

How to identify IBM MQ client connections and stop them

<https://www.ibm.com/support/pages/node/616249>

Date last updated: 07-Jul-2023

Angel Rivera - rivera@us.ibm.com
IBM MQ Support

+++ Objective

To provide hints on how to identify specific connections to an IBM MQ queue manager from IBM MQ Client applications, in order to stop those connections.

IBM MQ client applications can connect to a queue manager through a transport type of "client" (TCP network), instead of "bindings" (local shared memory).

IBM MQ runmqsc, IBM MQ Explorer and SupportPac MS6B (chstat script) can be set to identify connections and stop them.

The processes described here have been tested with the following configurations:

1: An MQ JMS client, used by WebSphere Application Server, with the automatic client reconnect option enabled.

2: The amqsputc sample MQ application, with the automatic client reconnect option disabled.

3: The amqspbac high availability put sample MQ application, with the automatic client reconnect option enabled.

4: MQ Explorer, with the automatic client reconnect option disabled.

+++ Acknowledgements +++

Thanks to Morag Hughson for her suggestions for improving this document. I appreciate very much for the clarification provided in the following article.

<https://mqgem.wordpress.com/2016/06/23/maxchannels-vs-dis-qmstatus-conns/>

MaxChannels vs DIS QMSTATUS CONNS

Morag Hughson

++ Summary

If you already have some knowledge and experience in this area, here is a set of example commands for using runmqsc to identify MQ client connections and stop them:

- To display all the Applications that are connected (both bindings and client), use:

```
display qmstatus conns
```

- To display the client applications using a server-connection channel:

```
display conn(*) where(channel NE '') APPLTAG CHANNEL CONNAME CONNOPTS
```

- To stop a connection (using the desired connection number from "display conn(*)"):

```
stop conn(connectionNumberfromDisplayConn)
```

- To stop a server-connection channel and allow it to be restarted by new connections:

```
stop channel(channelName) status(INACTIVE)
```

- To stop a server-connection channel and force it to remain stopped:

```
stop channel(channelName) status(STOPPED)
```

- To identify the application that has an IBM MQ queue opened

```
display conn(*) where(objname eq Q1) type(*) all
```

- To display the version.release of the applications that are connected via a channel:

```
DISPLAY CHSTATUS(*) RAPPLTAG RVERSION CONNAME
```

- To calculate the number of running channels:

Windows: you can enter the following compound commands.

```
echo DISPLAY CHSTATUS(*) | runmqsc QMGR | find /c "AMQ8417"
```

UNIX: you can enter the following compound commands.

```
echo "DISPLAY CHSTATUS(*)" | runmqsc QMGR | grep -c 'AMQ8417'
```

++ Update from 16-Oct-2023 ++

This updated revision of this document returns the contents to the original one. Over the years, many small sections were added to this tutorial and they were cluttering the contents. Those small sections were moved to a separate document, dedicated to miscellaneous questions regarding connections and channels.

<https://www.ibm.com/support/pages/node/6984195>

Collection of articles and FAQs regarding IBM MQ Channels and Connections

++ Important distinction between a "connection" and a "channel instance" (MaxChannels)

a) Total Connections: local (non channel) and client (using a channel instance)

```
DISPLAY QMSTATUS CONNS
```

This runmqsc command shows how many connections are currently made into the queue manager.

This is a count of the number of applications (or some queue manager processes too) that have made an MQCONN/MQCONN to the queue manager.

These connections might be local or client connections (which use a channel instance) - both types contribute to the total.

To see the local connections (no channel) use:

```
DISPLAY CONN(*) ALL WHERE (CHANNEL EQ ' ')
```

To see the remote connections (using channel) use:

```
DISPLAY CONN(*) ALL WHERE (CHANNEL NE ' ')
```

The above commands might be confusing because they use "negative logic".

For example, if the attribute CHANNEL is null/empty (represented by ' ' in the WHERE clause), then there is NO channel being used, and it is a local connection.

On the other hand, if the attribute CHANNEL is NOT NULL, that is, there is a string in the attribute CHANNEL, then there is a channel being used.

b) Client Connections that use a channel instance and contribute to MaxChannels

```
DISPLAY CHSTATUS(*) CURSHCNV
```

This runmqsc command shows how many running channel instances are made to the queue manager.

If you add up all the numbers shown in CURSHCNV, then this total will be less than (or equal to) the number of channel-based connections that are listed when:

```
DISPLAY CONN(*) ALL WHERE (CHANNEL NE ' ')
```

Both queue manager channels and client channels contribute to that total.

This is the number to compare against the attribute MaxChannels of the qm.ini for the queue manager (the default is 100 and it is not explicitly included in the qm.ini).

++ The chapters are:

Chapter 1: Configuration

Chapter 2: Identifying the clients that are connected through channels from the MQ Explorer

Chapter 3: Identifying the clients that are connected through channels from runmqsc: DISPLAY CONN(*) where(channel NE ")

Chapter 4: Identifying the clients that are connected through channels, using 'chstat' from SupportPac MS6B

Chapter 5: Stopping a connection through MQ Explorer

Chapter 6: Stopping a connection through runmqsc: STOP CONN

+ Scenario 1: Stopping a connection from a client that is not using automatic client reconnect.

+ Scenario 2: Trying to stop a connection from a client that is using automatic client reconnect.

Chapter 7: Stopping a connection through runmqsc: STOP CHANNEL STATUS(STOPPED)

+ Scenario 1: Stopping a channel and specifying STATUS(INACTIVE)

+ Scenario 2: Stopping a channel and specifying STATUS(STOPPED)

```
+++++
+++ Chapter 1: Configuration
+++++
```

Even though the functions to identify and to stop the connections have not changed thru the versions, for completeness, this techdoc uses a variety of MQ versions and fix packs and multiple operating systems: MQ 9.0, 9.1, 9.2 and 9.3, Windows, Linux x86, C-Based, JMS.

Queue manager in Linux:

Name: QM93LNX
Host: reggioni1.fyre.ibm.com at port 1415
Version: 9.3.0.2

Client type 1:

MQ client in Linux x86-32 (MQ JMS 7.0.1.7 from WAS):

Host: veracruz.raleigh.ibm.com (9.27.46.236)
WAS 7.0.0.23 running with MQ JMS 7.0.1.7

Server-connection channel: SYSTEM.DEF.SVRCONN

Reconnection note: The MQ JMS client connection from WAS specifies the reconnection option.

Client type 2:

MQ client in Windows 64-bit (sample amqsputc)

Host: angelillo.raleigh.ibm.com [9.27.46.181]
Version: 7.1.0.6

Server-connection channel: SYSTEM.DEF.SVRCONN

Commands:

```
set MQSERVER=SYSTEM.DEF.SVRCONN/TCP/reggioni1.fyre.ibm.com(1415)
amqsputc Q1 QM93LNX
```

Reconnection note: The amqsphac sample, written in C, does not specify the option for automatic client reconnection.

Client type 3:

MQ client in Linux x86-64 (high availability put sample amqsphac):

Host: mosquito.raleigh.ibm.com (9.27.47.38)
Version: 8.0.0.2

Server-connection channel: TEST.SVRCONN

Commands:

```
export MQSERVER='TEST.SVRCONN/TCP/reggioni1.fyre.ibm.com(1415)'
amqsphac Q1 QM93LNX
```

Reconnection note: The amqsphac sample, written in C, specifies the following option for the connection handle:

```
cno.Options = MQCNO_RECONNECT; /* reconnectable connection */
```

Client type 4:

MQ Explorer 9.3.2 in Windows:

Host: angelillo.raleigh.ibm.com [9.27.46.181]

Version: 9.3.2.0

Server-connection channel: SYSTEM.ADMIN.SVRCONN

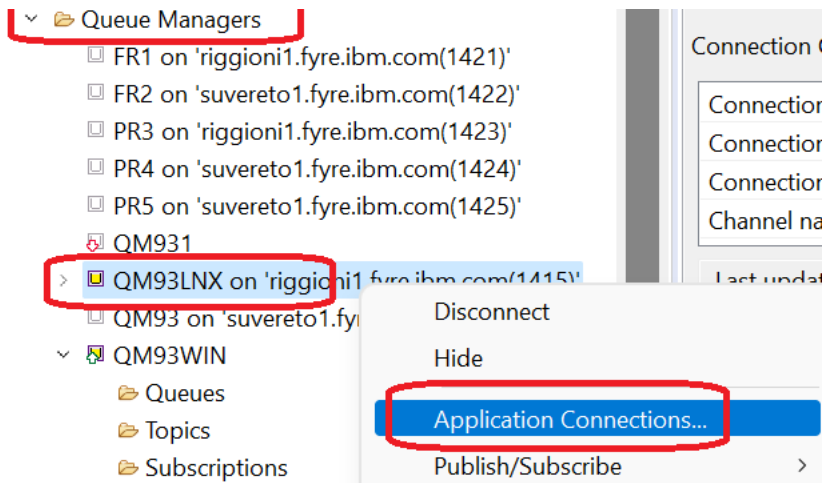
Reconnection note: When the MQ Explorer connects to a remote queue manager, does not use the reconnection option.

+++++
+++ Chapter 2: Identifying the clients that are connected through channels from the MQ Explorer
+++++

Start the MQ Explorer.

From the left panel, select the desired queue manager and right click to show the context menu.

Select "Application Connections...".



You will see a new window that has 2 panels.
The top panel shows all the connections.

App name	App type	App description	Channel name	Conn name
amqzmuf0	Queue manager	IBM MQ Distribu...		
amqzmur0	Queue manager	IBM MQ Delayed...		
amqzmur0	Queue manager	IBM MQ Deferre...		
MQ Explorer 9.3.2	Queue manager	IBM MQ Explorer	SYSTEM.ADMI...	9.163.34.18
runmqchi	Channel initiator	IBM MQ Channel...		
runmqchi	Channel initiator	IBM MQ Channel...		
runmqchl	Queue manager	IBM MQ Channel	QM93LNX.QM...	9.163.34.18(1414)
runmqsc	Queue manager	IBM MQ Script C...		

Scheme: Standard for Application Connections - Distributed

Last updated: 15:57:25 (30 items)

Notice that there is a total of 30 connections (which includes both local bindings and client connections that use channels).

This total can be found via runmqsc as follows:
To display all the connections (both bindings and client), use:

```
display qmstatus conns
AMQ8705I: Display Queue Manager Status Details.
QMNAME(QM93LNX)           TYPE(QMGR)
STATUS(RUNNING)           CONNS(30)
```


Because the contents of the above screen capture is a bit hard to read, the plain text for the combined 1st column (App name), 8th column (Channel name) and 9th column (Conn name) is shown below.

A new column called "Index" is added for the convenience of being able to talk later about each entry, and referring to the index will facilitate the discussion.

Also, underneath each entry the Connection Options (column number 7) will be included:

Index	App name	Channel name	Conn name
1	IBM MQ client for Java	SYSTEM.DEF.SVRCONN	9.27.46.236
	Connection Options: Shared, Share block, Reconnect		
2	amqsputc.exe	SYSTEM.DEF.SVRCONN	9.27.46.181
	Connection Options: Shared, Share block		
3	amqsphac	TEST.SVRCONN	9.27.47.38
	Connection Options: Shared, Share block, Reconnect		
4	MQ Explorer 9.3.2	SYSTEM.ADMIN.SVRCONN	9.27.46.181
	Connection Options: Shared, Share block		

The following has more explanation for each entry:

1: This connection is from the MQ Client from WAS, which is using the MQ classes for JMS running in host "veracruz.raleigh.ibm.com" with IP address 9.27.46.236. It is a host with Linux x86

The MQ client for Java is using the reconnect option.

The channel is SYSTEM.DEF.SVRCONN

2: It is from the MQ Client from Windows sample "amqsputc.exe", which is running in host "angelillo.raleigh.ibm.com" with IP 9.27.46.181. It is a host with Windows.

It is not using the reconnect option.

The channel is SYSTEM.DEF.SVRCONN

3: It is from the MQ Client high availability sample "amqsphac", which is running in host "mosquito.raleigh.ibm.com" with IP 9.27.47.38. It is a host with Linux x86-64:

It is using the reconnect option.

The channel is TEST.SVRCONN

4: It is from the MQ Explorer which is running in host "angelillo.raleigh.ibm.com" with IP 9.27.46.181. It is a host with Windows.

It is not using the reconnect option.

The channel is SYSTEM.ADMIN.SVRCONN

```

+++++
+++ Chapter 3: Identifying the clients that are connected through channels from runmqsc:
DISPLAY CONN(*) where(channel NE "")
+++++

```

You can use runmqsc to list all the connections that are connected through a channel.

The following general command will be used:

```
display conn(*)
```

This general command will show both the clients that are connected through bindings (local) and through client mode (network, using a server-connection channel).

However, to narrow down the list of connections that are connected through a channel, then we will use the filter 'where' and specify those entries that have a value in the attribute 'channel name'; that is, where the channel name is not null.

If the value for the attribute 'channel name' is null, then it means that it is a connection using bindings mode and we are not interested on them in this techdoc.

This type of 'where clause' is a bit tricky because the filter in runmqsc does not have the SQL equivalent of "where attribute is not null" and thus you have to use the following WHERE expression:

```
ChannelName Not Equals to singleQuote singleQuote
```

Which translates into:

```
where(channel NE '')
```

Note:

For completeness, the equivalent for "where attribute is null" is:

```
where(channel EQ '')
```

Let's run the desired display conn(*) command with the filter, showing only certain attributes, in order to get a short output.

```
$ runmqsc QM93LNX
```

```
display conn(*) where(channel NE "") APPLTAG CHANNEL CONNAME CONNOPTS
```

Note that for readability, I am introducing a separating line between entries.

```
AMQ8276: Display Connection details.
```

```
CONN(B097365514220020)
```

```
EXTCONN(414D5143514D5F373520202020202020)
```

```
TYPE(CONN)
```

```
APPLTAG(MQ Client for Java) CHANNEL(SYSTEM.DEF.SVRCONN)
```

```
CONNAME(9.27.46.236)
```

```
CONNOPTS(MQCNO_HANDLE_SHARE_BLOCK,MQCNO_SHARED_BINDING,MQCNO_RECONNECT)
```

.
AMQ8276: Display Connection details.
CONN(B097365501810020)
EXTCONN(414D5143514D5F373520202020202020)
TYPE(CONN)
APPLTAG(MQ_1\bin\amqsputc.exe) CHANNEL(SYSTEM.DEF.SVRCONN)
CONNNAME(9.27.46.181)
CONNOPTS(MQCNO_HANDLE_SHARE_BLOCK,MQCNO_SHARED_BINDING)

.
AMQ8276: Display Connection details.
CONN(B097365503800020)
EXTCONN(414D5143514D5F373520202020202020)
TYPE(CONN)
APPLTAG(amqsphac) CHANNEL(TEST.SVRCONN)
CONNNAME(9.27.47.38)
CONNOPTS(MQCNO_HANDLE_SHARE_BLOCK,MQCNO_SHARED_BINDING,MQCNO_RECONNECT)

.
AMQ8276: Display Connection details.
CONN(B0973655027F0020)
EXTCONN(414D5143514D5F373520202020202020)
TYPE(CONN)
APPLTAG(MQ Explorer 9.3.2) CHANNEL(SYSTEM.ADMIN.SVRCONN)
CONNNAME(9.27.46.181)
CONNOPTS(MQCNO_HANDLE_SHARE_BLOCK,MQCNO_SHARED_BINDING)

For comparison with the shorter output, let's show all the attributes for only one of the above connections, in that way, you can see most of the attributes.

display CONN(B097365501810020)
AMQ8276: Display Connection details.
CONN(B097365501810020)
EXTCONN(414D5143514D5F373520202020202020)
TYPE(CONN)
PID(20741) TID(35)
APPLDESC(IBM MQ Channel)
APPLTAG(MQ_1\bin\amqsputc.exe)
APPLTYPE(SYSTEM) ASTATE(NONE)
CHANNEL(SYSTEM.DEF.SVRCONN) CONNNAME(9.27.46.181)
CONNOPTS(MQCNO_HANDLE_SHARE_BLOCK,MQCNO_SHARED_BINDING)
USERID(rivera) UOWLOG()
UOWSTDA() UOWSTTI()
UOWLOGDA() UOWLOGTI()
URTYPE(QMGR)
EXTURID(XA_FORMATID[] XA_GTRID[] XA_BQUAL[])
QMURID(0.0) UOWSTATE(NONE)

To show even more data, such as which is the queue to which the application is connected, you can add: type(all)

In this example, the application is using the queue Q1:

OBJNAME(Q1) OBJTYPE(Queue)

display CONN(B097365501810020) type(all)

AMQ8276: Display Connection details.

CONN(B097365501810020)

EXTCONN(414D5143514D5F3735202020202020)

TYPE(*)

PID(20741) TID(35)

APPLDESC(MQ Channel)

APPLTAG(MQ_1\bin\amqsputc.exe)

APPLTYPE(SYSTEM) ASTATE(NONE)

CHANNEL(SYSTEM.DEF.SVRCONN) CONNAME(9.27.46.181)

CONNOPTS(MQCNO_HANDLE_SHARE_BLOCK,MQCNO_SHARED_BINDING)

USERID(rivera) UOWLOG()

UOWSTDA() UOWSTTI()

UOWLOGDA() UOWLOGTI()

URTYPE(QMGR)

EXTURID(XA_FORMATID[] XA_GTRID[] XA_BQUAL[])

QMURID(0.0) UOWSTATE(NONE)

OBJNAME(Q1) OBJTYPE(Queue)

ASTATE(NONE) HSTATE(INACTIVE)

OPENOPTS(MQOO_OUTPUT,MQOO_FAIL_IF QUIESCING)

READA(NO)

NOTES:

The following attributes from the output of DISPLAY CONN are not about the channel itself, but about the client application:

APPLDESC()

APPLTAG(MQ Client for Java)

```

+++++
+++ Chapter 4: Identifying the clients that are connected through channels, using 'chstat'
from SupportPac MS6B
+++++

```

Another way to identify the client connections is to use the following SupportPac which provides a korn shell script (called 'chstat') to list and/or kill all connections to a queue manager by queue, channel, or IP address.

The script can be downloaded from:
<https://www.ibm.com/support/pages/node/574725>
 MS6B: WebSphere MQ Connection Management Utility

For this technical document, the script was downloaded in to the following directory of the Linux host that has the queue manager:
 /downloads/mq/ms6b

Usage note: You can use the "-example" option to have the syntax for performing most of the functions provided by the script: `chstat -example`

The following command shows the connected applications:

```
$ chstat -mQM93LNX -channel
```

Total Connections	Channel Name	Connections By IP	Application Tag
21	BINDINGS_CONN	7 LOCALHOST	amqfcxba
		3 LOCALHOST	amqfqpub
		1 LOCALHOST	amqpcsea
		1 LOCALHOST	amqrrmfa
		1 LOCALHOST	amqzdmaa
		1 LOCALHOST	amqzfuma
		5 LOCALHOST	amqzmuf0
		1 LOCALHOST	runmqchi
		1 LOCALHOST	runmqsc
1	SYSTEM.ADMIN.SVRCONN	1 9.27.46.181	MQExplorer9.3.2
2	SYSTEM.DEF.SVRCONN	1 9.27.46.181	MQ_1/bin/amqsputc.exe
		1 9.27.46.236	MQClientforJava
1	TEST.SVRCONN	1 9.27.47.38	amqsphac

Total Active Channels: 4
 Total Active Connections: 25
 Total Unique Clients: 4

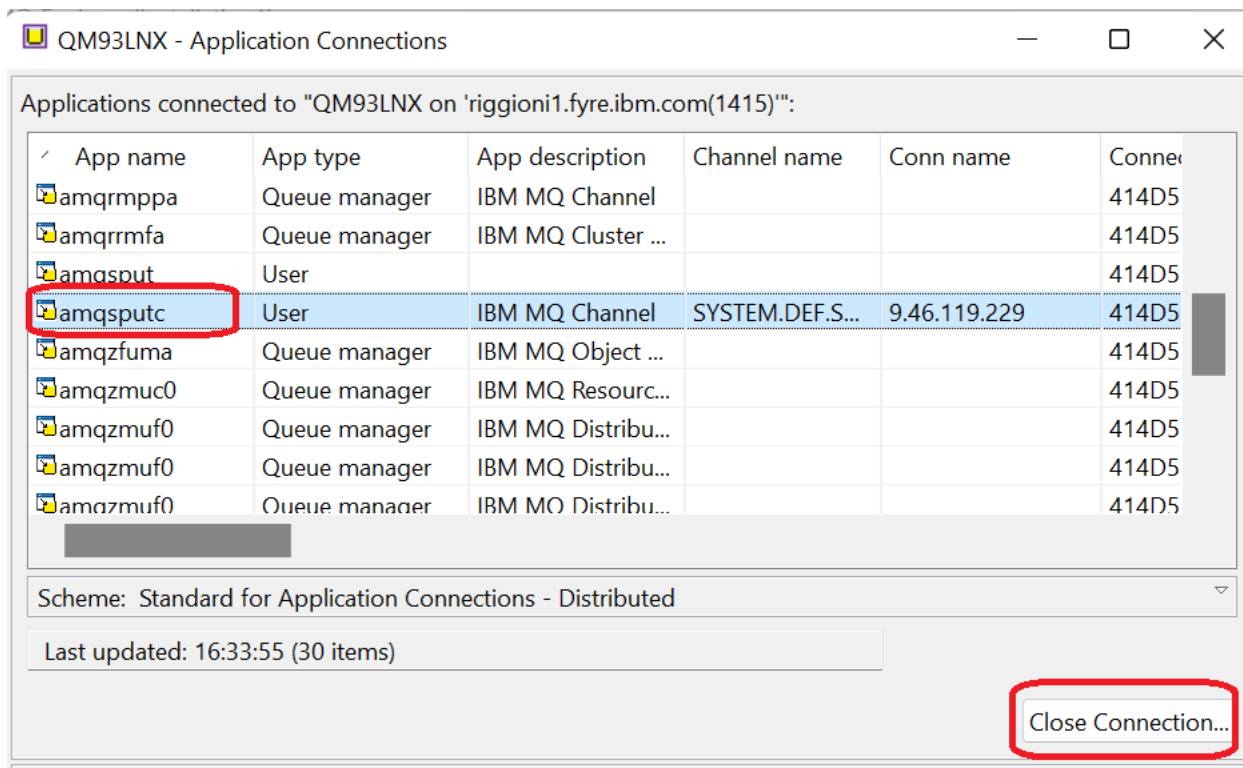
+++++
+++ Chapter 5: Stopping a connection through MQ Explorer
+++++

Let's terminate the connection from the amqsputc.exe in Windows, which currently looks like this:

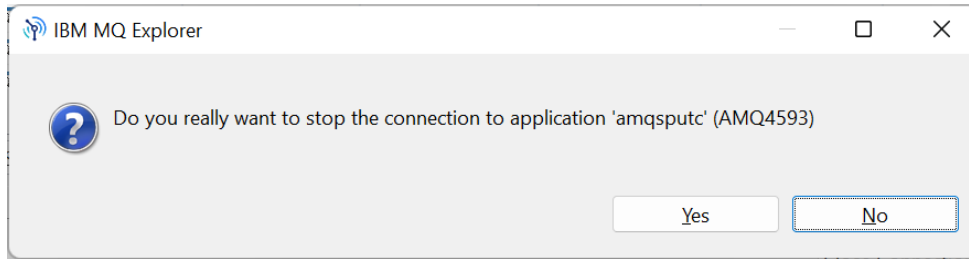
```
C:\> amqsputc Q1 QM93LNX  
Sample AMQSPUT0 start  
target queue is Q1  
message-1
```

Notice that one message has been placed in the queue Q1: "message-1"

From MQ Explorer, select the connection for amqsputc.exe and then click on the button "Close Connection"



You will get a dialog to confirm the stopping of the connection:



Click Yes.

The connection is going to be stopped (no longer active) and it will not appear anymore in this window from the MQ Explorer.

When looking at the command prompt where the sample is running, it seems that nothing happened:

```
C:\>amqsputc Q1 QM93LNX
Sample AMQSPUT0 start
target queue is Q1
message-1
```

However, if you try to enter another message and press enter at the amqsputc.exe prompt, then you will get the rc 2009 indicating a broken.

```
C:\> amqsputc Q1 QM93LNX
Sample AMQSPUT0 start
target queue is Q1
message-1
message-2
MQCLOSE ended with reason code 2009
Sample AMQSPUT0 end
```

```
C:\>mqrc 2009
      2009 0x000007d9 MQRC_CONNECTION_BROKEN
```

Look now at the bottom of the error log for the queue manager, and you will see an error entry that corresponds to the stopping of the connection:

```
03/23/2023 04:38:14 PM - Process(20741.5) User(rivera) Program(amqrmppa)
      Host(veracruz) Installation(Installation2)
      VRMF(9.3.0.2) QMgr(QM93LNX)
AMQ9546E: Error return code received.
EXPLANATION:
The program has ended because return code 8409612 was returned from function
```



```
+++++
+++ Chapter 6: Stopping a connection through runmqsc: STOP CONN
+++++
```

+ Scenario 1: Stopping a connection from a client that is not using automatic client reconnect.

From Chapter 3, the output from runmqsc for DISPLAY CONN(*) that shows the entry for amqsputc (which does not use automatic client reconnect) is the following:

```
AMQ8276I: Display Connection details.
CONN(B097365501810020)
EXTCONN(414D5143514D5F3735202020202020)
TYPE(CONN)
APPLTAG(MQ_1\bin\amqsputc.exe) CHANNEL(SYSTEM.DEF.SVRCONN)
CONNNAME(9.27.46.181)
CONNOPTS(MQCNO_HANDLE_SHARE_BLOCK,MQCNO_SHARED_BINDING)
```

To stop that connection, issue the following runmqsc, by copying the connection value from the above output:

```
STOP CONN(B097365501810020)
AMQ8457I: IBM MQ connection stopped.
```

If you issue again the DISPLAY CONN(*) command, you will see that the number of connections was decreased from 4 to 3, because the connection from amqsputc is no longer there.

You can wait around a minute and issue the DISPLAY CONN(*) again. You will notice that the count remains the same and that there is no entry for amqsputc.

This means that the sample amqsputc has not tried to reconnect, and effectively, its connection was terminated. Good!

+ Scenario 2: Trying to stop a connection from a client that is using automatic client reconnect.

From Chapter 3, the output from runmqsc for DISPLAY CONN(*) that shows the entry for amqsphac (which uses automatic client reconnect) is the following:

```
AMQ8276I: Display Connection details.  
CONN(B097365503800020)  
EXTCONN(414D5143514D5F373520202020202020)  
TYPE(CONN)  
APPLTAG(amqsphac)                CHANNEL(TEST.SVRCONN)  
CONNNAME(9.27.47.38)  
CONNOPTS(MQCNO_HANDLE_SHARE_BLOCK,MQCNO_SHARED_BINDING,MQCNO_RECONNECT)
```

Currently the amqsphac application is happily putting messages into the queue:

```
message <Message 974>  
message <Message 975>
```

Now let's try to stop the connection using the method in the Scenario 1:

```
STOP CONN(B097365503800020)  
AMQ8457I: IBM MQ connection stopped.
```

And if you quickly issue the DISPLAY CONN(*) again, you will notice that the output will not show the entry for the connection CONN(B097365503800020) and you may think that the connection from amqsphac has terminated.

But if you look again at the amqsphac application you will see that because the application is using automatic client reconnection, the MQ libraries detected the broken connection and they tried to reconnect, and were able to successfully reconnect!

```
message <Message 1096>  
message <Message 1097>  
14:39:27 : EVENT : Connection Reconnecting (Delay: 192ms)  
14:39:27 : EVENT : Connection Broken  
14:39:28 : EVENT : Connection Reconnected  
message <Message 1098>  
message <Message 1099>
```

If you issue again the DISPLAY CONN(*) from runmqsc, you will see that the entry for amqsphac is back!

But upon closer observation, you notice that the value for CONN is different, because it is indeed a new connection!

AMQ82761: Display Connection details.

CONN(B097365505940020)

EXTCONN(414D5143514D5F3735202020202020)

TYPE(CONN)

APPLTAG(amqsphac)

CHANNEL(TEST.SVRCONN)

CONNNAME(9.27.47.38)

CONNOPTS(MQCNO_HANDLE_SHARE_BLOCK,MQCNO_SHARED_BINDING,MQCNO_RECONNECT)

Hum!

This means that STOP CONN will not effectively terminate the connection from a client application that is using the option for automatic client reconnect.

Now you have this question: is there a way to terminate the connection from such client application and prevent the application from trying to reconnect again?

The answer is 'yes' and it is explained in the next chapter.

```
+++++ Chapter 7: Stopping a connection through runmqsc: STOP CHANNEL STATUS(STOPPED) +++++
```

The Scenario 2 from Chapter 6 shows that a client application that uses the automatic client reconnection cannot be effectively stopped by using the runmqsc command STOP CONN

One strategy is to allocate a separate server-connection channel for those applications that use the automatic client reconnection and if necessary, the MQ administrator can stop that dedicated channel, which in turn will terminate the connection.

However, it is important to specify the proper option for STOP CONN, which is:
STATUS(STOPPED)

The rest of this document shows 2 scenarios, one with STATUS(INACTIVE) and the other with STATUS(STOPPED).

+ Scenario 1: Stopping a channel and specifying STATUS(INACTIVE)

In this document, the connection from the high availability sample amqsphac uses the dedicated server-connection channel named "TEST.SVRCONN".

I used the MQ Explorer to stop that channel:
Selected the desired server-connection channel, right-click and issued: Stop ...
Then specified for New State: "inactive"

This is the equivalent of the following in runmqsc:
stop channel(channelName) status(INACTIVE)

The running amqsphac application got the following error message:

```
message <Message 112>
14:12:21 : EVENT : Reason(2202)
MQPUT ended with reason code 2202
Sample AMQSPHAC end
```

```
mqrc 2202:
MQRC_CONNECTION_QUIESCING
```

And the following entry was logged in the error log of the queue manager:

```
03/23/2023 03:05:38 PM - Process(20741.28) User(rivera) Program(amqrmppa)
Host(veracruz) Installation(Installation2)
VRMF(9.3.0.2) QMgr(QM93LNX)
AMQ9528I: User requested channel 'TEST.SVRCONN' to be stopped.
```

EXPLANATION:

The channel is stopping because of a request by the user.

OK! It seems that we reached the objective that an MQ administrator can terminate a connection from a client application.

... but ...

But the user from the remote system may try again to run the client application, and it will succeed, re-activating automatically the channel and creating a new connection!

```
rivera@mosquito: /home/rivera
$ amqsphac Q1 QM93LNX
Sample AMQSPHAC start
target queue is Q1
message <Message 1>
message <Message 2>
```

Thus, the use of STATUS(INACTIVE) will not prevent the remote application from trying again to contact the queue manager through the dedicated server-connection channel, which will start automatically.

Question: Is there a way to stop a server-connection channel and prevent that a remote application restarts that channel?
In order words, it is possible to force a server-connection channel to remain stopped and not to start automatically?

The answer is 'yes' and it is explained in Scenario 2 below.

+ Scenario 2: Stopping a channel and specifying STATUS(STOPPED)

To ensure that the channel remains stopped and does not restart in response from a new contact from a remote client, then specify that the new status should be "STOPPED":

I used the MQ Explorer to stop a channel:

Selected the desired server-connection channel, right-click and issued: Stop ...

Then specified for New State: "stopped"

This is the equivalent of:

```
stop channel(channelName) status(STOPPED)
```

Note: The status STOPPED is preserved during the restart of the queue manager.

The running amqsphac application got the following error message 2202

```
MQRC_CONNECTION_QUIESCING
```

```
message <Message 112>
```

```
14:12:21 : EVENT : Reason(2202)
```

```
MQPUT ended with reason code 2202
```

```
Sample AMQSPHAC end
```

But if the user of the remote system tries to restart the client application will get an error

```
2537 MQRC_CHANNEL_NOT_AVAILABLE
```

```
rivera@mosquito: /opt/mqm80/inc
```

```
$ amqsphac Q1 QM93LNX
```

```
Sample AMQSPHAC start
```

```
MQCONN ended with reason code 2537
```

```
Sample AMQSPHAC end
```

Then the following entry will be added to the queue manager:

```
03/23/2023 03:10:03 PM - Process(20741.30) User(rivera) Program(amqrmppa)
```

```
Host(veracruz) Installation(Installation2)
```

```
VRMF(9.3.0.2) QMgr(QM93LNX)
```

```
AMQ9534I: Channel 'TEST.SVRCONN' is currently not enabled.
```

```
EXPLANATION:
```

```
The channel program ended because the channel is currently not enabled.
```

```
ACTION:
```

```
Issue the START CHANNEL command to re-enable the channel.
```

```
+++++
+++ Chapter 8: Stopping a connection through SupportPac MS6B
+++++
```

One way to use the chstat script from MS6B to stop a connection is to specify the -kill option. An example is shown below.

```
rivera@veracruz: /downloads/mq/ms6b
$ chstat -mQM93LNX -kill -cSYSTEM.DEF.SVRCONN -ip9.27.46.236
```

Stopping Connections on SYSTEM.DEF.SVRCONN from 9.27.46.236 | total: 2

```
[Time: 03/23/23-15:55:26] [1] of [2] Stopping CONN(B599265502300020) |
EXTCONN(414D5143514D5F3735202020202020)
```

```
[Time: 03/23/23-15:55:26] [2] of [2] Stopping CONN(B5992655132E0020) |
EXTCONN(414D5143514D5F3735202020202020)
```

```
+++ end +++
```