z/TPF MQS1 Driver
User's Guide

This page intentionally left blank.

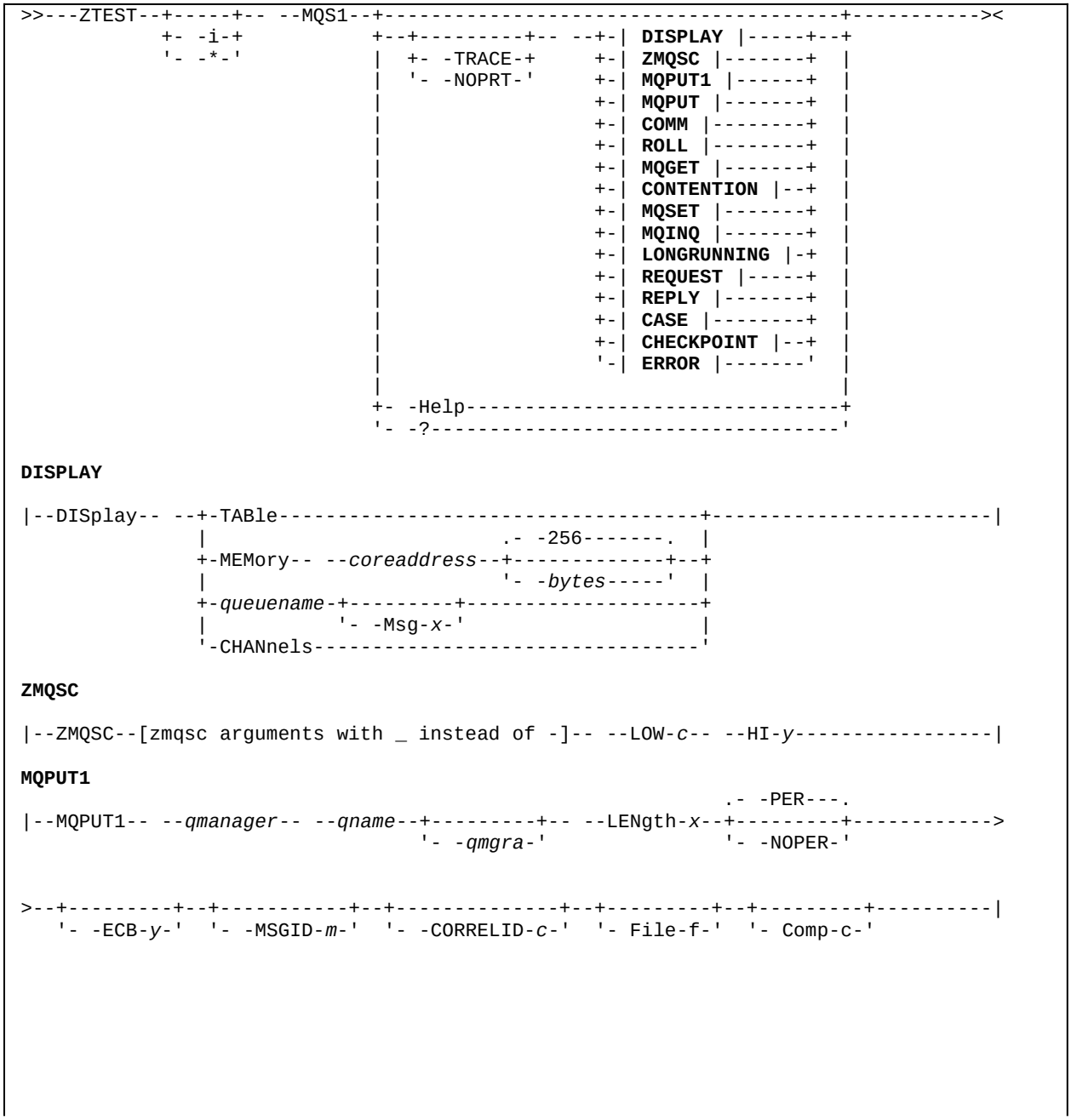
ZTEST MQS1

The MQS1 driver is a generic driver that test various areas of MQ support on TPF. It can imitate an application that performs various MQ API calls, such as MQPUT, MQGET, MQSET, etc. This driver also includes a test suite that contains various combinations of MQ APIs with transaction service provided by TPF.

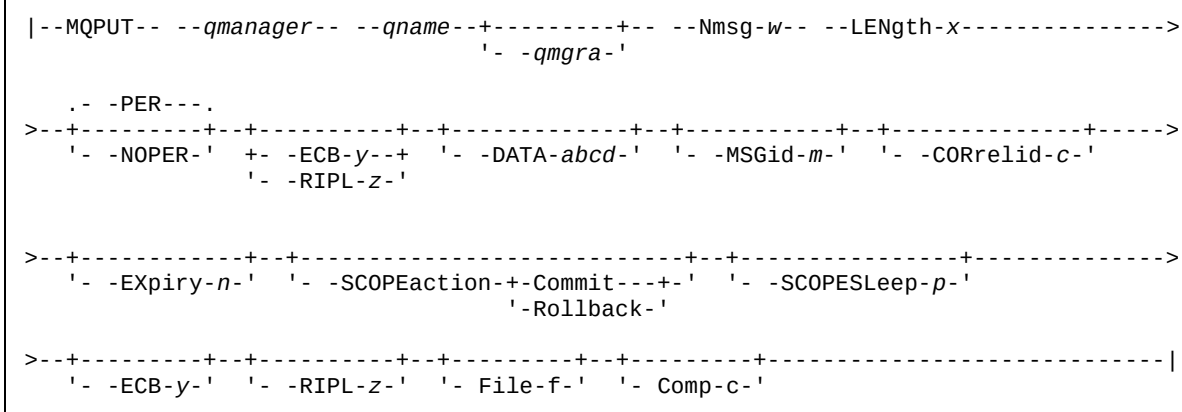
Requirements and restrictions

None.

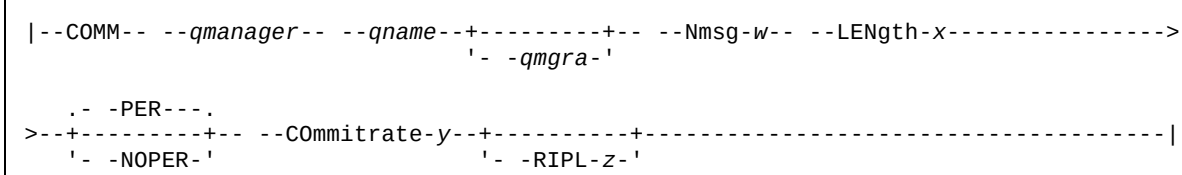
Format



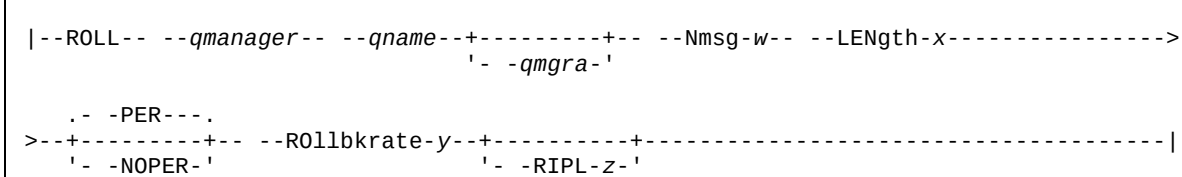
MQPUT



COMM

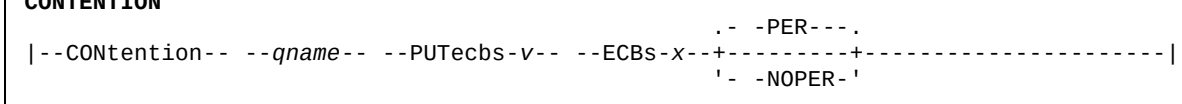


ROLL

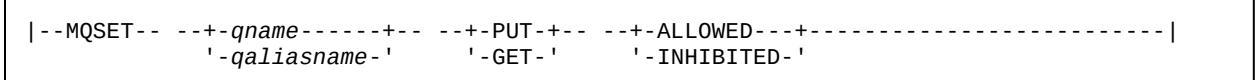


MQGET

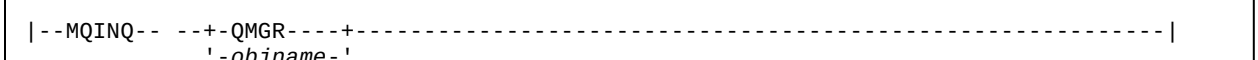
CONTENTION



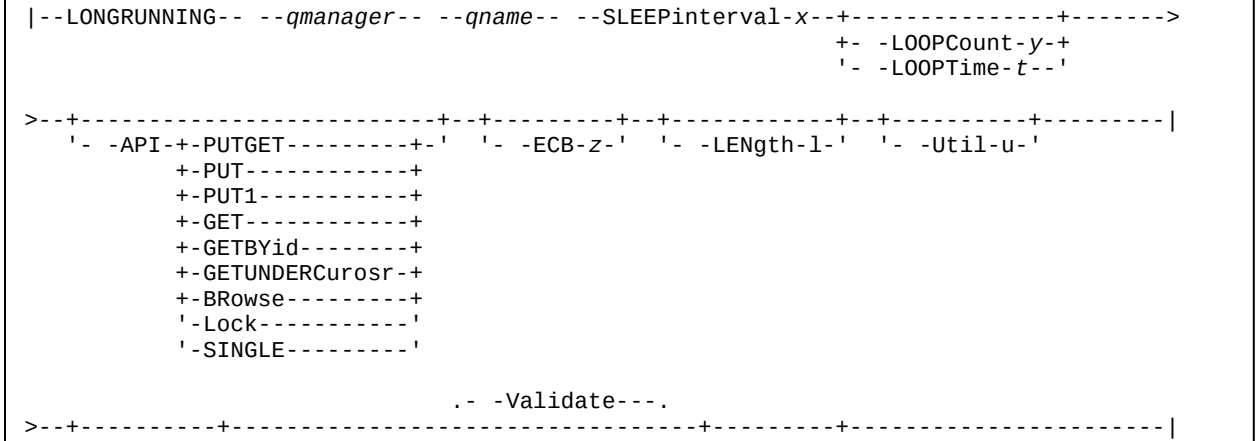
MQSET



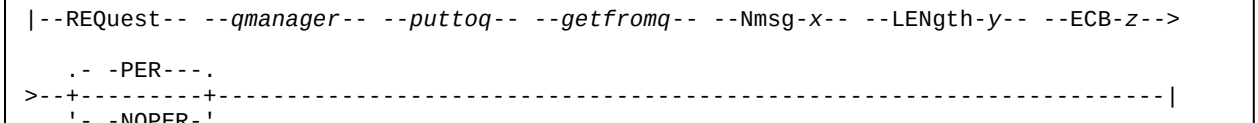
MQINQ



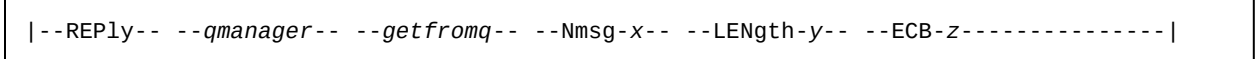
LONGRUNNING



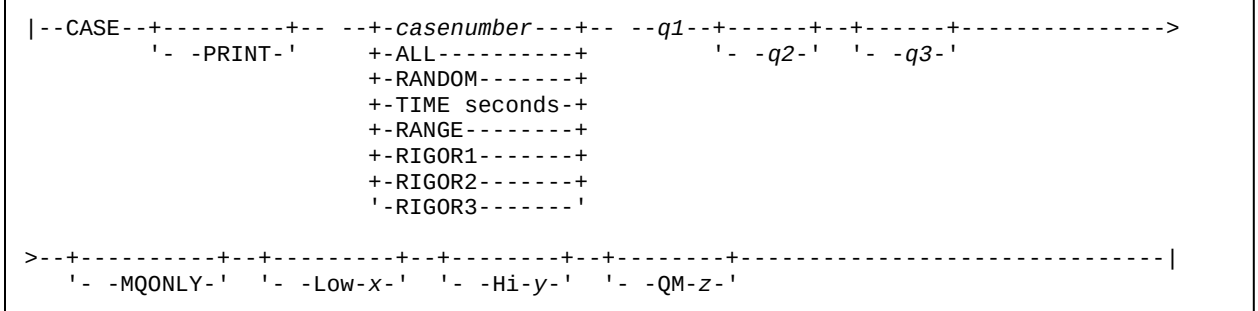
REQUEST



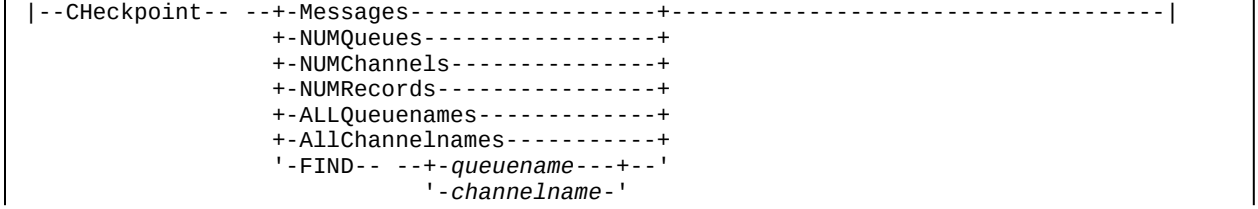
REPLY

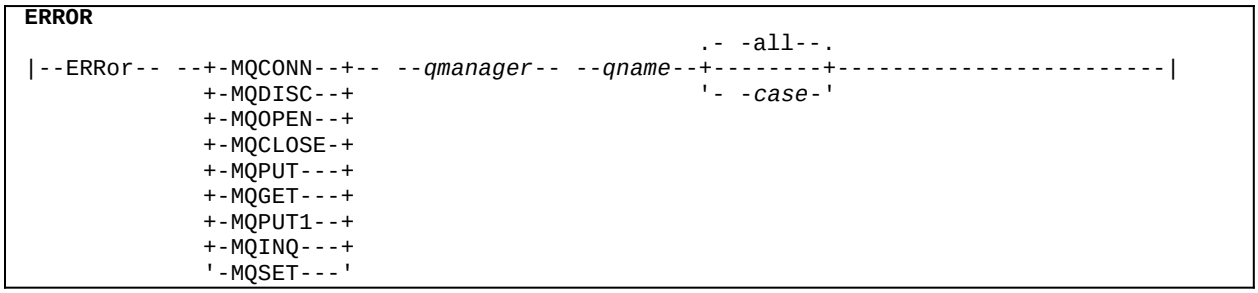


CASE



CHECKPOINT





i
indicates the specific I-stream in which the driver will be run. If *i* is not specified, the test case(s) will be executed on the I-stream on which the command is entered.

*****
specifies the driver will be invoked on all currently defined and available I-streams.

HELP | ?
displays help information on a function.

TRACE
displays every MQ APIs executed by the function.

NOPRT
suppresses all output except error messages.

DISplay
displays the core address of various MQ objects.

ZMQSC
enters a ZMQSC command through the driver.

MQPUT1
puts one message in a queue using the MQPUT1 API.

MQPUT
puts messages in a queue using the MQPUT API.

COMM | ROLL
tests the MQ APIs with tx_commit()/tx_rollback().

MQGET
gets messages from a queue using MQGET API.

CONTention
creates multiple ECBs to perform MQPUT and MQGET on the same queue.

MQSET
allows or prevents putting or getting messages from a queue.

MQINQ
displays information about a queue/process/queue manager.

LONGRUNNING
continuously does MQPUT/MQGET to a queue.

REQUEST | REPLY
sends request/reply type messages.

CASE
runs the test suite which tests various combinations of commits and rollbacks.

CHECKPOINT
searches the checkpoint for any queues or channels.

ERROR
runs a set of MQ API calls specifically testing error return paths.

Source code information

The MQS1 driver consists of the following program segments:

Header Files

Header File	Description
qmqz.h	This header file was specifically created to hold all QMQZ prototypes and various definitions.

Macros
None.

BSOs
None.

CSOs

Module	Makefile	Segment	Description
QMQZ	qmqz.mak	qmqz.cpp	Determines the command and jumps to the appropriate function in another qmqz*xx* file.
		qmqz01.cpp	Part of QMQZ.
		qmqz02.cpp	Subroutines used to regression test how the MQ API calls handle certain error conditions. This is useful to test the error path/handling of the APIs themselves.
		qmqz03.cpp	Used to send a request type message to a destination queue manager, and queue.
		qmqz04.cpp	The MQRC Lookup Table stores MQRC errors corresponding to the MQRC error numbers.
		qmqz05.cpp	Contains functions to gather information from MQ checkpointing.
		qmqz06.cpp	Suite of test cases for the MQ drivers.
		qmqz07.cpp	Part of CSO QMQZ

Additional information

None.

Examples

The following example displays the core addresses of MQ Tables, including: Queue Manager MQT_TO2 Table, Queue Definition Table, and Channel Definition Table.

```
ZTEST MQS1 DISplay TABle
```

The following example displays the name of each channel that has a channel definition in core.

```
ZTEST MQS1 DISplay CHANnels
```

The following example is the same as ZDCOR *coreaddress.bytes*, except it tells how many lines of zeros are skipped instead of simply not displaying lines of zeros. If *bytes* is not specified, 256 bytes are displayed.

```
ZTEST MQS1 DISplay MEMory coreaddress [bytes]
```

The following example displays the core address of queue *queuename* if *x* is not specified. If *x* is specified, it displays information about the *x*th message in core on queue *queuename*. The function only looks at the front message chain, so swept and unswept messages will not be found.

```
ZTEST MQS1 DISplay queuename [Msg-x]
```

The following example places one message of length *x* on *qname* defined in *qmanager*. PER defines the message to be persistent. NOPER defines the message to be not persistent. The default is as specified for the queue. If *y* is specified, *y* separate ECBs execute the operation (each putting one message). The *qmgralias* specifies the queue manager alias.

```
ZTEST MQS1 MQPUT1 qmanager qname [qmgralias] LENgth-x [NO]PER [ECB-y]
```

The following example places *w* messages of length *x* on *qname* defined in *qmanager*. PER defines the messages to be persistent. NOPER defines the messages to be not persistent. The default is as specified for the queue. If *y* is specified, *y* separate ECBs execute the operation, each putting *w* messages onto the queue. If *z* is specified, the system will re-IPL after the *z*th message is placed. If *z* is the total number of messages, the IPL occurs after the queue is closed. The queue manager alias is specified by *qmgra*.

```
ZTEST MQS1 MQPUT qmanager qname [qmgra] Nmsg-w LENgth-x [NO]PER
[ECB-y | RIPL-z]
```

In the following example, when putting only 1 msg, the exact input *msgid* and *correlid* would be used for the MSGID and CORRELID fields in MQMD. When more than 1 msg is put, input *msgid* and *correlid* with be appended with a number (starting 1 for msg 1, 2 for msg 2, etc) for each message. When entering MSGID and CORRELID, ensure that with the biggest number appended, the length of the MSGID or CORRELID is not bigger than 24 bytes. The DATA field overlays the first 4 bytes of the message data. The default is QMQZ.

```
ZTEST MQS1 MQPUT qmanager qname Nmsg-w Length-x MSGID-msgid CORRELID-correlid DATA-abcd
```

In the following example all the MQPUT actions would be in a single commit scope when SCOPEaction is specified. After all MQPUTs are done, the ECB will sleep for *p* seconds before committing/rolling back the whole transaction.

```
ZTEST MQS1 MQPUT qmanager qname Nmsg-w Length-x SCOPEaction-COMMIT|ROLLBACK SCOPESLeep-
p
```


The following example places *w* messages of length *x* and gets *w* messages of length on *qname* defined in *qmanager* through transactions. For every *y*th message placed, *y* messages are removed from the queue and a tx_commit commits the transaction. PER defines the messages to be persistent. NOPER defines the messages to be not persistent. The default is as specified for the queue. If *z* is specified, the system will re-IPL after the *z*th message is placed. If *z* is the total number of messages, the IPL occurs after the queue is closed. If *z* is the last message before a commit, the IPL occurs after the transaction is committed. The queue manager alias is specified by *qmgra*.

```
ZTEST MQS1 COMM qmanager qname [qmgra] Nmsg-w LENgth-x [NO]PER COmmitrate-y [RIPL-z]
```

The following example places *w* messages of length *x* on qname defined in *qmanager* through transactions. For every *y*th message placed, *y*-1 messages are removed from the queue and tx_rollback prevents the transaction from going to the queue. PER defines the messages to be persistent. NOPER defines the messages to be not persistent. The default is as specified for the queue. If *z* is specified, the system will re-IPL after the *z*th message is placed. If *z* is the total number of messages, the IPL occurs after the queue is closed. If *z* is the last message before a rollback, the IPL occurs after the transaction is rolled back. The queue manager alias is specified by *qmgra*.

```
ZTEST MQS1 ROLL qmanager qname [qmgra] Nmsg-w LENgth-x [NO]PER ROllbackrate-y [RIPL-z]
```

The following example removes *w* messages from *qname* defined in *qmanager*. PRT displays the first *x* characters of the messages removed. NOPRT prevents display of message content. If you do not specify PRT or NOPRT, the messages are displayed if *w* is <= 25, otherwise no messages are displayed. If *y* is specified, *y* separate ECBs execute the operation, each getting *w* messages from the queue. SEQcheck causes out-of-sequence messages to produce a warning message on the console.

```
ZTEST MQS1 MQGET qmanager qname Nmsg-w LENgth-x [NO]PRT [ECB-z] [NO]SEQcheck
```

The following example is the same as the previous MQGET, except the ECB tries MQGETs for *w* seconds instead of removing *w* messages.

```
ZTEST MQS1 MQGET qmanager qname Sec-w LENgth-x [NO]PRT [ECB-z]
```

The following example removes *w* messages from *qname* defined in *qmanager*. If the queue is empty, the ECB will wait *y* seconds or until a message is added to the queue. If *y* is -1, the ECB will wait indefinitely for a message to be added to the queue. The first *n* messages are displayed.

```
ZTEST MQS1 MQGET qmanager qname Nmsg-w LENgth-x WAIT-y PRTN-n
```

The following example removes the first message that has message ID of *m* and correlation ID of *c*.

```
ZTEST MQS1 MQGET qmanager qname Nmsg-w LENgth-x MSGID-m CORRELID-c
```

The following example browses *w* messages The messages are not removed from the queue.

```
ZTEST MQS1 MQGET qmanager qname Nmsg-w LENgth-x BRowse-READ
```

The following example browses *w* messages. If *w* messages are successfully browsed, the *w*th message is removed from the queue.

```
ZTEST MQS1 MQGET qmanager qname Nmsg-w LENgth-x BRowse-GET
```

The following example browses *w* messages. If *w* messages are successfully browsed, the *w*th message is locked by this driver ECB.

```
ZTEST MQS1 MQGET qmanager qname Nmsg-w LENgth-x BRowse-LOCK
```

The following example browses *w* messages. If *w* messages are successfully browsed, the Browse *action* (READ|GET|LOCK) will be performed on the *w*th message. Then, the ECB goes to sleep for *p* seconds. When the driver ECB wakes up, it will perform the *wakeAction* (Exit|SERRCexit|LOCK|LOCKNext|UNLOCK|Browse|BRNext|GET).

```
ZTEST MQS1 MQGET qmanager qname Nmsg-w LENgth-x Browse-action SLEEP-p WAKEUP-
wakeAction
```

The following example browses *w* messages that match the *msgid*. If *w* messages are successfully browsed, the Browse *action* (READ|GET|LOCK) will be performed on the *w*th message. Then, the ECB goes to sleep for *p* seconds. When the driver ECB wakes up, it will perform the *wakeAction* (Exit|SERRCexit|LOCK|LOCKNext|UNLOCK|Browse|BRNext|GET).

```
ZTEST MQS1 MQGET qmanager qname Nmsg-w LENgth-x MSGID-msgid Browse-action SLEEP-p
WAKEUP-wakeAction
```

In the following example all the MQGET actions would be in a single commit scope when SCOPEaction is specified. After all MQGETs are done, the ECB will sleep for *p* seconds before committing/rolling back the whole transaction.

```
ZTEST MQS1 MQGET qmanager qname Nmsg-w Length-x SCOPEaction-COMMIT|ROLLBACK
SCOPESLeep-p
```

The following example creates *z* ECBs to perform the MQ APIs continuously, for *y* times (when LOOPCount is specified), or for *t* seconds (when LOOPTime is specified).

```
ZTEST MQS1 LONGRUNNING qmanager qname SLEEP-x [LOOPCount-y|LOOPTime-t] API-api ECB-z
```

The following example creates *w* ECBs putting and getting to queue *qname*. Each ECB puts *x* messages and gets *w-x* messages, where *x* is the *x*th ECB created.

```
ZTEST MQS1 CONtention qname ECBs-w (NO)PER
```

The following example allows or prevents PUTing or GETting from *qname* or *qaliasname*.

```
ZTEST MQS1 MQSET qname|qaliasname PUT|GET ALLOWED|INHIBITED
```

The following example outputs information about the running Queue Manager when QMGR is specified. If *objname* is specified, the driver will inquire any queue with the name *objname*. If the inquiry fails, the driver will then inquire any process with the name *objname*.

```
ZTEST MQS1 MQINQ QMGR|objname
```

The following example sends *x* request type messages of length *y* to a destination queue manager *qmanager*, and a defined queue *puttoq*. The user must also specify the reply-to queue *getfromq* to which the original message will be echoed. If *z* is specified, *z* separate ECBs execute the operation, each sending *x* messages.

```
ZTEST MQS1 REQuest qmanager puttoq getfromq Nmsg-x LENgth-y ECB-z
```

The following example sends reply messages to MQ request messages by removing *x* messages from *getfromq* defined in *qmanager* and replying to each request message found. If *z* is specified, *z* separate ECBs execute the operation, each replying to *x* messages.

```
ZTEST MQS1 REPly qmanager getfromq Nmsg-x LENgth-y ECB-z
```

The following example displays total number of messages in the checkpoint. Also shows the number of messages in each queue in the checkpoint, separated into messages prior to sweep and messages after sweep.

```
ZTEST MQS1 CHeckpoint Messages
```


The following example displays total number of queues in the checkpoint.

```
ZTEST MQS1 CHeckpoint NUMQueues
```

The following example displays total number of channels in the checkpoint.

```
ZTEST MQS1 CHeckpoint NUMChannels
```

The following example displays total number of records used to store the checkpoint.

```
ZTEST MQS1 CHeckpoint NUMRecords
```

The following example displays the name of each queue in the checkpoint.

```
ZTEST MQS1 CHeckpoint ALLQueuenames
```

The following example displays the name of each channel in the checkpoint.

```
ZTEST MQS1 CHeckpoint ALLChannelnames
```

The following example searches the checkpoint for any queues or channels by the name *queuename* or *channelname*.

```
ZTEST MQS1 CHeckpoint FIND queuename|channelname
```

The following example runs many ZMQSC commands, each command the same as the arguments entered, except numbers from *x* to *y* are appended to the end of the second argument. This example covers ZTEST MQS1 ZMQSC [zmqsc arguments with _ instead of -] LOW-*x* HI-*y*.

```
Example:
  ZTEST MQS1 ZMQSC DEF QL_LOCALQ COMMON_YES LOW-4 HI-7
will run:
  ZMQSC DEF QL-LOCALQ004 COMMON-YES
  ZMQSC DEF QL-LOCALQ005 COMMON-YES
  ZMQSC DEF QL-LOCALQ006 COMMON-YES
  ZMQSC DEF QL-LOCALQ007 COMMON-YES
```

The following example runs a set of MQ API calls specifically testing error return paths.

```
ZEST MQS1 ERRor MQCONN|MQOPEN|MQPUT|MQGET|MQPUT1 qmanager qname [case]
                MQINQ|MQSET|MQCLOSE|MQDISC qmanager qname [case]
```

References

For more information about reading syntax diagrams, also referred to as railroad diagrams, see *Accessibility information* in the TPF Product Information Center.

Appendix A: The Test Suite in qmqz06.cpp

Put in Root, Get in Root

Case 1
```
tx_begin
 Put(Msg 1)
 Get(Msg 1)
tx_commit
```

Case 2
```
tx_begin
 Put(Msg 1)
 Tx_begin
  Get(Msg 1)
 Tx_commit
 Get(NOMSG)
Tx_commit
```

Case 3
```
Tx_begin
 Put(Msg 1)
 Tx_begin
  Get(Msg 1)
 Tx_rollback
 Get(Msg 1)
Tx_commit
```

Case 4
```
Tx_begin
 Put(Msg 1)
 Tx_begin
  Put(Msg 2)
  Get(Msg 1)
 Tx_commit
 Get(Msg 2)
Tx_commit
```

Case 5
```
Tx_begin
 Put(Msg 1)
 Tx_begin
  Put(Msg 2)
  Get(Msg 1)
 Tx_rollback
 Get(Msg 1)
tx_commit
```

Case 6
```
Tx_begin
 Put(Msg 1)
 Tx_begin
  Put(Msg 2)
 Tx_commit
 Get(Msg 1)
 Get(Msg 2)
Tx_commit
```

Case 7
```
Tx_begin
 Put(Msg 1)
 Tx_Begin
  Put(Msg 2)
 Tx_rollback
 Get(Msg 1)
 Get(NOMSG)
Tx_commit
```

Put in Root, Get in Nest

Case 8
```
Tx_begin
 Put(Msg 1)
 Tx_begin
  Get(Msg 1)
 Tx_commit
tx_commit
```

Case 9
```
Tx_begin
 Put(Msg 1)
 Tx_begin
  Tx_begin
   Put(Msg 2)
  Tx_commit
  Get(Msg 1)
  Get(Msg 2)
 Tx_commit
tx_commit
```

Case 10
```
Tx_begin
 Put(Msg 1)
 Tx_begin
  Tx_begin
   Put(Msg 2)
  Tx_rollback
 Get(Msg 1)
 Get(NOMSG)
 Tx_commit
Tx_commit
```

Case 11
```
Tx_begin
 Put(Msg 1)
 Tx_begin
  Tx_begin
   Get(Msg 1)
  Tx_rollback
 Get(Msg 1)
 Get(NOMSG)
 Tx_commit
Tx_commit
```

Case 12
```
Tx_begin
 Put(Msg 1)
 Tx_begin
  Tx_begin
   Get(Msg 1)
  Tx_commit
 Get(NOMSG)
 Tx_commit
Tx_commit
```

Put in Nest, Get in Nest

Case 13
```
Tx_begin
 Tx_begin
  Put(Msg 1)
  Tx_begin
   Put(Msg 2)
  Tx_commit
  Get(Msg 1)
  Get(Msg 2)
 Tx_commit
tx_commit
```

Case 14
```
Tx_begin
 Tx_begin
  Put(Msg 1)
  Tx_begin
   Put(Msg 2)
  Tx_rollback
  Get(Msg 1)
  Get(NOMSG)
 Tx_commit
tx_commit
```

Case 15
```
Tx_begin
 Tx_begin
  Put(Msg 1)
  Tx_begin
   Get(Msg 1)
  Tx_commit
  Get(NOMSG)
 Tx_commit
tx_commit
```

Case 16
```
Tx_begin
 Tx_begin
  Put(Msg 1)
  Tx_begin
   Get(Msg 1)
  Tx_rollback
  Get(Msg 1)
 Tx_commit
tx_commit
```


Put in Nest, Get in Root

Case 17
```
Tx_begin
 Tx_begin
  Put(Msg 1)
 Tx_commit
 Get(Msg 1)
tx_commit
```

Case 18
```
Tx_begin
 Tx_begin
  Put(Msg 1)
 Tx_rollback
 Get(NOMSG)
tx_commit
```

Case 19
```
Tx_begin
 Tx_begin
  Put(Msg 1)
 Tx_commit
 Put(Msg 2)
 Get(Msg 1)
 Get(Msg 2)
tx_commit
```

Case 20
```
Tx_begin
 Tx_begin
  Put(Msg 1)
 Tx_rollback
 Put(Msg 2)
 Get(Msg 2)
tx_commit
```

Put out of Scope, Get in Root

Case 21
```
Put(Msg 1)
Tx_begin
 Get(Msg 1)
Tx_commit
```

Case 22
```
Put(Msg 1)
Tx_begin
 Tx_begin
  Put(Msg 2)
 Tx_commit
 Get(Msg 1)
 Get(Msg 2)
Tx_commit
```

Case 23
```
Put(Msg 1)
Tx_begin
 Tx_begin
  Put(Msg 2)
 Tx_rollback
Get(Msg 1)
Get(NOMSG)
Tx_commit
```

Case 24
```
Put(Msg 1)
Tx_begin
 Tx_begin
  Get(Msg 1)
 Tx_rollback
Get(Msg 1)
Get(NOMSG)
Tx_commit
```

Case 25
```
Put(Msg 1)
Tx_begin
 Tx_begin
  Get(Msg 1)
 Tx_commit
 Get(NOMSG)
Tx_commit
```

Put in Root, Get out of Scope

Case 26
```
Put(Msg1)
Tx_begin
 Tx_begin
  Get(Msg1)
 Tx_commit
tx_commit
```

Case 27
```
Put(Msg1)
Tx_begin
 Put(Msg2)
 Tx_begin
  Get(Msg1)
  Get(Msg2)
 Tx_commit
tx_commit
```

Put in Root, Get out of Scope

Case 28
```
Tx_begin
 Put(Msg 1)
Tx_commit
Get(Msg 1)
```

Case 29
```
Tx_begin
 Put(Msg 1)
Tx_rollback
Get(NOMSG)
```

Case 30
```
Tx_begin
 Put(Msg 1)
Tx_commit
Put(Msg 2)
Get(Msg 1)
Get(Msg 2)
```

Case 31
```
Tx_begin
 Put(Msg 1)
Tx_rollback
Put(Msg 2)
Get(Msg 2)
```

Put in Nest, Get out of Scope

Case 32
```
Tx_begin
 Tx_begin
  Put(Msg 1)
 Tx_commit
Tx_commit
Get(Msg 1)
```

Case 33
```
Tx_begin
 Tx_Begin
  Put(Msg 1)
 Tx_commit
Tx_rollback
Get(NOMSG)
```

Case 34
```
Tx_begin
 Tx_begin
  Put(Msg 1)
 Tx_rollback
Tx_commit
Get(NOMSG)
```

Case 35
```
Tx_begin
 Tx_begin
  Put(Msg 1)
 Tx_rollback
 Put(Msg 2)
Tx_rollback
Get(NOMSG)
```

Case 36
```
Tx_begin
 Tx_begin
  Put(Msg 1)
 Tx_commit
 Put(Msg 2)
Tx_commit
Get(Msg 1)
Get(Msg 2)
```

Case 37
```
Tx_begin
 Tx_begin
  Put(Msg 1)
 Tx_commit
 Put(Msg 2)
Tx_rollback
Get(NOMSG)
```

Case 38
```
Tx_begin
 Tx_begin
  Put(Msg 1)
 Tx_rollback
 Put(Msg 2)
Tx_commit
Get(Msg 2)
Get(NOMSG)
```

Case 39
```
Tx_begin
 Tx_begin
  Put(Msg 1)
 Tx_rollback
 Put(Msg 2)
Tx_rollback
Get(NOMSG)
```

Put out of Scope, Get out of Scope

Case 40
```
Put(Msg 1)
Tx_begin
 Get(Msg 1)
Tx_commit
Get(NOMSG)
```

Case 41
```
Put(Msg 1)
Tx_begin
 Get(Msg 1)
Tx_rollback
Get(Msg 1)
```

Case 42
```
Put(Msg 1)
Tx_begin
 Put(Msg 2)
Tx_commit
Get(Msg 1)
Get(Msg 2)
```

Case 43
```
Put(Msg 1)
Tx_begin
 Put(Msg 2)
Tx_rollback
Get(Msg 1)
Get(NOMSG)
```

Storage Block Message Sizes

Case 44
```
Put(Msg 1, Size 0)
Get(Msg 1, Size 0)
```

Case 45
```
Put(Msg 1, Size 1)
Get(Msg 1, Size 1)
```

Case 46
```
Put(Msg 1, Size FirstMsgSegmentSize)
Get(Msg 1, Size FirstMsgSegmentSize)
```

Case 47
```
Put(Msg 1, Size FirstMsgSegmentSize + 1)
Get(Msg 1, Size FirstMsgSegmentSize + 1)
```

Case 48
```
Put(Msg 1, Size FirstMsgSegmentSize + NextMsgSegmentSize)
Get(Msg 1, Size FirstMsgSegmentSize + NextMsgSegmentSize)
```

Case 49
```
Put(Msg 1, Size FirstMsgSegmentSize + NextMsgSegmentSize + 1)
Get(Msg 1, Size FirstMsgSegmentSize + NextMsgSegmentSize + 1)
```

Case 50
```
Put(Msg 1, Size FirstMsgSegmentSize + NextMsgSegmentSize + NextMsgSegmentSize)
Get(Msg 1, Size FirstMsgSegmentSize + NextMsgSegmentSize + NextMsgSegmentSize)
```

Put to Multiple Queues

Case 51
```
Put(Msg 1, Msg 2, Msg 3 to Q1, Q2, Q3)
Put(Msg 4, Msg 5, Msg 6 to Q1, Q2, Q3)
Get(Msg 1, Msg 2, Msg 3 from Q1, Q2, Q3)
Get(Msg 4, Msg 5, Msg 6 from Q1, Q2, Q3)
```


Note: For this test case, the queue Handles are closed implicitly.

Case 52

```
Tx_begin
 Put(Msg 1, Msg 2, Msg 3 to Q1, Q2, Q3)
 Put(Msg 4, Msg 5, Msg 6 to Q1, Q2, Q3)
 Get(Msg 1, Msg 2, Msg 3 from Q1, Q2, Q3)
 Get(Msg 4, Msg 5, Msg 6 from Q1, Q2, Q3)
Tx_commit
```

Case 53

```
Tx_begin
 Put(Msg 1, Msg 2, Msg 3 to Q1, Q2, Q3)
 Put(Msg 4, Msg 5, Msg 6 to Q1, Q2, Q3)
 Get(Msg 1, Msg 2, Msg 3 from Q1, Q2, Q3)
 Get(Msg 4, Msg 5, Msg 6 from Q1, Q2, Q3)
Tx_rollback
Get(NOMSG, NOMSG, NOMSG from Q1, Q2, Q3)
```

Case 54

```
Tx_begin
 Tx_begin
  Put(Msg 1, Msg 2, Msg 3 to Q1, Q2, Q3)
 Tx_commit
 Put(Msg 4, Msg 5, Msg 6 to Q1, Q2, Q3)
Tx_commit
Get(Msg 1, Msg 2, Msg 3 from Q1, Q2, Q3)
Get(Msg 4, Msg 5, Msg 6 from Q1, Q2, Q3)
```

Case 55

```
Tx_begin
 Tx_begin
  Put(Msg 1, Msg 2, Msg 3 to Q1, Q2, Q3)
 Tx_rollback
 Put(Msg 4, Msg 5, Msg 6 to Q1, Q2, Q3)
Tx_commit
Get(Msg 4, Msg 5, Msg 6 from Q1, Q2, Q3)
```

Case 56

```
Tx_begin
 Put(Msg 1, Msg 2, Msg 3 to Q1, Q2, Q3)
 Tx_begin
  Put(Msg 4, Msg 5, Msg 6 to Q1, Q2, Q3)
  Put(Msg 7, Msg 8, Msg 9 to Q1, Q2, Q3)
  Put(Msg 10, Msg 11, Msg 12 to Q1, Q2, Q3)
  Put(Msg 13, Msg 14, Msg 15 to Q1, Q2, Q3)
  Get(Msg 1, Msg 2, Msg 3 from Q1, Q2, Q3)
  Get(Msg 4, Msg 5, Msg 6 from Q1, Q2, Q3)
 Tx_rollback
 Put(Msg 16, Msg 17, Msg 18 to Q1, Q2, Q3)
Tx_commit
Get(Msg 1, Msg 2, Msg 3 from Q1, Q2, Q3)
Get(Msg 16, Msg 17, Msg 18 from Q1, Q2, Q3)
Get(NOMSG, NOMSG, NOMSG from Q1, Q2, Q3)
```

Large Message on Queue

Case 57
```
Put(40000 byte Message 1)
Get(40000 byte Message 1)
```

Case 58
```
Tx_begin
  Put(40000 byte Message 1)
  Get(40000 byte Message 1)
Tx_commit
```

Case 59
```
Tx_begin
  Put(40000 byte Message 1)
Tx_commit
Get(40000 byte Message 1)
```

Case 60
```
Tx_begin
  Put(40000 byte Message 1)
Tx_rollback
Get(NOMSG)
```

Case 61
```
Put(40000 byte Message 1)
Tx_begin
  Get(40000 byte Message 1)
Tx_commit
```

Case 62
```
Tx_begin
  Put(40000 byte Message 1)
  Put(0 byte Message 2)
Tx_commit
Get(40000 byte Message 1)
Get(0 byte Message 3)
```

Version 63
```
Tx_begin
  Put(1 Megabyte Message 1)
Tx_commit
Get(1 Megabyte Message 1)
```

How to modify a queue to handle large messages:

ZCTKA A MMHS-1024	to increase maximum malloc size in system.
ZRIPL	for ZCTKA to take effect.
ZMQSC ALT MQP-*qmgr* MAXMSGL-1500000	increase queue manager max message length.
ZMQSC ALT QL-*queue* MAXMSGL-1500000	increase queue max message length.

Multiple Transaction Scopes

Case 64
```
Put(Msg 1)
Tx_begin
 Get(Msg 1)
Tx_commit
Get(NOMSG)
```

Case 65
```
Put(Msg 1)
Tx_begin
 Get(Msg 1)
Tx_rollback
Get(Msg 1)
```

Case 66
```
Put(Msg 1)
Tx_begin
 Put(Msg 2)
Tx_commit
Get(Msg 1)
Get(Msg 2)
```

Case 67
```
Put(Msg 1)
Tx_begin
 Put(Msg 2)
Tx_rollback
Get(Msg 1)
Get(NOMSG)
```

Truncated Messages

Case 68
```
MQPUT(10 bytes of 'A')
Get Options = MQGMO_NONE
MQGET(0 bytes, look for TRUNCATED MSG FAILED)
Get Options = MQGMO_ACCEPT_TRUNCATED_MSG
MQGET(0 bytes, look for TRUNCATED MSG ACCEPTED)
```

Case 69
```
tx_begin()
 MQPUT(100 bytes of 'A')
 Get Options = MQGMO_ACCEPT_TRUNCATED_MSG
 MQGET(50 bytes, look for TRUNCATED MSG ACCEPTED, check message)
tx_commit()
```

Case 70


```
tx_begin()
 MQPUT(100 bytes of 'B')
 Get Options = MQGMO_ACCEPT_TRUNCATED_MSG
 MQGET(50 bytes, look for TRUNCATED MSG ACCEPTED, check message)
tx_rollback()
```

Case 71

```
tx_begin()
 MQPUT(100 bytes of 'C')
 Get Options = MQGMO_NONE
 MQGET(50 bytes, look for TRUNCATED MSG FAILED)
tx_commit()
Get Options = MQGMO_ACCEPT_TRUNCATED_MSG _
MQGET(50 bytes, look for TRUNCATED MSG ACCEPTED, check message)
```

Case 72

```
tx_begin()
 MQPUT(100 bytes of 'C')
 Get Options = MQGMO_NONE
 MQGET(50 bytes, look for TRUNCATED MSG FAILED)
Tx_rollback()
```

Get with Wait

Case 101

```
for x = each supported Get option:
  Get Options = x + MQGMO_WAIT
  Get(look for appropriate return code)
```

Case 102

```
Get option = MQGMO_NONE
Put(Msg 1)
Get(Msg 1)
```

Case 103

```
Get Options = MQGMO_WAIT
Wait Interval = MQWI_UNLIMITED
Put(Msg 1)
Get(Msg 1)
```

Case 104

```
Get Options = MQGMO_WAIT
for WaitInterval = {-3, -9, -12, -5}
  Get(look for MQRC_WAIT_INTERVAL_ERROR)
```

Browse Option Errors

The following MQGMO options are used in test cases 200-203:

```
MQGMO_NONE
MQGMO_MSG_UNDER_CURSOR
MQGMO_LOCK | MQGMO_UNLOCK | MQGMO_BROWSE_NEXT
MQGMO_BROWSE_FIRST | MQGMO_BROWSE_NEXT
MQGMO_LOCK
MQGMO_UNLOCK | MQGMO_WAIT
MQGMO_UNLOCK | MQGMO_BROWSE_FIRST
MQGMO_LOCK | MQGMO_ACCEPT_TRUNCATED_MSG
MQGMO_BROWSE_FIRST
MQGMO_BROWSE_NEXT
MQGMO_BROWSE_MSG_UNDER_CURSOR
MQGMO_LOCK | MQGMO_UNLOCK
MQGMO_BROWSE_MSG_UNDER_CURSOR | MQGMO_MSG_UNDER_CURSOR
MQGMO_UNLOCK | MQGMO_MSG_UNDER_CURSOR
```

Case 200

```
Queue is opened for browse only.
for x = various MQGMO options (see above)
   Get Options = x
```

```
   Get(look for appropriate return codes)

Case 201
Queue is opened for input and browse.
for x = various MQGMO options (see above)
   Get Options = x
   Get(look for appropriate return codes)

Case 202
Queue is opened for output only.
for x = various MQGMO options (see above)
   Get Options = x
   Get(look for appropriate return codes)

Case 203
Queue is opened for input only.
for x = various MQGMO options (see above)
   Get Options = x
   Get(look for appropriate return codes)

Case 204
Put 100 character message.
Input buffer size is 50.
Get Options = MQGMO_NONE
Get(look for MQRC_TRUNCATED_MSG_FAILED)
Get Options = MQGMO_ACCEPT_TRUNCATED_MSG
Get(look for MQRC_TRUNCATED_MSG_ACCEPTED)
Get(look for MQRC_NO_MSG_AVAILABLE)

Case 205
Open Options = MQOO_BROWSE | MQOO_OUTPUT
tx_begin
Put three messages ( A , B,
tx_commit
Get Options = MQGMO_BROWSE_NEXT
Get A (look foMQRC_NONE)
Get B (look foMQRC_NONE)
Get C (look foMQRC_NONE)
Get (look for MQRC_NO_MSG_AVAILABLE)

Case 206
Open Options = MQOO_BROWSE | MQOO_OUTPUT
tx_begin
Put 100 character message
tx_commit
Input buffer size is 50.
Get Options = MQGMO_BROWSE_FIRST
Get (look for MQRC_TRUNCATED_MSG_FAILED)
Get Options = MQGMO_ACCEPT_TRUNCATED_MSG | MQGMO_BROWSE_NEXT
Get (look for MQRC_TRUNCATED_MSG_ACCEPTED)
Get (look for MQRC_NO_MSG_AVAILABLE)

Case 207
Open Options = MQOO_BROWSE | MQOO_OUTPUT
tx_begin
Put message A
tx_commit
```



```
tx_begin
Put message B
Get Options = MQGMO_BROWSE_FIRST
Get A (look foMQRC_NONE)
Get (look for MQRC_NO_MSG_AVAILABLE)
tx_commit
```

Case 208

```
Open Options = MQOO_BROWSE | MQOO_OUTPUT | MQOO_INPUT_AS_Q_DEF
tx_begin
Put message.
tx_commit
Get Options = MQGMO_BROWSE_FIRST
Get (look for MQRC_NONE)
Get Options = MQGMO_MSG_UNDER_CURSOR
Get (look for MQRC_NONE)
Get (look for MQRC_NO_MSG_UNDER_CURSOR)
```

Case 209

```
Open Options = MQOO_BROWSE | MQOO_OUTPUT | MQOO_INPUT_AS_Q_DEF
tx_begin
Put message.
tx_commit
Get Options = MQGMO_BROWSE_FIRST
Get (look for MQRC_NONE)
Get Options = MQGMO_BROWSE_MSG_UNDER_CURSOR
Get (look for MQRC_NONE)
Get Options = MQGMO_MSG_UNDER_CURSOR
Get (look for MQRC_NONE)
```

Case 210

```
Open Options = MQOO_BROWSE | MQOO_OUTPUT
tx_begin
Put message.
tx_commit
Get Options = MQGMO_BROWSE_FIRST
Get (look for MQRC_NONE)
Get Options = MQGMO_BROWSE_NEXT
Get (look for MQRC_NO_MSG_AVAILABLE)
Get Options = MQGMO_BROWSE_MSG_UNDER_CURSOR
Get (look for MQRC_NONE)
```

Case 211

```
Open Options = MQOO_BROWSE | MQOO_OUTPUT
Requires 1 queue and 2 handles.
tx_begin
Put message.
tx_commit
Get Options = MQGMO_BROWSE_FIRST | MQGMO_LOCK
Get (look for MQRC_NONE, handle 1)
Get Options = MQGMO_BROWSE_FIRST
Get (look for MQRC_NO_MSG_AVAILABLE, Handle 2)
Get (look for MQRC_NONE, handle 1)
Get (look for MQGMO_NONE, handle 2)
```

Case 212

```
Open Options = MQOO_BROWSE | MQOO_OUTPUT
Requires 1 queue and 2 handles.
```

```
tx_begin
Put message A
Put message B
tx_commit
Get Options = MQGMO_BROWSE_FIRST
Get A (look foMQRC_NONE, handle 1)
Get Options = MQGMO_BROWSE_FIRST | MQGMO_LOCK
Get A (look foMQRC_NONE, handle 2)
Get Options = MQGMO_BROWSE_MSG_UNDER_CURSOR
Get (look for MQRC_NO_MSG_UNDER_CURSOR, handle 1)
Get Options = MQGMO_BROWSE_NEXT
Get B (look foMQRC_NONE, handle 2)
Get Options = MQGMO_BROWSE_MSG_UNDER_CURSOR
Get A (look foMQRC_NONE, handle 1)
```

Case 213

```
Open Options = MQOO_BROWSE | MQOO_OUTPUT
Requires 1 queue and 2 handles.
tx_begin
Put one message.
tx_commit
Get Options = MQGMO_BROWSE_FIRST | MQGMO_LOCK
Get (look for MQRC_NONE, handle 1)
Get (look for MQRC_NONE, handle 1)
Get Options = MQGMO_BROWSE_FIRST
Get (look for MQRC_NO_MSG_AVAILABLE, Handle 2)
Get Options = MQGMO_BROWSE_MSG_UNDER_CURSOR
Get (look for MQRC_NONE, handle 1)
Get Options = MQGMO_BROWSE_FIRST
Get (look for MQRC_NONE, handle 2)
```

Case 214

```
Open Options = MQOO_BROWSE | MQOO_OUTPUT
tx_begin
Put one message.
tx_commit
Get Options = MQGMO_BROWSE_FIRST | MQGMO_LOCK
Get (look for MQRC_NONE)
ReOpen the queue
Get Options = MQGMO_BROWSE_FIRST
Get (look for MQRC_NONE)
```

Case 215

```
Open Options = MQOO_BROWSE | MQOO_OUTPUT
tx_begin
Put one message
tx_commit
Get Options = MQGMO_BROWSE_FIRST | MQGMO_LOCK
Get (look for MQRC_NONE)
Disconnect, Connect, Open.
Get Options = MQGMO_BROWSE_FIRST
Get (look for MQRC_NONE)
```

Case 216

```
Open Options = MQOO_BROWSE | MQOO_OUTPUT | MQOO_INPUT_AS_Q_DEF
Requires 1 queue and 2 handles.
tx_begin
```



```
Put one message.
tx_commit
Get Options = MQGMO_BROWSE_MSG_UNDER_CURSOR
Get (look for MQRC_NO_MSG_UNDER_CURSOR, handle 1)
Get Options = MQGMO_BROWSE_FIRST
Get (look for MQRC_NONE, handle 1)
Get Options = MQGMO_NONE
Get (look for MQRC_NONE, handle 2)
Get Options = MQGMO_BROWSE_MSG_UNDER_CURSOR
Get (look for MQRC_NO_MSG_UNDER_CURSOR, handle 1)
Get Options = MQGMO_UNLOCK
Get (look for MQRC_NO_MSG_LOCKED, handle 1)
```

Case 217

```
Open Options = MQOO_BROWSE | MQOO_OUTPUT
tx_begin
Put one message.
tx_commit
Get Options = MQGMO_BROWSE_FIRST
Get (look for MQRC_NONE)
Get Options = MQGMO_UNLOCK
Get (look for MQRC_NO_MSG_LOCKED)
```

Case 218

```
Open Options = MQOO_BROWSE | MQOO_OUTPUT | MQOO_INPUT_AS_Q_DEF
tx_begin
Put one message
tx_commit
Get Options = MQGMO_BROWSE_FIRST | MQGMO_LOCK
Get (look for MQRC_NONE)
Get Options = MQGMO_NONE
Get (look for MQRC_NONE)
Get Options = MQGMO_UNLOCK
Get (look for MQRC_NO_MSG_LOCKED)
```

Case 219

```
Open Options = MQOO_BROWSE | MQOO_OUTPUT | MQOO_INPUT_AS_Q_DEF
tx_begin
Put message A
Put message B
tx_commit
Get Options = MQGMO_BROWSE_FIRST
Get A (look foMQRC_NONE)
Get Options = MQGMO_NONE
Get A (look foMQRC_NONE)
Get Options = MQGMO_BROWSE_NEXT
Get B; (look forMQRC_NONE)
```

Case 220

```
Open Options = MQOO_BROWSE | MQOO_OUTPUT
Requires 1 queue and 2 handles.
tx_begin
Put one message
tx_commit
Get Options = MQGMO_BROWSE_FIRST | MQGMO_LOCK
Get (look for MQRC_NONE, handle 1)
Get Options = MQGMO_BROWSE_NEXT
Get (look for MQRC_NO_MSG_AVAILABLE, handle 1)
```

```
Get Options = MQGMO_BROWSE_FIRST
Get (look for MQRC_NONE, handle 2)
```

Case 221

```
Open Options = MQOO_BROWSE | MQOO_OUTPUT | MQOO_INPUT_AS_Q_DEF
Requires 1 queue and 2 handles.
tx_begin
Put one message
tx_commit
Get Options = MQGMO_BROWSE_FIRST | MQGMO_LOCK
Get (look for MQRC_NONE, handle 1)
Get Options = MQGMO_NONE
Get (look for MQRC_NO_MSG_AVAILABLE,Handle 2)
Get Options = MQGMO_UNLOCK
Get (look for MQRC_NONE, handle 1)
Get Options = MQGMO_NONE
Get (look for MQRC_NONE, handle 2)
```

Case 222

```
Open Options = MQOO_BROWSE | MQOO_OUTPUT
tx_begin
Put message A
tx_commit
Get Options = MQGMO_BROWSE_FIRST
Get A (look foMQRC_NONE)
Get Options = MQGMO_BROWSE_NEXT
Get (look for MQRC_NO_MSG_AVAILABLE)
tx_begin
Put message B
tx_commit
Get B (look foMQRC_NONE)
```

Case 223

```
Requires 1 queue and 2 handles
Open Options:
    handle 1 - MQOO_BROWSE | MQOO_OUTPUT | MQOO_INPUT_AS_Q_DEF
    handle 2 - MQOO_INPUT_AS_Q_DEF
tx_begin
Put one message
tx_commit
Get Options = MQGMO_BROWSE_FIRST | MQGMO_LOCK
Get (look for MQRC_NONE, handle 1)
Get Options = MQGMO_NONE
Get (look for MQRC_NO_MSG_AVAILABLE, handle 2)
Get (look for MQRC_NONE, handle 1)
Get Options = MQGMO_BROWSE_MSG_UNDER_CURSOR
Get (look for MQRC_NO_MSG_UNDER_CURSOR)
```

Case 224

```
Open Options = MQOO_BROWSE | MQOO_OUTPUT
tx_begin
Put one message
tx_commit
Get Options = MQGMO_BROWSE_FIRST | MQGMO_LOCK
Get (look for MQRC_NONE)
Get Options = MQGMO_UNLOCK
Get (look for MQRC_NONE)
```



```
Get Options = MQGMO_BROWSE_MSG_UNDER_CURSOR
Get (look for MQRC_NONE)
Get (look for MQRC_NONE)
```

Case 225

```
Open Options = MQOO_OUTPUT | MQOO_BROWSE | MQOO_INPUT_AS_Q_DEF
tx_begin
Put one message
tx_commit
tx_begin
Get Options = MQGMO_BROWSE_FIRST
Get (look for MQRC_NONE)
Get Options = MQGMO_MSG_UNDER_CURSOR
Get (look for MQRC_NONE)
tx_commit
Get Options = MQGMO_BROWSE_MSG_UNDER_CURSOR
Get (look for MQRC_NO_MSG_UNDER_CURSOR)
```

Note: Please see qmqz06.cpp for details on Case 222 - Case 253.