MQSeries maxconn v1.0.1

A Security Channel Exit For Limiting Client Connections

Tony Madden
tm@mqware.com

1. Amendments

1.1 Version 1.0.1

1) Fix terminal error when parsing state data from State queue.
2) Allow comment lines in parameter file (first byte of comment line is '#')

2. Overview

The maxconn exit is an MQSeries security exit that allows client connections to be limited according to the name of the associated ServerConnection channel. Each Server Connection channel can be allocated a maximum number of connections and the maxconn exit will ensure that this maximum is not exceeded.

Client connections to a queue manager are limited by the MaxChannels property. If the client connections originate from multiple applications, then it is a possibility that one application could "swamp" the queue manager with client connections, thus preventing any connections being made from other applications.

The maxconn exit can be employed to limit the number of connections available to each application. The steps involved to set up maxconn are as follows:

- Each application will be assigned a Server Connection channel that is specific to that application and which must be used for all client connections.
- The maxconn exit will be configured with the appropriate maximum connection values for all the application specific Server Connection channels.
- The maxconn exit is deployed on the application specific Server Connection channels.

3. maxconn State Properties

The maxconn exit uses a set of properties (called State properties) to guide its processing. The State properties consist of initialisation and runtime information, and are stored in an MQ queue (the State queue).

There are two sets of State information. "Bootstrap" State consists of the initialisation information, which is used the first time maxconn is invoked. "Volatile" State consists of runtime information, which is updated for each connection request or termination.

Each ServerConnection channel that uses maxconn must have its own State queue. Bootstrap State is stored in a "Bootstrap" message, and rVolatile State is stored in a "Volatile" message. The Bootstrap message is created by a console application called maxconnutil (described later in this manual). Each message has a specific Correlation Id that enables direct access to the required message.

The maxconn Bootstrap and Volatile messages both contain comma delimited properties in identical order. The properties are listed below in the order in which they appear in Bootstrap and Volatile messages.

If the Bootstrap and Volatile messages differ in their treatment of a given property then that is noted in the text.

1. StateDataType

Specifies whether the message is Bootstrap or Volatile. Values are:

"B" message is Bootstrap.

"V" message is Volatile.

2. MaxConnections

The maximum number of connections.

3. ExistingConnections

The number of currently connected applications using the ServerConnection channel in question. This value is incremented by one each time a maxconn exit allows a connection, and is decremented by one each time a maxconn exit is terminated (i.e. an MQXR_TERM call) following a successful connection. Note that if maxconn refuses a connection then the associated MQXR_TERM call will not result in a decremented *ExistingConnections* value.

NOTE- this property is zero in the Bootstrap message.

4. MaxInUseAttempts

The maxconn exit opens the maxconn queue for EXCLUSIVE_INPUT. It is possible that multiple contemporaneous connection attempts will result in MQOPEN calls returning MQRC_OBJECT_IN_USE. The *MaxInUseAttempts* parameter specifies how many times an instance of maxconn should try to open the queue before ending in error. The maxconn exit will wait up to 200 milliseconds between each attempt.

5. TimesAtConnectionMaxSinceLastPCFCheck

If *PcfCheckRatio* is greater than 1, then *TimesAtConnectionMaxSinceLastPCFCheck* contains the number of consecutive times the exit has refused a connection since the last time a PCF Inquiry on channel status.
If *PcfCheckRatio* is 1, then *TimesAtConnectionMaxSinceLastPCFCheck* is always zero.

NOTE- this property is zero in the Bootstrap message

6. PcfCheckRatio

If maxconn determines that the number of existing connections for the relevant channel is equal to or greater than the *MaxConnections* for that channel, then maxconn will refuse the connection.
Before refusing the connection, maxconn has the ability to issue a PCF Inquire CHSTATUS request for the channel, to verify that the actual number of existing connections agrees with the *ExistingConnections* value held by the maxconn State data.
The default value for *PcfCheckRatio* is one – i.e. maxconn will issue a PCF inquiry prior to each and every connection refusal. If this is felt to be excessive, then the *PcfCheckRatio* parameter can be used to specify the ratio of connection refusals to PCF inquiries – for example, a *PcfCheckRatio* of 5 means that a PCF inquiry will be issued once every 5 refused connections.
A *PcfCheckRatio* value of zero means that no PCF inquiries will be issued.

7. EventFlag

Specifies whether events should be produced.
Values are:

0	do not produce event message
1	write event messages to log file
3	write event messages to log file and MQ queue

8. WarnThreshold

If event reporting is enabled, then the *WarnThreshold* will raise a Warn event.
If warn events are not required then *WarnThreshold* will be zero.

9. EventState

EventState will be one of the following values:

1	OK
3	WARN
4	MAX

See Event Processing for definitions of these values.

NOTE- this property is zero in the Bootstrap message.

10. TraceLevel

Trace information can be written from the exit to the log file.
Values are:

0	no trace
1	info trace
2	debug trace

11. LogFileDirectory

The directory into which log files will be written. The log file is used for logging errors, event and trace information.

12. LogFileName

The log file name. The log file is used for logging errors, event and trace information.

13. EventQueue

The name of the queue to which maxconn event messages will be written.

4. Processing Overview

The maxconn exit uses a set of properties (called State properties) to guide its processing. The State properties consist of initialisation and runtime information, and are stored in an MQ queue (the State queue).

There are two sets of State information. "Bootstrap" State consists of the initialisation information, which is used the first time maxconn is invoked. "Volatile" State consists of runtime information, which is updated for each connection request or termination.

Each ServerConnection channel that uses maxconn must have its own State queue. Initialisation information is stored in a "Bootstrap" message, and runtime information is stored in a "Volatile" message. The Bootstrap message is created by a console application called maxconnutil (described later in this manual). Each message has a specific Correlation Id that enables direct access to the required message.

The maxconn exit processing is split into two parts – the connection request and the connection termination.

4.1 Connection Request

Request processing is done when the client application connects. The maxconn exit is called with MQXR_SEC_MSG. The main steps are:

1. Open Exclusive the State queue.
2. Destructively read the Volatile message from the State queue.
3. If no Volatile message exists, then
    - browse the Bootstrap message from the State queue.
    - PCF Inquire on the number of existing connections using the current channel (i.e. inquire on chstatus).
4. Determine whether the exiting number of connections equals or exceeds the maximum number of connections. If so, reject the current connection request.
5. Update the Volatile data and write it back to the State queue.
6. Return with either MQXCC_OK (i.e. accept the connection request) or MQXCC_SUPPRESS_FUNCTION (i.e. refuse the connection request).

4.2 Connection Termination

When a client connection is terminated, the exit is called with MQXR_TERM.

If the termination call is associated with a successful connection, then the following steps are performed:

1. Open Exclusive the State queue.
2. Destructively read the Volatile message from the State queue.
3. Update the Volatile data and write the Volatile data back to the State queue.

If the termination call is associated with an unsuccessful connection request, then no processing is required unless trace logging is enabled. In this case, the following steps are performed:

1. Open State queue for browsing.
2. Browse the Volatile message.
3. Get log file location and write trace data to it.

5. The maxconn State Queue and Messages

The maxconn State queue contains the Bootstrap and Volatile messages. Each ServerConnection channel which uses maxconn must have a separate State queue.

5.1 The State Queue

The State queue name is "MAXCONN."*ChannelName*, where *ChannelName* is the name of the ServerConnection channel in question. Normally, all State queues will be created by the maxconnutil utility, which uses a model queue called MAXCONN.MODEL.QUEUE.

5.2 Queue Access

The maxconn exit normally opens the State queue for Exclusive Input. That is, for a given channel, connection requests are single-threaded. This is to allow accurate updating of Volatile data.

5.3 Model Queue

The MAXCONN.MODEL.QUEUE should be created with a Definition Type of PERMDYN and a Default Persistence of NO (i.e. Not Persistent).

5.4 Correlation Ids

The maxconn exit retrieves Bootstrap and Volatile messages by keying on Correlation Id. The Bootstrap message has a Correlation Id of "B"*ChannelName* and the Volatile message has a Correlation Id of "V"*ChannelName*, where *ChannelName* is the name of the ServerConnection channel.

Note that the Correlation Ids are channel specific and that it is therefore feasible that all State messages could reside on the same local queue (in which case the State queue names required by the exit would be aliases which resolve to that single local queue). However, if this is done then connection requests will be single threaded across all the channels that use maxconn. This could have performance implications.

5.5 Persistence

The Bootstrap message is written to the State queue by the maxconnutil utility with a persistence property of MQPER_PERSISTENT.

The Volatile data is written to the State queue by the maxconn exit with a persistence property of MQPER_PERSISTENCE_AS_Q_DEF. It is strongly recommended that the State queue (or the model queue MAXCONN.MODEL.QUEUE) has a default Persistence of NO (i.e. Not Persistent) because this will improve performance of the exit. The Volatile data is just that – volatile. It is not meant to persist across queue manager restarts.

6. Event Processing

The maxconn exit can produce event data when threshold events occur. Event data can be produced in a text format and in a PCF format. Text event records are written to the logfile. PCF format data are written as persistent messages to the maxconn event queue.
Log events are only produced if events are enabled.
PCF events are only produced if MQ Events are enabled.

6.1 Types of Event

Four types of event are recognised by maxconn:

OK

This event is produced when the number of connections decreases to below the WarnConnection level (if defined) or below the MaxConnection level (if WarnThreshold not defined).

WARN

Produced when the number of connections using the ServerConnection channel in question equals or exceeds the WarnThreshold but are less than the MaxConnections value. If no WarnThreshold has been defined then no *WARN* events will be produced.

MAX

Produced when the number of connections equals or exceeds the MaxConnections value.

6.2 MQ PCF Event Message Format

The maxconn MQ event messages are in PCF format. The PCF parameter blocks are guaranteed to be in the following order:

MQCFH

Command=MQCMD_CHANNEL_EVENT

Reason=user defined reason codes:

140	*OK Event*
141	*Warn Event*
142	*Max Event*

The MQCFH is the PCF header block

MQCFST containing state information

Parameter=MQCACF_STRING_DATA

The String field contains state information about the event. There are three components:

1. A literal value that specifies which event has occurred.
2. A literal value that specifies the context in which the event occurred. If the event was produced during a connection request, then the literal value is “REQU”. If the event was produced during connection termination, then the literal value is “TERM”.

3. A copy of the Volatile State information (i.e. the data which is contained in the Volatile message on the State queue) after it has been updated by the current invocation of the exit.

See Event Processing for the format of the state information.

MQCFST containing qmgr name

Parameter=MQCA_Q_MGR_NAME

The String field contains the qmgr name

MQCFST containing ServerConnection channel name

Parameter=MQCACH_CHANNEL_NAME

The String field contains the ServerConnection channel name

MQCFST containing connection name

Parameter=MQCACH_CHANNEL_NAME

The String field contains the connection name.

6.3 Format of the State Information (the first MQCFST parameter)

The first MQCFST parameter block contains State Information. The information is held in the String field of the MQCFST variable. The format of the State Information is a PCFSTRINGHEADER structure followed by a PCFSTRINGVARIABLEHEADER structure followed by the Volatile State message data.

PCFSTRINGHEADER and PCFSTRINGVARIABLEHEADER are maxconn structures (defined in maxconn.h):

1. **PCFSTRINGHEADER fields**

This structure describes the event.

eventType 10 bytes

The Event Type specifies what type of event is occurring. The values are:

"EVENT=OK*bb*"	The event message signals that an OK event has occurred.
"EVENT=WARN"	The event message signals that a WARN event has occurred - the number of connections has reached or exceeded the warn threshold but is less than the MaxConnections value
"EVENT=MAX*b*"	The event message signals that a MAX event has occurred - the number of connections has equalled or exceeded the MaxConnections value.

NOTE – where necessary, the Event Type values are padded with trailing spaces (*b* signifies a space character in the text values shown above).

eventContext 4 bytes

The Event Context field specifies whether the event has been created during a connection request or a connection termination. The values are:

"REQU"	The event message was created during a connection request
"TERM"	The event message was created during a connection termination.

2. PCFSTRINGVARIABLEHEADER fields

This structure describes the properties of the variable length text string that immediately follows the PCFSTRINGVARIABLEHEADER structure. For maxconn, the text string will be the contents of the volatile State message.

varLength 3 bytes

The length of the associated text string (note – in display numeric format)

Volatile State message data variable length

The variable length text string. Contains the Volatile State message data.

6.4 Determining What Event Type Is Signalled By An MQ PCF Event Message

Each event message specifies a particular event. There are two methods of determining which maxconn event is specified. These are described in the following two sub-sections.

6.4.1 Using the MQCFH reason field

The reason code values are defined within the limits of MQRC_APPL_FIRST and MQRC_APPL_LAST. The values and meanings are:

140 The current event message specifies that an OK event has occurred.

141 The current event message specifies that a WARN event has occurred - the number of connections has reached or exceeded the warn threshold but is less than the MaxConnections value

142 The current event message specifies that a MAX event has occurred - the number of connections has equalled or exceeded the MaxConnections value.

6.4.2 Using pattern recognition

Some MQ monitor tools include pattern recognition functionality. It is thus possible to search the event message for a string value that will indicate what type of event is signalled by the message.

The eventType field of the PCFSTRINGHEADER structure can be used for such a purpose. The *eventType* field is found at offset 56 (MQCFH_STRUC_LENGTH + MQCFST_STRUC_LENGTH_FIXED) from the start of the event message data.

6.5 maxconn Events – an Example

A ServerConnection channel is using the maxconn exit. Event processing is activated. The maximum connection level is 500. The WarnThreshold is 400.

The sequence of activity is as follows:

1. Queue manager starts.
2. The number of client connections rises until it equals the WarnThresholds value of 400. A *WARN* event is produced (event state changes from *OK* to *WARN*).

3. A client disconnects and the number of client connections falls back to 399. An *OK* event is produced (event state changes from *WARN* to *OK*).
4. A client connects and the number of connections rises back to 400. A *WARN* event is produced (event state changes from *OK* to *WARN*).
5. The number of connections continues rising until it hits the MaxConnections value of 500. A *MAX* event is produced (event state changes from *WARN* to *MAX*).
6. Subsequent connection requests are refused.
7. A client disconnects. The number of connections is now 499 - less than MaxConnections but more than WarnThresholds. A *WARN* event is produced (event changes moves from *MAX* to *WARN*).
8. A hundred clients disconnect. When the number of connections moves from 400 (the WarnThreshold value) to 399 an *OK* event is produced (event state changes from *WARN* to *OK*).

7. Logging

The maxconn exit writes information to a log file. There are four types of data which can be written to the log file:

- Errors error information. Errors are always logged
- Info provides basic debug information. Optional. Set by the LogFlag parameter.
- Debug provides detailed debug information and includes the Info level of logging. Optional. Set by the LogFile parameter.
- Event provides information about maxconn events. Optional. Set by the EventFlag parameter.

7.1 Log File Location and Name

The name and location of the log file is specified by the parameters LogFileName and LogFileDirectory.
If the LogFileDirectory path includes spaces, then enclose the directory path in double quotes.
If LogFileName is not specified, then it defaults to MAXCONN.*QmgrName*.*ChannelName*.LOG.
If the LogFileDirectory is not specified then it defaults to "c:" on NT, and to the relevant home directory on Unix.

7.2 Log File Sample

Here is a sample from a log file where debug logging and event logging are active. The ServerConn channel is called VTEST.SVRCONN and the local queue manager is QM1. The WarnThreshold is set to 1 and MaxConnections is set to 2 (obviously, these are not representative values and are used merely for demonstration purposes).

Note that every log record includes the RemoteUserId and the Connection Name of the client. These two values might be useful for identification purposes.

Note that all references to "Existing" connections refer to the number of currently active connections using the ServerConn channel, *excluding* the in-flight connection request.

07/14/02 19:23:42 REQU. DEBUG. Connection Process Commencing . Volatile Data <B,2,0,20,0,1,3,1,0,3,c:\dev\cppdev\chanmax,MAXCONN.QM1.VTEST.SVRCONN.LOG,MAXCONN.EVENT.QUEUE,>. ExtractedValues: MaxConnections=2, ExistingConnections=0, TraceLevel=3, EventState=0 . QMgr=QM1. Channel= VTEST.SVRCONN. RemoteUserId=tmadden. ConnName=172.34.55.5

The literal "REQU" means that this is a log record written during a request for a connection. "DEBUG" means that this is a debug trace record. "Volatile Data" is followed by the contents of the Volatile state message. Note that, if no Volatile message exists, then maxconn would read the Bootstrap message and this trace record would then display Bootstrap data. The RemoteUserId indicates the userid of the client. The ConnName is the connection name of the client.

07/14/02 19:23:42 REQU. INFO . * Connection request allowed for connection name <169.254.46.165> using channel VTEST.SVRCONN. Existing=0. Max=2. QMgr=QM1. Channel= VTEST.SVRCONN. RemoteUserId=tmadden. ConnName=172.34.55.5**

"INFO" means that this is an informational trace record. "Existing=0" means that the existing number of connections (excluding the current connection request) is zero. "Max=2" means that the maximum number of connections is 2.

07/14/02 19:23:42 REQU. EVENT. EVENT=WARN. Number of connections (1) equals or exceeds warn threshold (1). QMgr=QM1. Channel= VTEST.SVRCONN. RemoteUserId=tmadden. ConnName=172.34.55.5

"EVENT" means that this is an event record. "EVENT=WARN" means that the warn threshold will be reached when the current connection is completed.

07/14/02 19:23:42 REQU. DEBUG. Rewrite volatile data <V,2,1,20,0,1,3,1,8,3,c:\dev\cppdev\chanmax,MAXCONN.QM1.VTEST.SVRCONN.LOG,MAXCONN.EVENT.QUEUE,>. QMgr=QM1. Channel= VTEST.SVRCONN. RemoteUserId=tmadden. ConnName=172.34.55.5

This record shows the volatile data prior to it being re-written to the State queue during a connection request.

07/14/02 19:23:43 TERM. EVENT. EVENT=OK . Number of connections (0) is OK. QMgr=QM1. Channel= VTEST.SVRCONN. RemoteUserId=tmadden. ConnName=172.34.55.5

The literal "TERM" means that this is a log record written during the termination of a connection. "EVENT" means that this is an event record. This record indicates that, after the current connection termination, the number of connections will be zero and so the event warning threshold is no longer exceeded.

07/14/02 21:30:43. TERM. ERROR. Error on MQPUT1 for queue MAXCONN.EVENT.QUEUE. Reason=2053. QMgr=QM1. Channel= VTEST.SVRCONN. RemoteUserId=tmadden ConnName=169.254.46.165. maxconn v1.0

This is a sample error message. The text describes the error (in this case, the fact that the event queue is full). Note that the version of maxconn is appended to error messages.

07/14/02 19:23:43 TERM. DEBUG. Rewrite volatile data <V,2,0,20,0,1,3,1,1,3,c:\dev\cppdev\chanmax,MAXCONN.QM1.VTEST.SVRCONN.LOG,MAXCONN.EVENT.QUEUE,>. QMgr=QM1. Channel= VTEST.SVRCONN. RemoteUserId=tmadden. ConnName=172.34.55.5

This record shows the volatile data prior to it being re-written to the State queue during a connection termination.

07/14/02 19:23:43 TERM. INFO. * Connection terminated for connection name <169.254.46.165> using channel VTEST.SVRCONN. Existing=0. Max=2. QMgr=QM1. Channel= VTEST.SVRCONN. RemoteUserId=tmadden. ConnName=172.34.55.5**

“INFO” means that this is an informational trace record. It specifies that a connection has been terminated.
“Existing=0” means that the number of existing connections (excluding the current connection) is zero.
“Max=2” means that the maximum number of connections is 2.

8. Performance Considerations

maxconn is designed to be as efficient as possible (all suggestions for improvements are most welcome!). Its use will obviously have some negative impact on the speed of client connections. It is a user decision whether the benefits of throttling the client connections exceed the costs.

The overhead of using maxconn will be minimal for long-lived client applications which client connect at startup and do not disconnect until shutdown.

If a client application is designed which involves a high number of short-lived Connect-Process-Disconnect type transactions, with no connection pooling, then the overhead of maxconn might be significant.

The other situation in which maxconn might have a significant performance impact is during a mass failover from one server to another.

8.1 MQSeries I/O

maxconn state data is held in the Volatile message on the State queue. This message is assumed to be non-persistent (it takes its persistence from the default persistence of the State queue).

All I/O is outside of syncpoint.

8.2 Synchronization

During a connection request and a connection termination (after a successful connection request only) maxconn opens the State queue for Exclusive Input and keeps the queue open until it has read the Volatile message, processed it and re-written the Volatile message. Therefore, concurrent connection requests and/or terminations using the same ServerConnection channel will be effectively single threaded by the exit.

8.3 Client Failover

If the Client architecture involves a primary Server and a secondary Server, then a mass failover to the secondary will be impacted by the use of maxconn. However, such failovers are rare, so the impact might be considered acceptable.

If the client architecture involves a primary Server and an overflow Server defined in a Channel Table, then additional factors come into play. The overflow Server is used when the primary Server has reached maximum connections. That is, the primary Server (i.e. the maxconn exit) has to refuse a connection prior to the re-connect to the overflow Server. If a significant number of client connections overflow then the value of the *PcfCheckRatio* property should be re-evaluated. The default value of 1 means that, prior to each refused connection, a PCF Inquiry on CHSTATUS is performed. This is excessive for the overflow situation described here, which involves significant overflowing.

8.4 Logging

Obviously, logging will have an effect on performance. Log data consists of event, trace and error information. The exact impact depends on the efficiency of the file system and the type of logging (trace logging will have more impact than merely logging errors, for example).

9. maxconnutil utility

The maxconnutil console application has two functions:

1. It is used to load the "Bootstrap" (initialisation) information into the State queues used by the exit.
2. It is used to display the state information on a State Queue.

Each channel that uses maxconn must have its own State queue. Maxconnutil will create these queues if they do not exist.

9.1 Pre-requisites

If maxconnutil is to create State queues, then a model queue called MAXCONN.MODEL.QUEUE must be defined with a definition type of PERMDYN. It is recommended that the default persistence property of the model queue is NO (i.e. Not Persistent).

9.2 Maxconnutil Arguments

The arguments for maxconnutil are specified in main arguments and in a parameter file.

9.2.1 Main arguments

Some of the main arguments are only used for the Load function, some are only used for the Display function and others are common.

Common Arguments:

-qm *[qmgr name]* (optional)

The local queue manager name. If this argument is not specified, then the default queue manager is used.

-? (optional)

Help information. If this argument is specified then all other arguments are ignored.

Arguments for Loading State queue(s):

-l (required)

Specifies that the function is Load State queue(s)

-pf *[parameter file]* (required)

The parameter file path and name. See the next section for a description of the parameter file contents.

-s (optional)

Specifies that maxconnutil should only clear messages that relate to the associated ServerConnection channel. These will be messages with correlation ids of "B"*Channelname* and "V"*ChannelName*, where *ChannelName* is the name of the associated ServerConnection channel.
The default is that maxconnutil will clear all messages from the queue

Arguments for Displaying Contents of a State queue:

-d (required)

Specifies that the function is Display Contents of a State Queue

-cn (required)

The name of the ServerConnection channel served by the State Queue. The State queue name will be constructed as "MAXCONN."*ServerConnectionChannel*, where ServerConnectionChannel is the channel name (but see the -q argument for alternative queue names).

-q (optional)

It might be necessary to access the State data from a queue with a non-standard name - i.e. the State data has either been copied from a State queue to the queue in question, or the State queue is being accessed via another name. If so, then use the -q argument to specify the queue name.

9.3 The Load Function Parameter File

9.3.1 Overview

The load function loads Bootstrap state data onto a queue. The queue is called "MAXCONN."*ChannelName* where *ChannelName* is the name of the associated ServerConnection channel. If the queue exists then it is cleared of messages prior to loading. If the queue does not exist then it is created prior to loading.

The load function uses a parameter file that contains information about the Server Connection channels that are to be serviced by maxconn.

9.3.2 Parameters

The parameter file contains information about the Server Connection channels that are to be serviced by maxconn.

Multiple sets of channel parameters can be defined within the parameter file. Each set of channel parameters refers to a specific ServerConnection channel. Each set must be prefixed by the literal value "**ChannelParameters:** " – note that the value ends in a colon.

Comment lines must have the hash character ('#') in the first byte.

The channel parameter names are not case specific and can be specified in any order within a given set. The parameters are:

ServerConnChannel (required)

The name of the channel.

MaxConnections (required)

The maximum number of connections.

WarnThreshold (optional)

A threshold number of connections. When this is reached then a *WARN* event is raised.
If this argument is not specified then the warn threshold is set to zero and no *WARN* events are produced.

LogFileDirectory (optional)

The directory for the log file. Log data consists of event, trace and error information.
If the directory path includes spaces, then enclose the directory path in double quotes.
If a log file directory is specified then it must exist. If no log file directory is specified (or the specified directory does not exist) then the directory will default to c: (NT) or the home directory of the MCA user (Unix).

LogFileName (optional)

The name of the log file. Log data consists of event, trace and error information.
If no log file name is specified then the name defaults to MAXCONN.*QMgrName*.*ChannelName*.LOG.

MaxInUseAttempts (optional)

The maxconn exit opens the maxconn queue for EXCLUSIVE_INPUT. It is possible that there could be multiple concurrent connection attempts, which would result in MQOPEN calls returning MQRC_OBJECT_IN_USE. The *MaxInUseAttempts* parameter specifies how many times an instance of maxconn should try to open the queue before ending in error. There will be a wait of 200 milliseconds between each attempt.
If this parameter is not specified then the default value of 20 attempts will be used.

EventQueue (optional)

The name of the queue to which maxconn event messages will be written. If this parameter does not exist then events will be written to "MAXCONN.EVENT.QUEUE".

PcfCheckRatio (optional)

If maxconn determines that the number of existing connections for the relevant channel is equal to or greater than the *MaxConnections* for that channel, then maxconn will refuse the connection.
Before refusing the connection, maxconn has the ability to issue a PCF Inquire CHSTATUS request for the channel, to verify that the actual number of existing connections agrees with the *ExistingConnections* value held by the maxconn State data.
The default value for *PcfCheckRatio* is one – i.e. maxconn will issue a PCF inquiry prior to each and every connection refusal. If this is felt to be excessive, then the *PcfCheckRatio* parameter can be used to specify the ratio of connection refusals to PCF inquiries – for example, a *PcfCheckRatio* of 5 means that a PCF inquiry will be issued once every 5 refused connections.
A *PcfCheckRatio* value of zero means that no PCF inquiries will be issued.

TraceLevel (optional)

Trace information can be written from the exit to the log file. There are two levels of trace.
Valid values for the *TraceLevel* parameter are:

'no'	No trace information will be written (default). Note that error data will always be written to the log file.
'info'	Trace records are written when a connection request is accepted or refused, and when the connection is terminated.
'debug'	Additional, detailed trace records are written.

EventFlag (optional)

Specify whether event processing is active.

'no'	do not produce event messages (default)
'yes'	write event data to logfile
'mq'	write event data to logfile and MQ event queue

9.3.3 An example of a Parameter file

Here is an example of a parameter file:

Two ServerConnection channels will be using the maxconn exit.

The SVRCONN.BRANCH.TILLS ServerConnection channel is used by a real-time cash register application. There will be many of these applications and 350 per queue manager has been defined as the maximum. If the number of connections per queue manager exceeds 250 then a warn event is required. Events will be written to the default event queue MAXCONN.EVENT.QUEUE.

The SVRCONN.BRANCH.BKRM ServerConnection channel is used by a management application that is used irregularly and is not critical to the business. There will not be many of these applications and events are not necessary.

The maxconnutil parameter file is d:\maxconn\parms\parm.txt.

The local queue manager is QM01.

The maxconnutil console command will be:

Maxconnutil -qm QM01 -pf d:\maxconn\parms\parm.txt

The contents of the parameter file will be:

```
ChannelParameters:
ServerConnChannel           SVRCONN.BRANCH.TILLS
MaxConnections      350
WarnThreshold       250
EventFlag           mq
LogFileDirectory    d:\maxconn

ChannelParameters:
ServerConnChannel           SVRCONN.BRANCH.BKRM
MaxConnections      10
PcfCheckRatio       25
LogFileDirectory    d:\maxconn
```

The maxconn state queues will be called MAXCONN.SVRCONN.BRANCH.TILLS and MAXCONN.SVRCONN.BRANCH.BKRM. If these queues exist then maxconnutil will clear them of

messages and load a Bootstrap message. If these queues don't exist then maxconnutil will create them and load a Bootstrap message. A model queue called MAXCONN.MODEL.QUEUE must exist if maxconnutil is to create queues. It is recommended that the default persistence of this model queue is Not Persistent.

Note that the same file directory is used for both channel logs. The log file names are left to default. The default log names will be MAXCONN.QM01. SVRCONN.BRANCH.TILLS.LOG and MAXCONN.QM01.SVRCONN.BRANCH.BKRM.LOG.

Note that the TraceFlag is not set, so informational/debug trace logging is not active. The log will only be written to for events and errors.

10. Installation and Verification

10.1 Exit

maxconn is a security exit with an entry point called “maxconn”. The relevant ServerConnection channels must be updated to include a security exit property specifying the fully qualified path for maxconn and the entry point.

For example, if the location is *c:\utilities\mq\lib* , then the SCYEXIT value would be ‘*c:\utilities\mq\lib\maxconn(maxconn)’*. Remember to enclose the value in single quotes – this is necessary in order to preserve the entry point value as lower case.

10.2 Queues

If maxconnutil is to create State queues, then a model queue called MAXCONN.MODEL.QUEUE must be defined with a definition type of PERMDYN. It is recommended that the default persistence property of the model queue is NO (i.e. Not Persistent) – see Persistence for the reason.

If MQ events are enabled and the default maxconn event queue is to be used, then a queue called MAXCONN.EVENT.QUEUE must be defined. Event messages are explicitly persistent.

10.3 Quick Install and Verification

A quick installation of maxconn can be made using the provided sample files. The install instructions assume the use of a default queue manager and that the test install is on Windows:

1. Edit the sample MQ definitions file **vmqsc.txt**. This file creates the maxconn objects for the verification. Specify the location of the maxconn dll in the SCYEXIT parameter of the “alter channel” command. For example, if the location is *c:\utilities\mq\lib* , then the SCYEXIT value would be ‘*c:\utilities\mq\lib\maxconn(maxconn)’*. Remember to enclose the value in single quotes – this is necessary in order to preserve the entry point value as lower case.
2. Run “runmqsc < vmqsc.txt” from the install directory.
3. Edit the sample parameter file **vparm.txt**. Specify a directory for the log file.
4. Run “maxconnutil -l -pf vparm.txt” from the install directory. This will create and load a State queue called MAXCONN.VTEST.SVRCONN.
5. You now have a working maxconn environment.
6. Edit the file **vtestput.bat**. This file contains commands to set the MQSERVER environment variable and then issue an amqsputc against the test queue MAXCONN.TEST.Q. Specify the appropriate connection name for the MQSERVER variable.
7. Double click on the vtestput.bat file in Windows Explorer five times. This will result in five windows popping up. The first four will show a successful execution of amqsputc and will be awaiting input. The fifth window will show a failure with reason code 2059 (MQRC_QMGR_NOT_AVAILABLE). This is the expected result, because the sample parameter file specified that a maximum of 4 client connections could use the channel MAXCONN.SVRCONN.
8. Look in the log directory for the log file. The log file name will be “MAXCONN.*QmgrName*.VTEST.SVRCONN.LOG”. The trace level in the sample parameter file was “debug”, so the log file will contain records which describe the processing of the maxconn exit in

detail. The records will detail that four connection requests were received successfully and that one was refused.

9. The sample parameter file also enabled event logging to MAXCONN.EVENT.QUEUE. This queue will contain PCF format event messages. There will be two event messages. The first specifies that the number of connections has reached the warn threshold level of 3. The second event message specifies that the number of connections has reached the maximum level of 4. See Event Processing for a description of maxconn PCF event messages.
10. Run "maxconnutil –d –sc VTEST.SVRCONN" from the install directory. This will display the current contents of the State queue for the VTEST.SVRCONN channel. There will be two messages – a Bootstrap message and a Volatile message. The Volatile message contains the current state of the maxconn setup for VTEST.SVRCONN. Note that it specifies that the *ExistingConnections* are 4 and that the *EventState* is MAX.
11. Now close the five windows (opened due to double clicking on vtestput.bat).
12. Look at the log file again. It will detail that four disconnect requests were received.
13. The MAXCONN.EVENT.QUEUE will contain an additional two event messages (the third and fourth). The third message specifies that the connection count is within the warn threshold. The forth message specifies that the connection count is "OK" – i.e. is below the warn threshold. See Event Processing for a description of maxconn PCF event messages.
14. Run "maxconnutil –d –sc VTEST.SVRCONN" from the install directory. This will display the current contents of the State queue for the VTEST.SVRCONN channel. There will be two messages – a Bootstrap message and a Volatile message. The Volatile message contains the current state of the maxconn setup for VTEST.SVRCONN. Note that it specifies that the *ExistingConnections* are 0 and that the *EventState* is OK.

11. Un-Install

11.1 Exit

Remove the exit path and name from the SecurityExit property of the relevant ServerConnection channel(s).

Delete the exit module.

11.2 Queues

Delete MAXCONN.EVENT.QUEUE and MAXCONN.MODEL.QUEUE.

Delete all relevant MAXCONN.*ChannelName* queues, where *ChannelName* is the name of the associated ServerConnection channel.