activity Trace on the MQ Appliance to only apply to a specific application, as you’ve seen by configuring the `ApplicationTrace` stanzas in `mqat.ini` on the queue manager, you have to use a different methodology.
- You have to create a subscription to a special system topic that identifies the application you want to trace.

Subscriptions to Activity Trace

- Application Name

```
$SYS/MQ/INFO/QMGR/qmgr_name/ActivityTrace/ApplName/appl_name
```

- Channel Name

```
$SYS/MQ/INFO/QMGR/qmgr_name/ActivityTrace/ChannelName/chl_name
```

- Connection ID

```
$SYS/MQ/INFO/QMGR/qmgr_name/ActivityTrace/ConnectionId/conn_id
```

- Also available in IBM MQ V9.0.0

Subscriptions to Activity Trace – Notes

N O T E S

- The example we just showed was equivalent to the settings we'd previously seen in the `mqat.ini` file. However, there is more that can be done with these subscriptions.
- The pattern of the topic string you subscribe to follows the pattern shown. You can subscribe to activity trace for an Application Name (as already seen) a Channel Name, or a Connection ID.
- This method of collecting Application Activity Trace was originally provided only on the MQ Appliance, for the reasons already discussed, but since it offers more function than just the equivalent manner of choosing a specific application to trace, it was later, in MQ V9.0.0, added to the base queue manager too.

More Subscriptions to Activity Trace

- Trace all sample Applications

```
$SYS/MQ/INFO/QMGR/qmgr_name/ActivityTrace/ApplName/amqs*
```

- Trace all Channels in Cluster SALES

```
$SYS/MQ/INFO/QMGR/qmgr_name/ActivityTrace/ChannelName/SALES.*
```

- Trace all Activity Trace

```
$SYS/MQ/INFO/QMGR/qmgr_name/ActivityTrace/#
```

More Subscriptions to Activity Trace – Notes

N

O

T

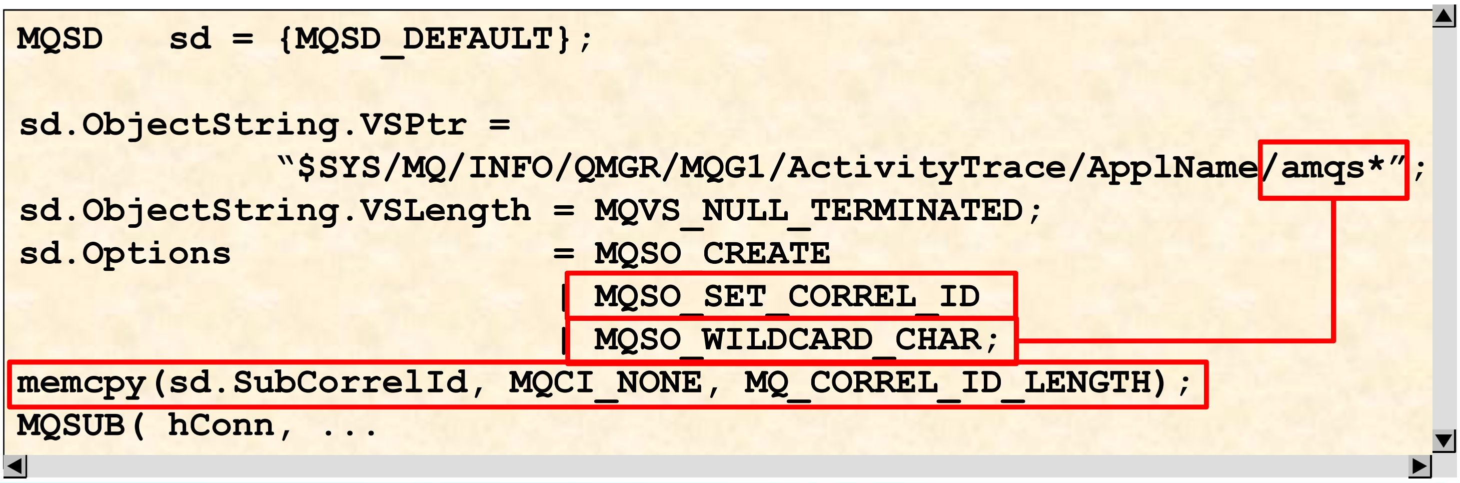
E

S

- This page shows a few more examples of the topic strings you would subscribe to, to turn on Application Activity Trace for certain things.

Creating a Subscription for Activity Trace

```
DEFINE SUB('ActivityTraceSamples')
TOPICSTR('$SYS/MQ/INFO/QMGR/MQG1/ActivityTrace/ApplName/amqs*')
DEST(SYSTEM.ADMIN.TRACE.ACTIVITY.QUEUE)
DESTCORL(0) WSCHEMA(CHAR)
```



```
MQSD   sd = {MQSD_DEFAULT};

sd.ObjectString.VSPtr =
            “$SYS/MQ/INFO/QMGR/MQG1/ActivityTrace/ApplName/amqs*”;
sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
sd.Options               = MQSO_CREATE
                         | MQSO_SET_CORREL_ID
                         | MQSO_WILDCARD_CHAR;
memcpy(sd.SubCorrelId, MQCI_NONE, MQ_CORREL_ID_LENGTH);
MQSUB( hConn, ...
```

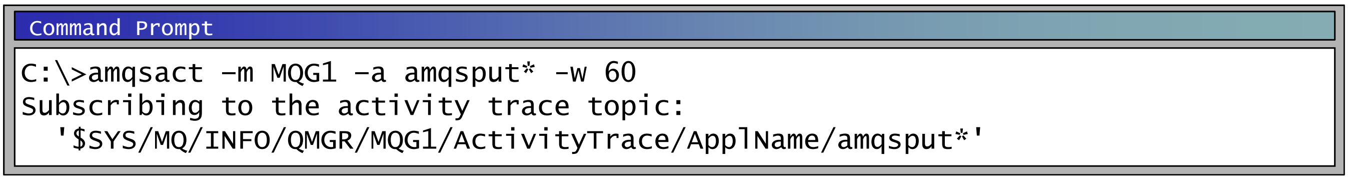
Creating a Subscription – Notes

N O T E S

- If you plan to make subscriptions in order to gather Application Activity Trace, there are a few important points to note in order for them to work correctly.
- First of all you must use the correct topic string. You’ve seen examples already on previous pages. If you have a wildcard in the topic string, you MUST use the correct wildcard scheme. For example if you have an ‘*’ or a ‘?’ wildcard, then you must use the character wildcard scheme, and if you have a ‘#’ or a ‘+’ wildcard then you muse use the topic wildcard scheme (which is the default).
- Secondly, you must ensure you are receiving the Correlation ID from the publisher, rather than every message delivered to you having your subscriber Correlation ID. In order to do this you must make a subscription with a `SubCorrelId` (`DESTCORL`) of all nulls (`MQCI_NONE`). As discussed earlier, the Correlation ID is what represents the connection and helps to tie together MQ API calls made by the same connection. If this is not correctly set then this tying together will not work.

Using amqsact to make subscriptions

- **amqsact will make subscriptions from V8.0.0.2**
  - For connecting to an MQ Appliance
- **The base queue manager will publish Activity Trace to those topics from V9.0.0**



```
C:\>amqsact -m MQG1 -a amqsput* -w 60
Subscribing to the activity trace topic:
  '$SYS/MQ/INFO/QMGR/MQG1/ActivityTrace/ApplName/amqsput*'
```

- **You have three different flags for the different resource types**
  - `-a ApplName`
  - `-c ChannelName`
  - `-i ConnectionId`
- **Beware that amqsact subscriptions don't request the publishers Correl ID**
  - Source `amqsact0.c` is provided, so you could edit and rebuild

Using amqsact to make subscriptions – Notes

N

O

T

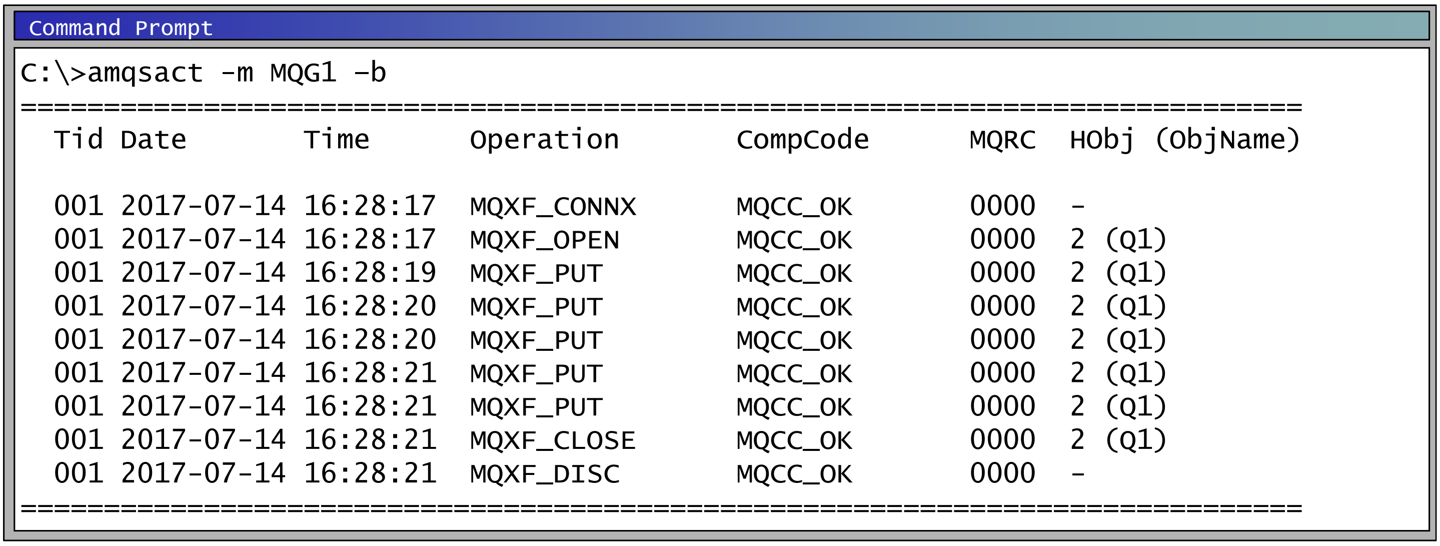
E

S

- The sample that is supplied with IBM MQ to parse and display the output from Application Activity Trace is also able to make subscriptions. This ability was added to the sample in the base product in V8.0.0 FixPack 2 for use when connecting to the MQ Appliance. It will successfully make the subscription if you use it against a V8 queue manager, but nothing will be published to that topic on a non-Appliance queue manager until it is running at V9.0.0.
- You have three different flags you can use to drive `amqsact` in this way. `–a` to subscribe to `ApplName` topics, `–c` to subscribe to `ChannelName` topics, and `–i` to subscribe to `ConnectionId` topics.
- Beware that the subscriptions which `amqsact` makes do not request the publishers Correlation ID, so you won't be able to correlate between calls made by the same connection. If you're only running single connection applications, that won't matter.
- `amqsact` is a sample with source provided, so you could edit the sample and rebuild it with the additional option shown on the previous page.

Tools to View Application Activity Trace

- **IBM Supplied sample program `amqsact`**



Command Prompt

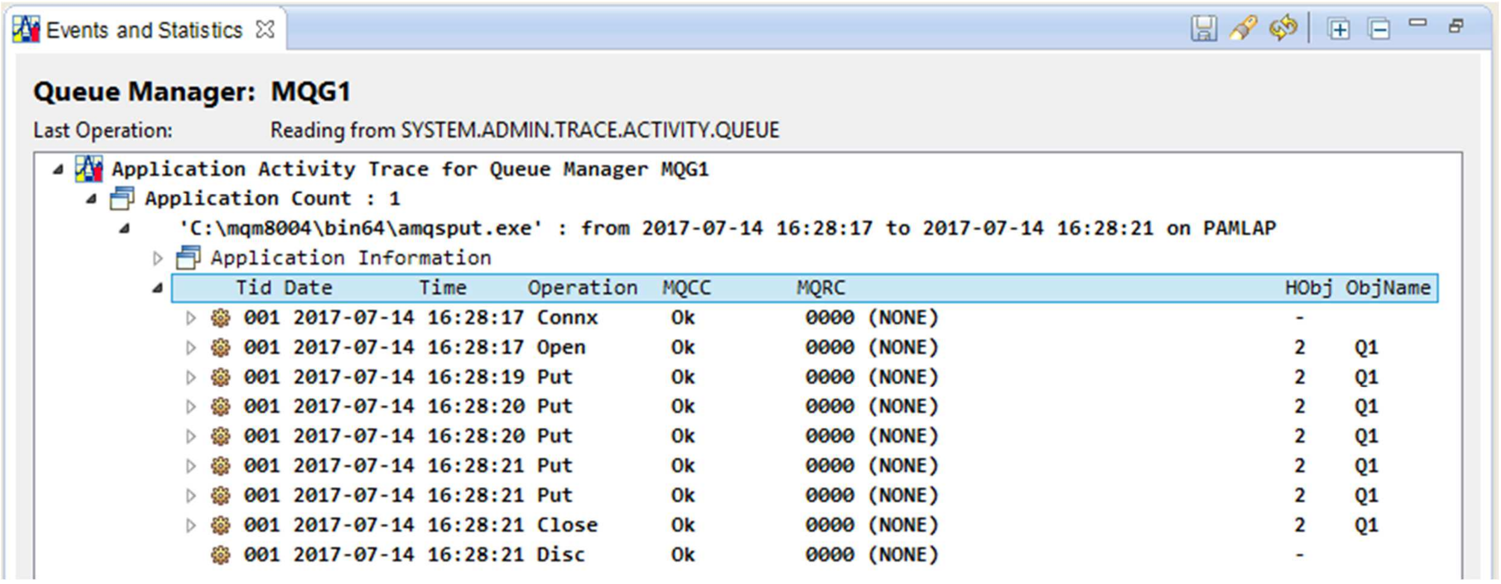
```
C:\>amqsact -m MQG1 -b
============================================================================
  Tid Date        Time      Operation        CompCode      MQRC  HObj (ObjName)

  001 2017-07-14 16:28:17  MQXF_CONNX       MQCC_OK       0000  -
  001 2017-07-14 16:28:17  MQXF_OPEN        MQCC_OK       0000  2 (Q1)
  001 2017-07-14 16:28:19  MQXF_PUT         MQCC_OK       0000  2 (Q1)
  001 2017-07-14 16:28:20  MQXF_PUT         MQCC_OK       0000  2 (Q1)
  001 2017-07-14 16:28:20  MQXF_PUT         MQCC_OK       0000  2 (Q1)
  001 2017-07-14 16:28:21  MQXF_PUT         MQCC_OK       0000  2 (Q1)
  001 2017-07-14 16:28:21  MQXF_PUT         MQCC_OK       0000  2 (Q1)
  001 2017-07-14 16:28:21  MQXF_CLOSE       MQCC_OK       0000  2 (Q1)
  001 2017-07-14 16:28:21  MQXF_DISC        MQCC_OK       0000  -
============================================================================
```

- **`amqsactz` – enhanced version of above hosted on Capitalware website**

Tools to View Application Activity Trace

- **Mark Taylor's MQ Explorer extension – Support Pac MS0P**



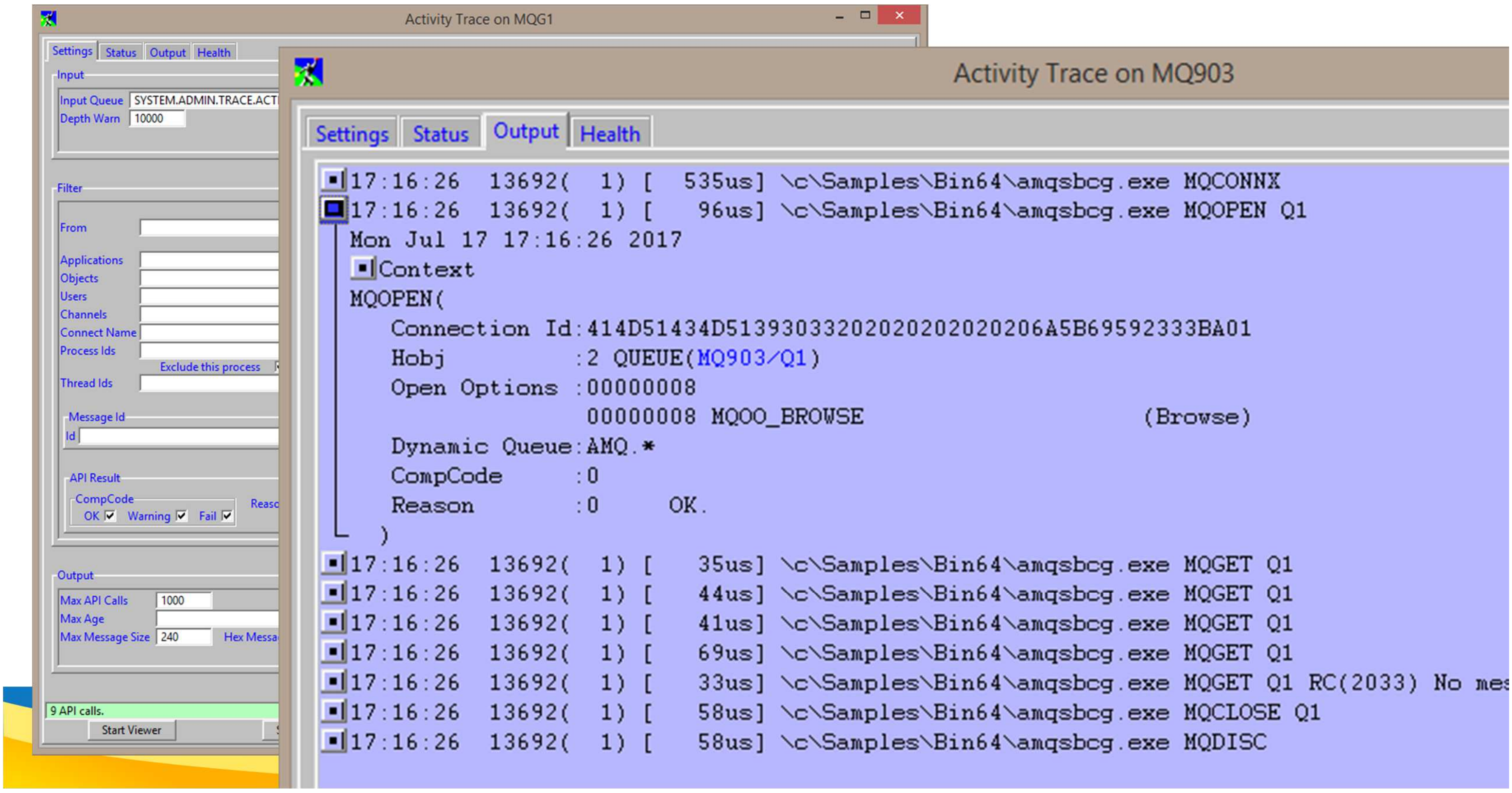
Queue Manager: MQG1

Last Operation: Reading from SYSTEM.ADMIN.TRACE.ACTIVITY.QUEUE

```
Application Activity Trace for Queue Manager MQG1
  Application Count : 1
    'C:\mqm8004\bin64\amqsput.exe' : from 2017-07-14 16:28:17 to 2017-07-14 16:28:21 on PAMLAP
      Application Information
        Tid Date       Time     Operation  MQCC      MQRC                    HObj ObjName
        001 2017-07-14 16:28:17 Connx      Ok        0000 (NONE)             -
        001 2017-07-14 16:28:17 Open       Ok        0000 (NONE)             2     Q1
        001 2017-07-14 16:28:19 Put        Ok        0000 (NONE)             2     Q1
        001 2017-07-14 16:28:20 Put        Ok        0000 (NONE)             2     Q1
        001 2017-07-14 16:28:20 Put        Ok        0000 (NONE)             2     Q1
        001 2017-07-14 16:28:21 Put        Ok        0000 (NONE)             2     Q1
        001 2017-07-14 16:28:21 Put        Ok        0000 (NONE)             2     Q1
        001 2017-07-14 16:28:21 Close      Ok        0000 (NONE)             2     Q1
        001 2017-07-14 16:28:21 Disc       Ok        0000 (NONE)             -
```

Tools to View Application Activity Trace

- **MQGem Software's MO71 – Application Activity Trace Viewer**



Activity Trace on MQ1

Settings | Status | Output | Health

Input

Input Queue: SYSTEM.ADMIN.TRACE.ACT

Depth Warn: 10000

Filter

From

Applications

Objects

Users

Channels

Connect Name

Process Ids

Exclude this process

Thread Ids

Message Id

Id

API Result

CompCode: OK ☑ Warning ☑ Fail ☑

Reaso

Output

Max API Calls: 1000

Max Age

Max Message Size: 240

Hex Messa

9 API calls.

Start Viewer

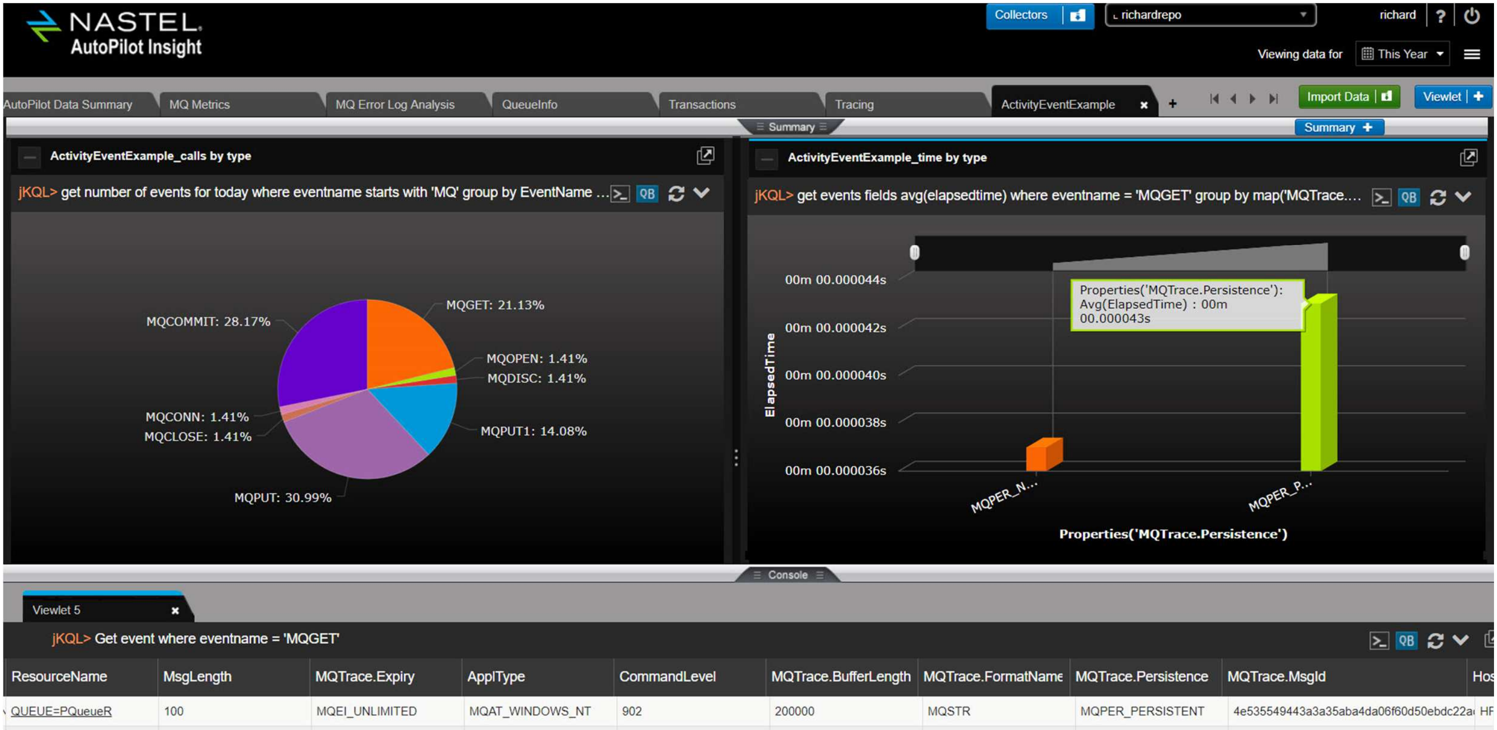
Activity Trace on MQ903

Settings | Status | Output | Health

```
17:16:26  13692(  1) [  535us] \c\Samples\Bin64\amqsbcg.exe MQCONNX
17:16:26  13692(  1) [   96us] \c\Samples\Bin64\amqsbcg.exe MQOPEN Q1
Mon Jul 17 17:16:26 2017
Context
MQOPEN(
   Connection Id:414D51434D5139303320202020202020206A5B69592333BA01
   Hobj         :2 QUEUE(MQ903/Q1)
   Open Options :00000008
                 00000008 MQOO_BROWSE                  (Browse)
   Dynamic Queue:AMQ.*
   CompCode     :0
   Reason       :0     OK.
  )
17:16:26  13692(  1) [   35us] \c\Samples\Bin64\amqsbcg.exe MQGET Q1
17:16:26  13692(  1) [   44us] \c\Samples\Bin64\amqsbcg.exe MQGET Q1
17:16:26  13692(  1) [   41us] \c\Samples\Bin64\amqsbcg.exe MQGET Q1
17:16:26  13692(  1) [   69us] \c\Samples\Bin64\amqsbcg.exe MQGET Q1
17:16:26  13692(  1) [   33us] \c\Samples\Bin64\amqsbcg.exe MQGET Q1 RC(2033) No mes
17:16:26  13692(  1) [   58us] \c\Samples\Bin64\amqsbcg.exe MQCLOSE Q1
17:16:26  13692(  1) [   58us] \c\Samples\Bin64\amqsbcg.exe MQDISC
```

Tools to View Application Activity Trace

- **Nastel's AutoPilot Insight**



ActivityEventExample_calls by type

jKQL> get number of events for today where eventname starts with 'MQ' group by EventName ...

ActivityEventExample_time by type

jKQL> get events fields avg(elapsedtime) where eventname = 'MQGET' group by map('MQTrace....

jKQL> Get event where eventname = 'MQGET'

ResourceName	MsgLength	MQTrace.Expiry	ApplType	CommandLevel	MQTrace.BufferLength	MQTrace.FormatName	MQTrace.Persistence	MQTrace.MsgId	Hos
QUEUE=PQueueR	100	MQEI_UNLIMITED	MQAT_WINDOWS_NT	902	200000	MQSTR	MQPER_PERSISTENT	4e535549443a3a35aba4da06f60d50ebdc22a	HF

Tools to View Activity Trace – Notes

N O T E S

- There are various tools that parse and display Application Activity Trace.
- Supplied with the product is a sample `amqsact` which can be used to collect and display the trace information. It can also make subscriptions as shown on an earlier slide. An updated version of it called `amqsactz` is available from http://www.capitalware.com/mq_code_c.html
- Mark Taylor's Support Pac MS0P which extends MQ Explorer, has a formatter for Activity Trace messages.
- MQGem Software's MO71 has an Activity Trace Viewer with extensive filtering capability.
- Nastel's AutoPilot also processes Activity Trace (thanks to Richard Nikula for the screenshot)

Application Activity Trace: Useful Resources

- **IBM Knowledge Center**
  - Application Activity Trace (IBM MQ)
  - https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.0.0/com.ibm.mq.mon.doc/q037520_.htm
  - Application Activity Trace (MQ Appliance)
  - https://www.ibm.com/support/knowledgecenter/SS5K6E_1.0.0/com.ibm.mqa.doc/monitoring/mo00020_.htm

- **developerWorks article by Emma Bushby**
  - "Increasing the visibility of messages using WebSphere MQ Application Activity Trace"
  - https://www.ibm.com/developerworks/websphere/library/techarticles/1306_bushby/1306_bushby.html

- **Presentation from MQTC v2.0.1.5 by Tim Zielke**
  - "MQ Problem Determination with Tracing"
  - http://www.mqtechconference.com/sessions_v2015/MQTC_v2015_Tracing.pdf
- **Blog Post about MO71 Application Activity Trace Viewer**
  - https://mqgem.wordpress.com/2017/07/13/application-activity-trace-viewer/