TMQUEUE(8)

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

SYNOPSIS

tmqueue [*OPTIONS*]

Chapter 2

DESCRIPTION

This is special ATMI server which is used for transactional persistent queue operations. *tmqueue* is backend server used by *tpenqueue()* and *tpdequeue()* calls. Queue server works in pair with *tmsrv* instance, where both *tmqueue* and *tmsrv* are configured in XA environment which uses *NDRX_XA_RMLIB=libndrxxaqdisk.so* and *NDRX_XA_DRIVERLIB=libndrxxaqdisks.so*(static registration driver) or *NDRX_XA_DRIVERLIB=libndrxxaqdiskd.so*(dynamic registration driver). Each message basically is file in file-system which synchronously is kept in queue server's memory. The files in file system are staged between *active*, *prepared* and *committed* states. During the *active* and *prepared* stages messages are made after the XA transaction ID (with sequence number after dash). when message is committed, it is stored in *committed* folder, where file name contains the actual message id. This infrastructure also is used for issuing XA commands which does the remove and update of the message i.e. at the commit moment command files are used to update/remove committed message.

The folder where to store the processing qspace files are *NDRX_XA_OPEN_STR* and *NDRX_XA_CLOSE_STR* environment variables. You may combine multiple instances of *tmqueue* for the same qspace, but it is recommended to keep qspace folders different. However, it shall also work with same folder.

The queue server internally uses thread pools for handling the work. Multiple thread pools are used to avoid the deadlocking where internal processing might do the XA driver calls which in the end sends notifications to the same server back.

Three thread-pools are used:

1) First one is for accepting the ATMI *tpcall()* requests.

2) Second thread pool is used by forward services (which basically are senders for automatic queues).

3) Third thread pool is used by notifications received from transaction manager ™. TM notifies the queue server for completion (commit) of the message. So that *tmqueue* can unlock the message in memory.

Every instance of *tmqueue* will advertise following list of services:

1. @TMQ-<Cluster Node ID>-<Server ID>
2. @QSP<Queue space>

For example for Queue Space *MYSPACE*, Cluster Node ID *6*, Enduro/X Server ID *100* services will look like:

1. @TMQ-6-100
2. @QSPMYSPACE

The automatic forwarder pool, ever configured time, will scan the automatic queues for non-locked messages. Once such message is found, the message is submitted for worker thread. The worker thread will do the synchronous call to target server (*srvnm* from *q.conf*), wait for answer and either update tries counter or remove the message if succeed. If message is submitted with *TPQREPLYQ* then on success, the response message from invoked service is submitted to reply queue. If message destination service fails for number of attempts, then message is forwarded to *TPQFAILUREQ*, in this case original *TPQCTL* is preserved.

During the startup, *tmqueue* will try to read from disk the messages, if there are any *committed prepared*.

The *tpenqueue()* and *tpdequeue()* can be invoked either as part of the global transaction or not. If global transaction is not used, then *tmqueue* will open the transactions internally.

For more usage of the persistent queues, configuration, command line and codding samples, see *atmitests/test028_tmq* folder.

Chapter 3

OPTIONS

-m *QUEUE_SPACE*
The name of the queue space.

-q *QUEUE_CONFIG*
This either full or relative path to the queue definition file (see *q.conf(5)* for more info on syntax. When changes are done in this file, it is possible to reload the configuration during the runtime, with *xadmin mqrc* command. Note that queues are not removed, but changed or added.

[-s *SCAN_TIME*]
This time in seconds used by main forwarder thread to scan for any automatic messages to be submitted for processing.

[-p *SERVICE_THREAD_POOL_SIZE*]
This is thread pool size of used for *tpenqueue()*, *tpdequeue()* serving. Also this thread pool is utilized by *xadmin mqX* commands. Default is *10*.

[-u *NOTIFY_THREAD_POOL_SIZE*]
This is number of threads which are processing callbacks from XA driver (for operation completion notifications).

[-f *NOTIFY_THREAD_POOL_SIZE*]
This is number of worker threads for forwarder. Default value is *10*.

[-t *XA_TRANSACTION_TIMEOUT*]
Maximum number of seconds to wait for XA transaction to be completed. This used internally when global transactions are not used. Default is *30*.

[-m *BACKGROUND_TRIES*]
Maximum number of attempts to process locked transaction. Tries counter is increased by *-s* scan time steps. If tries count is reached, transaction becomes stalled. And it can be force re-attempted manually from xadmin only **xadmin commit** or **xadmin abort** for example. Default is *100*.

[-c *TIMEOUT_TEST_TIME*]
Number of seconds by which run periodical scans of the expired transactions. Default is *1*.

Chapter 4

EXIT STATUS

0
Success

1
Failure

Chapter 5

BUGS

Report bugs to support@mavimax.com

Chapter 6

SEE ALSO

xadmin(8) q.conf(5)

Chapter 7

COPYING