

Oracle® Tuxedo Message Queue (OTMQ)

Reference Guide

12c Release 1 (12.1.1)

June 2012

ORACLE®

Copyright © 2012 Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

1. Oracle Tuxedo Message Queue Function Reference

tpdequeue()	1-2
tpenqueue()	1-11
tpqattach()	1-22
tpqdetach()	1-27
tpqbind()	1-30
tpqlocate()	1-34
tpenqplus()	1-38
tpdeqplus()	1-47
tpqpublish()	1-55
tpqsubscribe()	1-58
tpqunsubscribe()	1-63
tpqconfirmmsg()	1-64
tpqsetselect()	1-67
tpqcancelselect()	1-74
tpqreadjrn()	1-76
tpqshowpending()	1-79
tpqgetmsga()	1-81
tpqcancelget()	1-84
tpqerrno()	1-87
tpqexit()	1-87
tpqsterror()	1-88

2. Oracle Tuxedo Message Queue Command Reference

buildqclient	2-1
buildqserver	2-4
ConvertQSPACE	2-7

tmqadmin.....	2-8
---------------	-----

3. Oracle Tuxedo Message Queue UBB Server Reference

TuxMsgQLD.....	3-1
TuxMsgQ.....	3-4
TuxMQFWD.....	3-6
TMQ_NA.....	3-7
TMS_TMQM.....	3-8
TMQ_EVT.....	3-8
TMQFORWARDPLUS.....	3-9

Oracle Tuxedo Message Queue Function Reference

Table 1-1 Oracle Tuxedo Message Queue Functions

Name	Description
<code>tpdequeue()</code>	Routine to dequeue a message from a queue.
<code>tpenqueue()</code>	Routine to enqueue a message.
<code>tpqattach()</code>	Connects an application program to the OTMQ message queuing space by attaching it to a message queue.
<code>tpqdetach()</code>	Detaches a selected message queue or all of the application's message queues from the message queuing qspace.
<code>tpqbind()</code>	Dynamically associates a queue name to a queue reference at run-time.
<code>tpqlocate()</code>	Locates the queue name for the specified queue name or queue alias.
<code>tpenqplus()</code>	Sends a message to a target queue in target qspace using a set of standard OTMQ delivery modes
<code>tpdeqplus()</code>	Retrieves the next available message from a selected queue and moves it to the location specified in the data argument.
<code>tpqpublish()</code>	Used to publish a topic data.
<code>tpqsubscribe()</code>	Used to subscribe to a topic.

Table 1-1 Oracle Tuxedo Message Queue Functions (Continued)

Name	Description
<code>tpqunsubscribe()</code>	Used to remove a subscription.
<code>tpqconfirmmsg()</code>	Confirms receipt of a message that requires explicit confirmation.
<code>tpqsetselect()</code>	Allows application developers to define complex selection criteria for message reception.
<code>tpqcancelselect()</code>	Releases the selection array and index handle associated with a previously generated selection mask.
<code>tpqreadjrn()</code>	Reads a message from an OTMQ local group journal.
<code>tpqshowpending()</code>	Requests the number of pending messages for a list of selected queues.
<code>tpqgetmsga()</code>	Requests asynchronous notification of a message arrival.
<code>tpqcancelget()</code>	Cancels all pending <code>tpqgetmsga</code> requests that match the value specified in the <code>sel_filter</code> argument.
<code>tpqerrno()</code>	Gets the <code>errno</code> of OTMQ system call.
<code>tpqexit()</code>	Terminates all attachments between the application and the OTMQ queue service.
<code>tpqstrerror()</code>	Gets Oracle Tuxedo Message Queue error message string details.

tpdequeue()

Name

`tpdequeue()`—Routine to dequeue a message from a queue.

Synopsis

```
#include <atmi.h>
#include <tmqentry.h>
#include <tmqreturn.h>
int tpdequeue(char *qspace, char *qname, TPQCTL *ctl, char **data, long
*len, long flags)
```


Description

`tpdequeue()` takes a message for processing from the queue named by *qname* in the *qspace* queue space.

By default, the message at the top of the queue is dequeued. The order of messages on the queue is defined when the queue is created. The application can request a particular message for dequeuing by specifying its message identifier or correlation identifier using the *ctl* parameter. *ctl* flags can also be used to indicate that the application wants to wait for a message, in the case when a message is not currently available. It is possible to use the *ctl* parameter to look at a message without removing it from the queue or changing its relative position on the queue. See the section below describing this parameter.

data is the address of a pointer to the buffer into which a message is read, and *len* points to the length of that message. **data* must point to a buffer originally allocated by `tpalloc()`. If a message is larger than the buffer passed to `tpdequeue`, the buffer is increased in size to accommodate the message. To determine whether a message buffer changed in size, compare its (total) size before `tpdequeue()` was issued with **len*. If **len* is larger, then the buffer has grown; otherwise, the buffer has not changed size. Note that **data* may change for reasons other than the buffer's size increased. If **len* is 0 upon return, then the message dequeued has no data portion and neither **data* nor the buffer it points to were modified. It is an error for **data* or *len* to be NULL.

The message is dequeued in transaction mode if the caller is in transaction mode and the `TPNOTRAN` flag is not set. This has the effect that if `tpdequeue()` returns successfully and the caller's transaction is committed successfully, then the message is removed from the queue. If the caller's transaction is rolled back either explicitly or as the result of a transaction timeout or some communication error, then the message will be left on the queue (that is, the removal of the message from the queue is also rolled back). It is not possible to enqueue and dequeue the same message within the same transaction.

The message is not dequeued in transaction mode if either the caller is not in transaction mode, or the `TPNOTRAN` flag is set. When not in transaction mode, if a communication error or a timeout occurs, the application will not know whether or not the message was successfully dequeued and the message may be lost.

The following is a list of valid *flags*:

TPNOTRAN

If the caller is in transaction mode and this flag is set, the message is not dequeued within the caller's transaction. A caller in transaction mode that sets this flag is still subject to the

transaction timeout (and no other) when dequeuing the message. If message dequeuing fails, the caller's transaction is not affected.

TPNOBLOCK

The message is not dequeued if a blocking condition exists. If this flag is set and a blocking condition exists such as the internal buffers into which the message is transferred are full, the call fails and `tperrno` is set to `TPEBLOCK`. If this flag is set and a blocking condition exists because the target queue is opened *exclusively* by another application, the call fails, `tperrno` is set to `TPEDIAGNOSTIC`, and the diagnostic field of the `TPQCTL` structure is set to `QMESHAPE`. In the latter case, the other application, which is based on a Oracle product other than the Oracle Tuxedo ATMI system, opened the queue for exclusive read and/or write using the Queuing Services API (QSAPI).

When `TPNOBLOCK` is not set and a blocking condition exists, the caller blocks until the condition subsides or a timeout occurs (either transaction or blocking timeout). This blocking condition does not include blocking on the queue itself if the `TPQWAIT` option in `flags` (of the `TPQCTL` structure) is specified.

TPNOTIME

Setting this flag signifies that the caller is willing to block indefinitely and wants to be immune to blocking timeouts. Transaction timeouts may still occur.

TPNOCHANGE

When this flag is set, the type of the buffer pointed to by `*data` is not allowed to change. By default, if a buffer is received that differs in type from the buffer pointed to by `*data`, then `*data`'s buffer type changes to the received buffer's type so long as the receiver recognizes the incoming buffer type. That is, the type and subtype of the dequeued message must match the type and subtype of the buffer pointed to by `*data`.

TPSIGRSTRT

Setting this flag indicates that any underlying system calls that are interrupted by a signal should be reissued. When this flag is not set and a signal interrupts a system call, the call fails and sets `tperrno` to `TPGOTSIG`.

If `tpdequeue()` returns successfully, the application can retrieve additional information about the message using the `ctl` data structure. The information may include the message identifier for the dequeued message; a correlation identifier that should accompany any reply or failure message so that the originator can correlate the message with the original request; the quality of service the message was delivered with, the quality of service any replies to the message should be delivered with; the name of a reply queue if a reply is desired; and the name of the failure queue on which the application can queue information regarding failure to dequeue the message. These are described below.

In a multithreaded application, a thread in the `TPINVALIDCONTEXT` state is not allowed to issue a call to `tpdequeue()`.

Control Parameter

The `TPQCTL` structure is used by the application program to pass and retrieve parameters associated with dequeuing the message. The `flags` element of `TPQCTL` is used to indicate what other elements in the structure are valid.

On input to `tpdequeue()`, the following elements may be set in the `TPQCTL` structure:

```
long flags;           /* indicates which of the values
                    * are set */
char msgid[32];      /* ID of message to dequeue */
char corrid[32];     /* correlation identifier of
                    * message to dequeue */
```

The following is a list of valid bits for the `flags` parameter controlling input information for `tpdequeue()`:

TPNOFLAGS

No flags are set. No information is taken from the control structure.

TPQGETBYMSGID

Setting this flag requests that the message with the message identifier specified by `ctl->msgid` be dequeued. The message identifier may be acquired by a prior call to `tpenqueue()`. Note that a message identifier changes if the message has moved from one queue to another. Note also that the entire 32 bytes of the message identifier value are significant, so the value specified by `ctl->msgid` must be completely initialized (for example, padded with NULL characters).

TPQGETBYCORRID

Setting this flag requests that the message with the correlation identifier specified by `ctl->corrid` be dequeued. The correlation identifier is specified by the application when enqueueing the message with `tpenqueue()`. Note that the entire 32 bytes of the correlation identifier value are significant, so the value specified by `ctl->corrid` must be completely initialized (for example, padded with NULL characters).

TPQWAIT

Setting this flag indicates that an error should not be returned if the queue is empty. Instead, the process should wait until a message is available. If `TPQWAIT` is set in conjunction with `TPQGETBYMSGID` or `TPQGETBYCORRID`, it indicates that an error should not be returned if no message with the specified message identifier or correlation identifier is present in the queue. Instead, the process should wait until a message meeting the

criteria is available. The process is still subject to the caller's transaction timeout, or, when not in transaction mode, the process is subject to the timeout specified on the `TMQUEUE` process by the `-t` option.

If a message matching the desired criteria is not immediately available and the configured action resources are exhausted, `tpdequeue()` returns `-1`, `tperrno` is set to `TPEDIAGNOSTIC`, and the diagnostic field of the `TPQCTL` structure is set to `QMESYSTEM`.

Note that each `tpdequeue()` request specifying the `TPQWAIT` control parameter requires that a queue manager (`TMQUEUE`) action object be available if a message satisfying the condition is not immediately available. If an action object is not available, the `tpdequeue()` request fails. The number of available queue manager actions are specified when a queue space is created or modified. When a waiting dequeue request completes, the associated action object associated is made available for another request.

TPQPEEK

If this flag is set, the specified message is read but is not removed from the queue. This flag implies the `TPNOTRAN` flag has been set for the `tpdequeue()` operation. That is, non-destructive dequeuing is non-transactional. Note that it is not possible to read messages enqueued or dequeued within a transaction before the transaction completes.

When a thread is non-destructively dequeuing a message using `TPQPEEK`, the message may not be seen by other non-blocking dequeuers for the brief time the system is processing the non-destructive dequeue request. This includes dequeuers using specific selection criteria (such as message identifier and correlation identifier) that are looking for the message currently being non-destructively dequeued.

On output from `tpdequeue()`, the following elements may be set in the `TPQCTL` structure:

```
long flags;           /* indicates which of the values
                     * should be set */

long priority;       /* enqueue priority */
char msgid[32];      /* ID of message dequeued */
char corrid[32];     /* correlation identifier used to
                     * identify the message */

long delivery_qos;   /* delivery quality of service */
long reply_qos;      /* reply message quality of service */
char replyqueue[16]; /* queue name for reply */
char failurequeue[16]; /* queue name for failure */
long diagnostic;     /* reason for failure */
long appkey;         /* application authentication client
```

```

                                * key */
long urcode;                    /* user-return code */
CLIENTID cltid;                /* client identifier for originating
                                * client */

```

The following is a list of valid bits for the *flags* parameter controlling output information from `tpdequeue()`. For any of these bits, if the flag bit is turned on when `tpdequeue()` is called, the associated element in the structure is populated with the value provided when the message was queued, and the bit remains set. If a value is not available or the bit is not set when `tpdequeue()` is called, `tpdequeue()` completes with the flag turned off.

TPQPRIORITY

If this flag is set, the call to `tpdequeue()` is successful, and the message was queued with an explicit priority, then the priority is stored in `ctl->priority`. The priority is in the range 1 to 100, inclusive, and the higher the number, the higher the priority (that is, a message with a higher number is dequeued before a message with a lower number). For queues not ordered by priority, the value is informational.

If no priority was explicitly specified when the message was queued and the call to `tpdequeue()` is successful, the priority for the message is 50.

TPQMSGID

If this flag is set and the call to `tpdequeue()` is successful, the message identifier is stored in `ctl->msgid`. The entire 32 bytes of the message identifier value are significant.

TPQCORRID

If this flag is set, the call to `tpdequeue()` is successful, and the message was queued with a correlation identifier, then the correlation identifier is stored in `ctl->corrid`. The entire 32 bytes of the correlation identifier value are significant. Any Oracle Tuxedo ATMI /Q provided reply to a message has the correlation identifier of the original request message.

TPQDELIVERYQOS

If this flag is set, the call to `tpdequeue()` is successful, and the message was queued with a delivery quality of service, then the flag—`TPQQOSDEFAULTPERSIST`, `TPQQOSPERSISTENT`, or `TPQQOSNONPERSISTENT`—is stored in `ctl->delivery_qos`. If no delivery quality of service was explicitly specified when the message was queued, the default delivery policy of the target queue dictates the delivery quality of service for the message.

TPQREPLYQOS

If this flag is set, the call to `tpdequeue()` is successful, and the message was queued with a reply quality of service, then the flag—`TPQQOSDEFAULTPERSIST`, `TPQQOSPERSISTENT`,

or `TPQQOSNONPERSISTENT`—is stored in `ctl->reply_qos`. If no reply quality of service was explicitly specified when the message was queued, the default delivery policy of the `ctl->replyqueue` queue dictates the delivery quality of service for any reply.

Note that the default delivery policy is determined when the reply to a message is enqueued. That is, if the default delivery policy of the reply queue is modified between the time that the original message is enqueued and the reply to the message is enqueued, the policy used is the one in effect when the reply is finally enqueued.

TPQREPLYQ

If this flag is set, the call to `tpdequeue()` is successful, and the message was queued with a reply queue, then the name of the reply queue is stored in `ctl->replyqueue`. Any reply to the message should go to the named reply queue within the same queue space as the request message.

TPQFAILUREQ

If this flag is set, the call to `tpdequeue()` is successful, and the message was queued with a failure queue, then the name of the failure queue is stored in `ctl->failurequeue`. Any failure message should go to the named failure queue within the same queue space as the request message.

The following remaining bits for the `flags` parameter are cleared (set to zero) when `tpdequeue()` is called: `TPQTOP`, `TPQBEFOREMSGID`, `TPQTIME_ABS`, `TPQTIME_REL`, `TPQEXPTIME_ABS`, `TPQEXPTIME_REL`, and `TPQEXPTIME_NONE`. These bits are valid bits for the `flags` parameter controlling input information for `tpenqueue()`.

If the call to `tpdequeue()` failed and `tperrno` is set to `TPEDIAGNOSTIC`, a value indicating the reason for failure is returned in `ctl->diagnostic`. The possible values are defined below in the Diagnostics section.

Additionally on output, if the call to `tpdequeue()` is successful, `ctl->appkey` is set to the application authentication key, `ctl->cltid` is set to the identifier for the client originating the request, and `ctl->urcode` is set to the user-return code value that was set when the message was enqueued.

If the `ctl` parameter is `NULL`, the input flags are considered to be `TPNOFLAGS`, and no output information is made available to the application program.

Return Values

Upon failure, `tpdequeue()` returns -1 and sets `tperrno` to indicate the error condition.

Errors

Upon failure, `tpdequeue()` sets `tperrno` to one of the following values. (Unless otherwise noted, failure does not affect the caller's transaction, if one exists.)

[TPEINVAL]

Invalid arguments were given (for example, `qname` is NULL, `data` does not point to space allocated with `tpalloc()` or `flags` are invalid).

[TPENOENT]

Cannot access the `qspace` because it is not available (that is, the associated [TuxMsgQ](#) server is not available), or cannot start a global transaction due to the lack of entries in the Global Transaction Table (GTT).

[TPEOTYPE]

Either the type and subtype of the dequeued message are not known to the caller; or, `TPNOCHANGE` was set in `flags` and the type and subtype of `*data` do not match the type and subtype of the dequeued message. In either case, `*data`, its contents, and `*len` are *not* changed. When the call is made in transaction mode and this error occurs, the transaction is marked abort-only, and the message remains on the queue.

[TPETIME]

This error code indicates that either a timeout has occurred or `tpdequeue()` has been attempted, in spite of the fact that the current transaction is already marked rollback only.

If the caller is in transaction mode, then either the transaction is already rollback only or a transaction timeout has occurred. The transaction is marked abort-only. If the caller is not in transaction mode, a blocking timeout has occurred. (A blocking timeout cannot occur if `TPNOBLOCK` and/or `TPNOTIME` is specified.) In either case, no changes are made to `*data`, its contents, or `*len`.

If a transaction timeout has occurred, then, with one exception, any attempts to perform further conversational work, send new requests, or receive outstanding replies will fail with `TPETIME` until the transaction has been aborted. The exception is a request that does not block, expects no reply, and is not sent on behalf of the caller's transaction (that is, `tpacall()` with `TPNOTRAN`, `TPNOBLOCK`, and `TPNOREPLY` set).

When a service fails inside a transaction, the transaction is put into the `TX_ROLLBACK_ONLY` state. This state is treated, for most purposes, as though it were equivalent to a timeout. All further ATMI calls for this transaction (with the exception of those issued in the circumstances described in the previous paragraph) will fail with `TPETIME`.

[TPEBLOCK]

A blocking condition exists and TPNOBLOCK was specified.

[TPGOTSIG]

A signal was received and TPSIGRSTRT was not specified.

[TPEPROTO]

tpdequeue () was called improperly. There is no effect on the queue or the transaction.

[TPESYSTEM]

An Oracle Tuxedo system error has occurred. The exact nature of the error is written to a log file. There is no effect on the queue.

[TPEOS]

An operating system error has occurred. There is no effect on the queue.

[TPEDIAGNOSTIC]

Dequeuing a message from the specified queue failed. The reason for failure can be determined by the diagnostic value returned via *ctl* structure.

Diagnostic

The following diagnostic values are returned during the dequeuing of a message:

[QMEINVAL]

An invalid flag value was specified.

[QMEBADRMID]

An invalid resource manager identifier was specified.

[QMENOTOPEN]

The resource manager is not currently open.

[QMETRAN]

The call was not in transaction mode or was made with the TPNOTRAN flag set and an error occurred trying to start a transaction in which to dequeue the message. This diagnostic is not returned by queue managers from Oracle Tuxedo release 7.1 or later.

[QMEBADMSGID]

An invalid message identifier was specified for dequeuing.

[QMESYSTEM]

A system error has occurred. The exact nature of the error is written to a log file.

[QMEOS]

An operating system error has occurred.

[QMEABORTED]

The operation was aborted. When executed within a global transaction, the global transaction has been marked rollback-only. Otherwise, the queue manager aborted the operation.

[QMEPROTO]

A dequeue was done when the transaction state was not active.

[QMEBADQUEUE]

An invalid or deleted queue name was specified.

[QMENOMSG]

No message was available for dequeuing. Note that it is possible that the message exists on the queue and another application process has read the message from the queue. In this case, the message may be put back on the queue if that other process rolls back the transaction.

[QMEINUSE]

When dequeuing a message by message identifier or correlation identifier, the specified message is in use by another transaction. Otherwise, all messages currently on the queue are in use by other transactions. This diagnostic is not returned by queue managers from Oracle Tuxedo release 7.1 or later.

[QMESHARE]

When dequeuing a message from a specified queue, the specified queue is opened *exclusively* by another application. The other application is one based on an Oracle product other than the Oracle Tuxedo system that opened the queue for exclusive read and/or write using the Queuing Services API (QSAPI).

See Also

[qmadm\(1\)](#), [tpalloc\(3c\)](#), [tpenqueue\(\)](#), [APPQ_MIB\(5\)](#), [TMQUEUE\(5\)](#)

tpenqueue()**Name**

tpenqueue()—Routine to enqueue a message.

Synopsis

```
#include <atmi.h>
#include <tmqentry.h>
#include <tmqreturn.h>
int tpenqueue(char *qspace, char *qname, TPQCTL *ctl, char *data, long len,
long flags)
```

Description

`tpenqueue()` stores a message on the queue named by *qname* in the *qspace* queue space. A queue space is a collection of queues, one of which must be *qname*.

When the message is intended for an Oracle Tuxedo ATMI system server, the *qname* matches the name of a service provided by the server. The system provided server, [TMQFORWARDPLUS](#), provides a default mechanism for dequeuing messages from the queue and forwarding them to servers that provide a service matching the queue name. If the originator expects a reply, then the reply to the forwarded service request is stored on the originator's queue, unless otherwise specified. The originator will dequeue the reply message at a subsequent time. Queues can also be used for a reliable message transfer mechanism between any pair of Oracle Tuxedo ATMI system processes (clients and/or servers). In this case, the queue name does not match a service name but some agreed upon name for transferring the message.

If *data* is non-NULL, it must point to a buffer previously allocated by `tpalloc()` and *len* should specify the amount of data in the buffer that should be queued. Note that if *data* points to a buffer of a type that does not require a length to be specified (for example, an FML fielded buffer), then *len* is ignored. If *data* is NULL, *len* is ignored and a message is queued with no data portion.

The message is queued at the priority defined for *qspace* unless overridden by a previous call to `tpsprio()`.

If the caller is within a transaction and the `TPNOTRAN` flag is not set, the message is queued in transaction mode. This has the effect that if `tpenqueue()` returns successfully and the caller's transaction is committed successfully, then the message is guaranteed to be available subsequent to the transaction completing. If the caller's transaction is rolled back either explicitly or as the result of a transaction timeout or some communication error, then the message will be removed from the queue (that is, the placing of the message on the queue is also rolled back). It is not possible to enqueue then dequeue the same message within the same transaction.

The message is not queued in transaction mode if either the caller is not in transaction mode, or the `TPNOTRAN` flag is set. Once `tpenqueue()` returns successfully, the submitted message is

guaranteed to be in the queue. When not in transaction mode, if a communication error or a timeout occurs, the application will not know whether or not the message was successfully stored on the queue.

The order in which messages are placed on the queue is controlled by the application via *ctl* data structure as described below; the default queue ordering is set when the queue is created.

The following is a list of valid *flags*:

TPNOTRAN

If the caller is in transaction mode and this flag is set, the message is not queued within the caller's transaction. A caller in transaction mode that sets this flag is still subject to the transaction timeout (and no other) when queuing the message. If message queuing fails, the caller's transaction is not affected.

TPNOBLOCK

The message is not enqueued if a blocking condition exists. If this flag is set and a blocking condition exists such as the internal buffers into which the message is transferred are full, the call fails and `tperrno` is set to `TPEBLOCK`. If this flag is set and a blocking condition exists because the target queue is opened *exclusively* by another application, the call fails, `tperrno` is set to `TPEDIAGNOSTIC`, and the diagnostic field of the `TPQCTL` structure is set to `QMESHARE`. In the latter case, the other application, which is based on an Oracle product other than the Oracle Tuxedo ATMI system, opened the queue for exclusive read and/or write using the Queuing Services API (QSAPI).

When `TPNOBLOCK` is not set and a blocking condition exists, the caller blocks until the condition subsides or a timeout occurs (either transaction or blocking timeout). If a timeout occurs, the call fails and `tperrno` is set to `TPETIME`.

TPNOTIME

Setting this flag signifies that the caller is willing to block indefinitely and wants to be immune to blocking timeouts. Transaction timeouts may still occur.

TPSIGRSTRT

If this flag is set and a signal interrupts any underlying system calls, the interrupted system call is reissued. If `TPSIGRSTRT` is not set and a signal interrupts a system call, `tpenqueue()` fails and `tperrno` is set to `TPGOTSIG`.

Additional information about queuing the message can be specified via *ctl* data structure. This information includes values to override the default queue ordering placing the message at the top of the queue or before an enqueued message; an absolute or relative time after which a queued message is made available; an absolute or relative time when a message expires and is removed from the queue; the quality of service for delivering the message; the quality of service that any

replies to the message should use; a correlation identifier that aids in correlating a reply or failure message with the queued message; the name of a queue to which a reply should be enqueued; and the name of a queue to which any failure message should be enqueued.

In a multithreaded application, a thread in the `TPINVALIDCONTEXT` state is not allowed to issue a call to `tpenqueue()`.

Control Parameter

The `TPQCTL` structure is used by the application program to pass and retrieve parameters associated with enqueueing the message. The `flags` element of `TPQCTL` is used to indicate what other elements in the structure are valid.

On input to `tpenqueue()`, the following elements may be set in the `TPQCTL` structure:

```

long flags;           /* indicates which of the values
                     * are set */
long deq_time;       /* absolute/relative for dequeuing */
long priority;       /* enqueue priority */
long exp_time        /* expiration time */
long delivery_qos    /* delivery quality of service */
long reply_qos       /* reply quality of service */
long urcode;         /* user-return code */
char msgid[32];      /* ID of message before which to queue
                     * request */
char corrid[32];     /* correlation identifier used to
                     * identify the msg */
char replyqueue[16]; /* queue name for reply message */
char failurequeue[16]; /* queue name for failure message */

```

The following is a list of valid bits for the `flags` parameter controlling input information for `tpenqueue()`:

TPNOFLAGS

No flags or values are set. No information is taken from the control structure.

TPQTOP

Setting this flag indicates that the queue ordering be overridden and the message placed at the top of the queue. This request may not be granted depending on whether or not the queue was configured to allow overriding the queue ordering. `TPQTOP` and `TPQBEFOREMSGID` are mutually exclusive flags.

TPQBEFOREMSGID

Setting this flag indicates that the queue ordering be overridden and the message placed in the queue before the message identified by *ctl->msgid*. This request may not be granted depending on whether or not the queue was configured to allow overriding the queue ordering. **TPQTOP** and **TPQBEFOREMSGID** are mutually exclusive flags. Note that the entire 32 bytes of the message identifier value are significant, so the value identified by *ctl->msgid* must be completely initialized (for example, padded with NULL characters).

TPQTIME_ABS

If this flag is set, the message is made available after the time specified by *ctl->deq_time*. The *deq_time* is an absolute time value as generated by `time(2)`, `mktime(3)`, or `gp_mktime(3c)` (the number of seconds since 00:00:00 Universal Coordinated Time—UTC, January 1, 1970). **TPQTIME_ABS** and **TPQTIME_REL** are mutually exclusive flags. The absolute time is determined by the clock on the machine where the queue manager process resides.

TPQTIME_REL

If this flag is set, the message is made available after a time relative to the completion of the enqueueing operation. *ctl->deq_time* specifies the number of seconds to delay after the enqueueing completes before the submitted message should be available.

TPQTIME_ABS and **TPQTIME_REL** are mutually exclusive flags.

TPQPRIORITY

If this flag is set, the priority at which the message should be enqueued is stored in *ctl->priority*. The priority must be in the range 1 to 100, inclusive. The higher the number, the higher the priority (that is, a message with a higher number is dequeued before a message with a lower number). For queues not ordered by priority, this value is informational.

If this flag is not set, the priority for the message is 50 by default.

TPQCORRID

If this flag is set, the correlation identifier value specified in *ctl->corrid* is available when a message is dequeued with `tpdequeue()`. This identifier accompanies any reply or failure message that is queued so that an application can correlate a reply with a particular request. Note that the entire 32 bytes of the correlation identifier value are significant, so the value specified in *ctl->corrid* must be completely initialized (for example, padded with NULL characters).

TPQREPLYQ

If this flag is set, a reply queue named in *ctl->replyqueue* is associated with the queued message. Any reply to the message will be queued to the named queue within the same

queue space as the request message. This string must be NULL terminated (maximum 15 characters in length).

TPQFAILUREQ

If this flag is set, a failure queue named in the *ctl->failurequeue* is associated with the queued message. If (1) the enqueued message is processed by `TMQFORWARD()`, (2) `TMQFORWARD` was started with the `-d` option, and (3) the service fails and returns a non-NULL reply, a failure message consisting of the reply and its associated `tpurcode` is enqueued to the named queue within the same queue space as the original request message. This string must be NULL-terminated (maximum 15 characters in length).

TPQDELIVERYQOS, TPQREPLYQOS

If the `TPQDELIVERYQOS` flag is set, the flags specified by *ctl->delivery_qos* control the quality of service for delivery of the message. In this case, one of three mutually exclusive flags—`TPQQOSDEFAULTPERSIST`, `TPQQOSPERSISTENT`, or `TPQQOSNONPERSISTENT`—must be set in *ctl->delivery_qos*. If `TPQDELIVERYQOS` is not set, the default delivery policy of the target queue dictates the delivery quality of service for the message.

If the `TPQREPLYQOS` flag is set, the flags specified by *ctl->reply_qos* control the quality of service for any reply to the message. In this case, one of three mutually exclusive flags—`TPQQOSDEFAULTPERSIST`, `TPQQOSPERSISTENT`, or `TPQQOSNONPERSISTENT`—must be set in *ctl->reply_qos*. The `TPQREPLYQOS` flag is used when a reply is returned from messages processed by `TMQFORWARD`. Applications not using `TMQFORWARD` to invoke services may use the `TPQREPLYQOS` flag as a hint for their own reply mechanism.

If `TPQREPLYQOS` is not set, the default delivery policy of the *ctl->replyqueue* queue dictates the delivery quality of service for any reply. Note that the default delivery policy is determined when the reply to a message is enqueued. That is, if the default delivery policy of the reply queue is modified between the time that the original message is enqueued and the reply to the message is enqueued, the policy used is the one in effect when the reply is finally enqueued.

The following is the list of valid flags for *ctl->delivery_qos* and *ctl->reply_qos*:

`TPQQOSDEFAULTPERSIST`

This flag specifies that the message is to be delivered using the default delivery policy specified on the target queue.

`TPQQOSPERSISTENT`

This flag specifies that the message is to be delivered in a persistent manner using the disk-based delivery method. Setting this flag overrides the default delivery policy specified on the target queue.

TPQQOSNONPERSISTENT

This flag specifies that the message is to be delivered in a non-persistent manner using the memory-based delivery method. Specifically, the message is queued in memory until it is dequeued. Setting this flag overrides the default delivery policy specified on the target queue. If the caller is transactional, non-persistent messages are enqueued within the caller's transaction, however, non-persistent messages are lost if the system is shut down, crashes, or the IPC shared memory for the queue space is removed.

TPQEXPTIME_ABS

If this flag is set, the message has an absolute expiration time, which is the absolute time when the message will be removed from the queue.

The absolute expiration time is determined by the clock on the machine where the queue manager process resides.

The absolute expiration time is indicated by the value stored in `ctl->exp_time`. The value of `ctl->exp_time` must be set to an absolute time value generated by `time(2)`, `mktime(3C)`, or `gp_mktime(3c)` (the number of seconds since 00:00:00 Universal Coordinated Time—UTC, January 1, 1970).

If an absolute time is specified that is earlier than the time of the enqueue operation, the operation succeeds, but the message is not counted for the purpose of calculating thresholds. If the expiration time is before the message availability time, the message is not available for dequeuing unless either the availability or expiration time is changed so that the availability time is before the expiration time. In addition, these messages are removed from the queue at expiration time even if they were never available for dequeuing. If a message expires while it is within a transaction, the expiration does not cause the transaction to fail. Messages that expire while being enqueued or dequeued within a transaction are removed from the queue when the transaction ends. There is no notification that the message has expired.

TPQEXPTIME_ABS, TPQEXPTIME_REL, and TPQEXPTIME_NONE are mutually exclusive flags. If none of these flags is set, the default expiration time associated with the target queue is applied to the message.

TPQEXPTIME_REL

If this flag is set, the message has a relative expiration time, which is the number of seconds *after* the message arrives at the queue that the message is removed from the queue. The relative expiration time is indicated by the value stored in `ctl->exp_time`.

If the expiration time is before the message availability time, the message is not available for dequeuing unless either the availability or expiration time is changed so that the availability time is before the expiration time. In addition, these messages are removed

from the queue at expiration time even if they were never available for dequeuing. The expiration of a message during a transaction, does not cause the transaction to fail. Messages that expire while being enqueued or dequeued within a transaction are removed from the queue when the transaction ends. There is no acknowledgment that the message has expired.

TPQEXPTIME_ABS, TPQEXPTIME_REL, and TPQEXPTIME_NONE are mutually exclusive flags. If none of these flags is set, the default expiration time associated with the target queue is applied to the message.

TPQEXPTIME_NONE

Setting this flag indicates that the message should not expire. This flag overrides any default expiration policy associated with the target queue. A message can be removed by dequeuing it or by deleting it via an administrative interface.

TPQEXPTIME_ABS, TPQEXPTIME_REL, and TPQEXPTIME_NONE are mutually exclusive flags. If none of these flags is set, the default expiration time associated with the target queue is applied to the message.

Additionally, the *urcode* element of TPQCTL can be set with a user-return code. This value will be returned to the application that dequeues the message.

On output from `tpenqueue()`, the following elements may be set in the TPQCTL structure:

```
long flags;           /* indicates which of the values
                      * are set */
char msgid[32];      /* ID of enqueued message */
long diagnostic;     /* indicates reason for failure */
```

The following is a valid bit for the *flags* parameter controlling output information from `tpenqueue()`. If this flag is turned on when `tpenqueue()` is called, the /Q server `TMQUEUE(5)` populates the associated element in the structure with a message identifier. If this flag is turned off when `tpenqueue()` is called, `TMQUEUE()` does *not* populate the associated element in the structure with a message identifier.

TPQMSGID

If this flag is set and the call to `tpenqueue()` is successful, the message identifier is stored in `ctl->msgid`. The entire 32 bytes of the message identifier value are significant, so the value stored in `ctl->msgid` is completely initialized (for example, padded with NULL characters). The actual padding character used for initialization varies between releases of the Oracle Tuxedo ATMI /Q component.

The remaining members of the control structure are not used on input to `tpenqueue()`.

If the call to `tpenqueue()` failed and `tperrno` is set to `TPEDIAGNOSTIC`, a value indicating the reason for failure is returned in `ctl->diagnostic`. The possible values are defined below in the Diagnostics section.

If this parameter is `NULL`, the input flags are considered to be `TPNOFLAGS` and no output information is made available to the application program.

Return Values

Upon failure, `tpenqueue()` returns -1 and sets `tperrno` to indicate the error condition. Otherwise, the message has been successfully queued when `tpenqueue()` returns.

Errors

Upon failure, `tpenqueue()` sets `tperrno` to one of the following values. (Unless otherwise noted, failure does not affect the caller's transaction, if one exists.)

[TPEINVAL]

Invalid arguments were given (for example, `qspace` is `NULL`, `data` does not point to space allocated with `tpalloc()`, or `flags` are invalid).

[TPENOENT]

Cannot access the `qspace` because it is not available (that is, the associated `TMQUEUE (5)` server is not available), or cannot start a global transaction due to the lack of entries in the Global Transaction Table (GTT).

[TPETIME]

This error code indicates that either a timeout has occurred or `tpenqueue()` has been attempted, in spite of the fact that the current transaction is already marked rollback only.

If the caller is in transaction mode, then either the transaction is already rollback only or a transaction timeout has occurred. The transaction is marked abort-only. If the caller is not in transaction mode, a blocking timeout has occurred. (A blocking timeout cannot occur if `TPNOBLOCK` and/or `TPNOTIME` is specified.)

If a transaction timeout has occurred, then, with one exception, any attempts to send new requests or receive outstanding replies will fail with `TPETIME` until the transaction has been aborted. The exception is a request that does not block, expects no reply, and is not sent on behalf of the caller's transaction (that is, `tpacall()` with `TPNOTRAN`, `TPNOBLOCK`, and `TPNOREPLY` set).

When a service fails inside a transaction, the transaction is put into the `TX_ROLLBACK_ONLY` state. This state is treated, for most purposes, as though it were equivalent to a timeout. All further ATMI calls for this transaction (with the exception of

those issued in the circumstances described in the previous paragraph) will fail with `TPETIME`.

[TPEBLOCK]

A blocking condition exists and `TPNOBLOCK` was specified.

[TPGOTSIG]

A signal was received and `TPSIGRSTRT` was not specified.

[TPEPROTO]

`tpenqueue()` was called improperly.

[TPESYSTEM]

An Oracle Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[TPEOS]

An operating system error has occurred.

[TPEDIAGNOSTIC]

Enqueuing a message on the specified queue failed. The reason for failure can be determined by the diagnostic returned via `ctl`.

Diagnostic

The following diagnostic values are returned during the enqueueing of a message:

[QMEINVAL]

An invalid flag value was specified.

[QMEBADRMID]

An invalid resource manager identifier was specified.

[QMENOTOPEN]

The resource manager is not currently open.

[QMETRAN]

The call was not in transaction mode or was made with the `TPNOTRAN` flag set and an error occurred trying to start a transaction in which to enqueue the message. This diagnostic is not returned by queue managers from Oracle Tuxedo release 7.1 or later.

[QMEBADMSGID]

An invalid message identifier was specified.

[QMESYSTEM]

A system error occurred. The exact nature of the error is written to a log file.

[QMEOS]

An operating system error occurred.

[QMEABORTED]

The operation was aborted. When executed within a global transaction, the global transaction has been marked rollback-only. Otherwise, the queue manager aborted the operation.

[QMEPROTO]

An enqueue was done when the transaction state was not active.

[QMEBADQUEUE]

An invalid or deleted queue name was specified.

[QMENOSPACE]

Due to an insufficient resource, such as no space on the queue, the message with its required quality of service (persistent or non-persistent storage) was not enqueued. `QMENOSPACE` is returned when any of the following configured resources is exceeded: (1) the amount of disk (persistent) space allotted to the queue space, (2) the amount of memory (non-persistent) space allotted to the queue space, (3) the maximum number of simultaneously active transactions allowed for the queue space, (4) the maximum number of messages that the queue space can contain at any one time, (5) the maximum number of concurrent actions that the Queuing Services component can handle, or (6) the maximum number of authenticated users that may concurrently use the Queuing Services component.

[QMERELLEASE]

An attempt was made to enqueue a message to a queue manager that is from a version of the Oracle Tuxedo system that does not support a newer feature.

[QMESHARE]

When enqueueing a message from a specified queue, the specified queue is opened *exclusively* by another application. The other application is one based on an Oracle product other than the Oracle Tuxedo system that opened the queue for exclusive read and/or write using the Queuing Services API (QSAPI).

See Also

[qmadmin\(1\)](#), [gp_mktime\(3c\)](#), [tpacall\(3c\)](#), [tpalloc\(3c\)](#), [tpdequeue\(\)](#), [tpinit\(3c\)](#), [tps prio\(3c\)](#), [APPQ_MIB\(5\)](#), [TMQFORWARD\(5\)](#), [TMQUEUE\(5\)](#)

tpqattach()

Name

`tpqattach()`—Connects an application program to the OTMQ message queuing space by attaching it to a message queue.

Synopsis

```
#include <atmi.h>
#include <tmqentry.h>
#include <tmqreturn.h>
int tpqattach (qspace, queue, ctl, qattachctl, flags)
```

Description

A message queue `qspace` is a collection of message queues that reside on a system, share global memory sections and files, and are served by the same server processes. An OTMQ message queue is an area of memory or disk where messages are stored and retrieved. See the installation and configuration guide for the platform you are using to learn how to configure the OTMQ environment.

To receive OTMQ messages, an application must attach to at least one message queue. The `tpqattach` function enables an application to attach in the following ways:

- An application can attach to a queue by specifying a name. To attach by name, the message queue must be created by run `createqueue` command of `tmqadmin`. Attaching by name enables an application to attach to a specific queue, send messages to the queue, and retrieve messages sent to that queue.
- An application can attach to a queue by specifying the queue alias. To attach by name alias, the message queue must be created by running the `tmqadmin createqueue` command. Attaching by name alias enables an application to attach to a specific queue, send messages to the queue, and retrieve messages sent to that queue. In addition, attaching by name alias eliminates the need to change code or recompile if the queue name alias changes. Therefore, attaching by name alias protects applications from changes in the OTMQ environment configuration.
- An application can attach to a temporary queue. To attach to a temporary queue, the application does not have to give a specific queue name or name alias. OTMQ will assign a queue and return the name of the queue which has been assigned. Temporary queues allow an application to perform messaging without knowing configuration details of the group.

Applications can specify an attachment as primary or secondary. All applications must have a primary queue. In addition, applications can attach to one or more secondary queues. Primary queues can be configured in the queue create command as the owners of secondary queues. When an application attaches to a primary queue that is the owner of secondary queues, the application is automatically attached to the secondary queues at the same time it is attached to the primary queue.

In addition, an application can attach to a multi-resource queue. A multi-resource queue can be read by many applications and is configured as part of the group definition.

Table 1-2 lists `tpqattach()` supported arguments:

Table 1-2 tpqattach() Arguments

Argument	Data Type	Mechanism	Prototype	Access
qspace	char	reference	char*	passed
queue	char	reference	char*	passed/returned
ctl	TPQCTL	reference	TPQCTL*	passed/returned
Qattachctl	Q_ATTACH_C TL	Reference	Q_ATTACH_C TL *	passed
Flags	long	Reference	Long	passed

qspace

Supplies the queue space name for enqueue the message. The max length is 15.

queue

Supplies the name of the permanent queue to attach to the application if the `attach_mode` argument specifies attachment by queue name or queue alias. Queue names are alphanumeric strings with no embedded spaces and allow the following special characters: underscore (`_`), hyphen (`-`), and dollar sign (`$`). The max length is 127.

References to queue names are case sensitive and must match the queue name entered in the create queue command by `tmqadmin`. Some example queue names are: `QUEUE_1`, `high-priority`, and `My$Queue`.

ctl

The TPQCTL structure is used by the application program to pass and retrieve parameters associated with enqueueing the message. The TPQCTL flags element is used to indicate what other elements in the structure are valid.

On input to `tpenqplus()`, the elements shown in [Listing 1-1](#) may be set in the `TPQCTL` structure.

Listing 1-1 `tpenqplus()` Elements

```

long flags;           /* indicates which of the values are set */
long deq_time;       /* absolute/relative time for dequeuing */
long priority;       /* enqueue priority */
long diagnostic;     /* indicates reason for failure */
char msgid[TMMSGIDLEN]; /* id of message before which to queue */
char corrid[TMCCRIDLEN]; /* correlation id used to identify message */
char replyqueue[TMQNAMELEN+1]; /* queue name for reply message */
char failurequeue[TMQNAMELEN+1]; /* queue name for failure message */
CLIENTID cltid; /* client identifier for originating client */
long urcode;        /* application user-return code */
long appkey;        /* application authentication client key */
long delivery_qos; /* delivery quality of service */
long reply_qos;     /* reply message quality of service */
long exp_time;      /* expiration time */
/* new members for TMQPlus */
long block;         /* specify block mode: WF, AK, NN */
long DIP; /* specify the delivery interesting point: MEM, SAF, DQF, DEQ, ACK,
CONF */
long uma;          /* undelivered message action */
long msg_class;    /* message class */
long msg_type;     /* message type */
PSB status_block; /* message delivery control point and UMA status block */
long redeliver_count; /* the max count which the message can be redelivered */
long seq_number[2]; /* message seq number, which is decided in client side to
decrease the TMQ load */
long timeout;      /* timeout value for block enq/deq operation */
char src_qspace[TMQSNAMELEN+1]; /* the source QSpace name. */
char src_qname[TMQNAMELEN+1]; /* the source queue name. */
char tgt_qspace[TMQSNAMELEN+1]; /* the source QSpace name. */
char tgt_qname[TMQNAMELEN+1]; /* the source queue name. */
char orig_src_qspace[TMQSNAMELEN+1]; /* the original source QSpace name. */
char orig_src_qname[TMQNAMELEN+1]; /* the original source queue name. */

```

```

char orig_tgt_qspace[TMQSNAMELEN+1];/* the original target QSpace name. */
char orig_tgt_qname[TMQNAMELEN+1]; /* the original target queue name. */
char hops; /* net hops */
long opcode;
long filter_idx;
long user_tag;
long geta_idx; /* index of pending pams_get_msga requests */
long endian;
long receipt_msg_type; /*used for uma message*/

```

Qattachctl

The is used by the application program to pass parameters associated with attach the queue. The elements shown in [Listing 1-2](#) may be set in the Q_ATTACH_CTL structure:

Listing 1-2 Q_ATTACH_CTL Elements

```

TM32I attachmode; /* Supplies the mode for attaching the application
to a message queue.*/
TM32I qtype; /* Supplies the queue type for the attachment. */
TM32I * namespace_list; /* Supplies a list of name tables to search
when the attach_mode argument. */
TM32I namespace_list_len; /* Supplies the number of entries in
the name_space_list argument. */
long timeout; /* The number of OTMQ time units (1 second
intervals) to allow for the attach to complete. */

```

Flags**TPNOTRAN**

If the caller is in transaction mode and this flag is set, the message is not queued within the caller transaction. A caller in transaction mode that sets this flag is still subject to the transaction timeout (and no other), when queuing the message. If message queuing fails, the caller transaction is not affected.

TPNOBLOCK

Caller attaches to the queue if a blocking condition exists. If this flag is set and a blocking condition exists (such as the internal buffers into which the message is transferred are full), the call fails and `tperrno` is set to `TPEBLOCK`. If this flag is set and a blocking condition exists because the target queue is opened exclusively by another application, the call fails, `tperrno` is set to `TPEDIAGNOSTIC`, and the diagnostic field of the `TPQCTL` structure is set to `QMESHARE`. In the latter case, the other application, which is based on an Oracle product other than the Oracle Tuxedo ATMI system, opened the queue for exclusive read and/or write using the Queuing Services API (QSAPI).

When `TPNOBLOCK` is not set and a blocking condition exists, the caller blocks until the condition subsides or a timeout occurs (either transaction or blocking timeout). If a timeout occurs, the call fails and `tperrno` is set to `TPETIME`.

TPNOTIME

Setting this flag signifies that the caller is willing to block indefinitely and wants to be immune to blocking timeouts. Transaction timeouts may still occur.

TPSIGRSTRT

If this flag is set and a signal interrupts any underlying system calls, the interrupted system call is reissued. If `TPSIGRSTRT` is not set and a signal interrupts a system call, `tpenqueue()` fails and `tperrno` is set to `TPGOTSIG`.

Return Value(s)

[TPEINVAL]

Invalid arguments were given.

[TPENOENT]

Cannot access the qspace because it is not available

[TPETIME]

This error code indicates that either a timeout has occurred or `tpqattach()` has been attempted, in spite of the fact that the current transaction is already marked rollback only.

[TPEBLOCK]

A blocking condition exists and `TPNOBLOCK` was specified.

[TPESYSTEM]

An Oracle Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[TPEOS]

An operating system error has occurred.

[TPEDIAGNOSTIC]

Attach to the specified queue failed. The reason for failure can be determined by the diagnostic returned via ctl.

Diagnostic:

[QMEINVAL]

An invalid flag value was specified.

[QMESYSTEM]

A system error occurred. The exact nature of the error is written to a log file.

[QMEOS]

An operating system error occurred.

[QMEABORTED]

The operation was aborted. When executed within a global transaction, the global transaction has been marked rollback-only. Otherwise, the queue manager aborted the operation.

[QMEBADQUEUE]

An invalid or deleted queue name was specified.

tpqdetach()

Name

tpqdetach()—Detaches a selected message queue or all of the application message queues from the message queuing qspace.

Synopsis

```
#include <atmi.h>
#include <tmqentry.h>
#include <tmqreturn.h>
int tpqdetach ( qspace, queue, detach_opt_list, detach_opt_len, msg_flushed
, flags)
```

Description

Detaches a selected message queue or all of the application's message queues from the message queuing qspace. When an application detaches from its primary queue, this function automatically detaches all secondary queue attachments defined for the primary queue. When the last message queue has been detached, the application is automatically detached from the OTMQ message queuing qspace.

If you are using implicit confirmation with recoverable messaging, you must ensure that the last message is confirmed before:

- detaching from the queue which received the message by calling `tpqdetach`
- detaching from the OTMQ qspace by calling `tpqexit`
- exiting your application

If you do not ensure that the last message was confirmed before detaching or exiting, the message will be re-delivered when the queue is reattached. The easiest method to ensure confirmation is to save the PSB delivery status of the last message received, check it for the required confirmation status, and then exit after the message has been confirmed.

Table 1-3 lists `tpqdetach()` supported arguments.

Table 1-3 `tpqdetach()` Arguments

Argument	Data Type	Mechanism	Prototype	Access
<code>qspace</code>	char	reference	char *	passed
<code>queue</code>	char	reference	char *	passed
<code>detach_opt_list</code>	int	reference	int *	passed
<code>detach_opt_list</code>	int	reference	int	passed
<code>msg_flushed</code>	int	reference	int *	returned
<code>flags</code>	long	reference	long	passed

qspace

Supplies the queue space of the queue to be detached. This function can be used to detach primary, secondary, and multi-resource queues.

queue

Supplies the queue name to be detached.

detach_opt_list

Supplies an array of `int` values used to control how the queue is detached. The predefined constants for this argument are:

TMQ_NOFLUSH_Q

Detaches the queue without flushing the pending messages stored in memory no matter the input parameter queue. The default action is to flush pending messages in the queue before it is detached. Messages are never flushed from multi-resource queues.

TMQ_DETACH_ALL

Detaches all of the application's message queues from the message queuing qspace. Using this constant performs the same action as calling the `tpqexit` function.

TMQ_CANCEL_SEL

Cancels all selection masks that reference the queue or queues that you are detaching. If you do not select this option and you do not cancel selection masks, OTMQ invalidates all selection masks that reference the queue or queues that you are detaching. You must cancel the invalidated selection masks using the `tpqcancelget` function.

detach_opt_len

Supplies the number of `int` values in the `detach_opt_list` array. The maximum number of `int` values is 32767.

msgs_flushed

Receives the number of messages that were flushed from the queue. Message count statistics are enabled on all systems by default; therefore, it is not necessary to enable statistics on UNIX and Windows NT systems in order to properly return this value.

flags

The following is a list of valid flags:

- TPNOTRAN
- TPNOBLOCK
- TPNOTIME
- TPSIGRSTRT

Return Value(s)

Upon failure, `tpqdetach()` returns -1 and sets `tperrno` to indicate the error condition. Otherwise, the queue has been successfully detached when `tpqdetach()` returns.

Errors:

[TPEINVAL]

Invalid arguments were given.

[TPENOENT]

Cannot access the qspace because it is not available

[TPETIME]

This error code indicates that either a timeout has occurred or `tpqdetach()` has been attempted, in spite of the fact that the current transaction is already marked rollback only.

[TPEBLOCK]

A blocking condition exists and `TPNOBLOCK` was specified.

[TPGOTSIG]

A signal was received and `TPSIGRSTR` was not specified.

[TPEPROTO]

`tpqdetach()` was called improperly.

[TPESYSTEM]

An Oracle Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[TPEOS]

An operating system error has occurred.

tpqbind()

Name

`tpqbind()` —Dynamically associates a queue name to a queue reference at run-time.

Synopsis

```
#include <atmi.h>
#include <tmqentry.h>
#include <tmqreturn.h>
int tpqbind (qspace , pNameCtl , scope, timeout);
```

Description

Dynamically associates a queue name to a queue reference at run-time. This enables a server application to dynamically sign up to service a queue alias at run-time. Thus, an end user can access a service without having to be aware that its normal host computer is down and that the service is being provided from another host computer. To use `tpqbind()`, you must first invoke `tpqattach()`.

Table 1-4 lists the `tpqbind()` supported arguments:

Table 1-4 tpqbind() Arguments

Argument	Data Type	Mechanism	Prototype	Access
qspace	char	reference	char *	passed
pNameCtl	Q_NAME_CTL	reference	Q_NAME_CTL *	passed
Scope	long	reference	long *	passed
timeout	long	reference	long	passed

qspace

Identifies the queue space to be bound. The `qspace` argument must same with the `qspace` argument of `tpqattach()`.

pNameCtl

The `Q_NAME_CTL` structure is used by the application program to pass and retrieve parameters associated with bind alias to queue name

```
typedef struct {
    char          pName[ TMQALIASLEN+1 ];
    char          pGroup[ TMQSNAMELEN+1 ];
    char          pQueue[ TMQNAMELEN+1 ];
    TM32I        nFlags;
    TM32I        nType;          /* L/G */
    TM32I        type;          /* client type */
    TM32I        nOwnerPid;     /* client pid */
    CLIENTID     cltid;
    TM32I        * namespace_list; /* for using pams
interface */
    TM32I        namespace_list_len; /* for using pams
interface */ } Q_NAME_CTL;
```

pName

Alias name for bind, identifies a global queue reference or a local queue reference.

pGroup

Identifies the queue space to be bound.

pQueue

Queue name.

Note: pGroup and pQueue values specified to this argument controls whether the queue name is bound or unbound:

If the queue pGroup and pQueue are specified, this function binds it to a pName. "If 0 is specified, this function unbinds the pName from its queue name. The calling application must be bound to pName to set it back to zero."

name_space_list

The name_space_list argument also controls the cache access as follows:

To lookup a local queue reference or queue name, specify both OTMQ _TBL_GRP and OTMQ _TBL_PROC. This causes the process cache to be checked before looking into the group cache.

To lookup a global queue reference, specify OTMQ_TBL_BUS OTMQ _TBL_GRP and OTMQ _TBL_PROC. This causes the process cache to be checked. Then, the group cache is checked before looking into the global name space.

Note that to lookup all caches in the global name space before looking in the master database, specify OTMQ _TBL_BUS_LOW instead of OTMQ _TBL_BUS.

To lookup only the slower but more up-to-date caches in the global name space before looking in the master database, specify OTMQ _TBL_BUS_MEDIUM instead of OTMQ _TBL_BUS.

name_space_list_len

Supplies the number of entries in the name_space_list argument. If the name_space_list_len argument is zero, uses OTMQ_TBL_GRP as the default in the name_space_list argument.

nFlags

CORE_FLAGS_BOUND: binds the alias to the queue name

CORE_FLAGS_CACHED: cache the alias to locale share memory.

CORE_FLAGS_LOCKED: lock the alias. If it is locked, it cannot be accessed.

nOwnerPid

Not used

nType

Not usedN

cltid

Not used.

scope

Specifies the scope for pName. The identifier for scope:

NAME_SCOPE_P: scope is process

NAME_SCOPE_L: scope is local

NAME_SCOPE_G: scope is global

For TP APIs, scope and namespace_list arguments, If scope is NULL, namespace_list must not NULL too, and if scope has a data value, namespace_list will invalidate.

timeout

The number of OTMQ time units (1 second intervals) to allow for the attach to complete. If a null pointer is specified, the BLOCKTIME property of the group's attach service is used (configured in the UBBCONFIG *SERVICES section). If service-wide BLOCKTIME is not configured, system-wide BLOCKTIME specified in the UBBCONFIG *RESOURCES section is used (default to approximately 60 seconds).

Return Value(s)

Upon failure, tpqbind() returns -1 and sets tperrno to indicate the error condition. Otherwise, the queue alias has been successfully binded when tpqbind() returns.

Errors:

[TPEINVAL]

Invalid arguments were given.

[TPENOENT]

Cannot access the qspace because it is not available

[TPETIME]

This error code indicates that either a timeout has occurred or tpqbind() has been attempted, in spite of the fact that the current transaction is already marked rollback only.

[TPEPROTO]

tpqbind() was called improperly. For example, invoke tpqbind without invoke tpqattach() first.

[TPESYSTEM]

An Oracle Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[TPEOS]

An operating system error has occurred.

tpqlocate()**Name**

`tpqlocate()` —Locates the queue name for the specified queue name or queue alias.

Synopsis

```
#include <atmi.h>
#include <tmqentry.h>
#include <tmqreturn.h>
int tpqlocate (qspace, pNameCtl, req_id, resp_q, scope, wait_mode, timeout
)
```

Description

Locates the queue name for the specified queue name or queue alias. By default, this function waits for the queue name to be returned. To use `tpqlocate()`, you must first invoke `tpqattach()`.

[Table 1-5](#) lists `tpqlocate()` supported arguments:

Table 1-5 tpqlocate() Arguments

Argument	Data Type	Mechanism	Prototype	Access
<code>qspace</code>	<code>char</code>	reference	<code>char *</code>	passed
<code>pNameCtl</code>	<code>pNameCtl</code>	reference	<code>pNameCtl *</code>	passed/returned
<code>req_id</code>	<code>long</code>	reference	<code>long</code>	passed
<code>resp_q</code>	<code>char</code>	reference	<code>char *</code>	passed
<code>scope</code>	<code>long</code>	reference	<code>long</code>	passed
<code>wait_mode</code>	<code>long</code>	reference	<code>long</code>	passed
<code>timeout</code>	<code>long</code>	reference	<code>long *</code>	passed

qspace

Supplies the queue space whose queue name is requested. Must be same with the `tpqattach()` `qspace` argument.

pNameCtl

The `Q_NAME_CTL` structure is used by the application program to pass and retrieve parameters associated with bind alias to queue name

```
typedef struct {
    char          pName[TMQALIASLEN+1];
    char          pGroup[TMQSNAMELEN+1];
    char          pQueue[TMQNAMELEN+1];
    TM32I        nFlags;
    TM32I        nType;          /* L/G */
    TM32I        type;          /* client type */
    TM32I        nOwnerPid;     /* client pid */
    CLIENTID     cltid;
    TM32I        * namespace_list; /* for using pams interface */
    TM32I        namespace_list_len; /* for using pams interface */
} Q_NAME_CTL;
```

`pName`

Alias name for bind, Identifies a global queue reference or a local queue reference.

`pGroup`

`qspace`, identifies the `qspace` name of the alias bind.

`pQueue`

queue name, identifies the queue name of the alias bind.

`pGroup` and `pQueue` values specified to this argument controls whether the queue name is bound or unbound:

- If the queue `pGroup` and `pQueue` are specified, this function binds it to a `pName`.
- If 0 is specified, this function unbinds the `pName` from its queue name. The calling application must be bound to `pName` to set it back to zero.

name_space_list

The `name_space_list` argument also controls the cache access as follows:

To lookup a local queue reference or queue name, specify both `OTMQ_TBL_GRP` and `OTMQ_TBL_PROC`. This causes the process cache to be checked before looking into the qspace cache.

To lookup a global queue reference, specify `OTMQ_TBL_BUS`, `OTMQ_TBL_GRP` and `OTMQ_TBL_PROC`. This causes the process cache to be checked. Then, the qspace cache is checked before looking into the global name space.

Note: that to lookup all caches in the global name space before looking in the master database, specify `OTMQ_TBL_BUS_LOW` instead of `OTMQ_TBL_BUS`.

To lookup only the slower but more up-to-date caches in the global name space before looking in the master database, specify `OTMQ_TBL_BUS_MEDIUM` instead of `OTMQ_TBL_BUS`.

name_space_list_len

Supplies the number of entries in the `name_space_list` argument. If the `name_space_list_len` argument is zero, uses `OTMQ_TBL_GRP` as the default in the `name_space_list` argument.

nFlags

`CORE_FLAGS_BOUND`: binds the alias to the queue name

`CORE_FLAGS_CACHED`: cache the alias to locale share memory.

`CORE_FLAGS_LOCKED`: lock the alias.

nOwnerPid

Reserved for future use

nType

Reserved for future use

cltid

Reserved for future use

wait_mode

Supplies the search mode of the `tpqlocate` function. The mode indicates whether the application waits for the search completion or receives the response in an acknowledgment message. There are two predefined constants for this argument:

- `"OTMQ_WF_RESP` (default setting)-The application issues the `tpqlocate` request and waits for the queue name to be returned.
- `"OTMQ_AK_RESP`-The application issues the `tpqlocate` name and continues processing. When the search is completed, the queue name is returned to the application's primary queue in a `LOCATE_Q_REP` message. The response message can

be redirected to an alternate queue name using the `resp_q` argument, The details of `LOCATE_Q_REP` message as next.

req_id

Supplies an application-specified transaction ID to associate with the `tpqlocate` function.

resp_q

Supplies an alternate queue to use for receiving the acknowledgment message of the queue name. If no response queue is specified, the acknowledgment message is sent to the sender program primary queue.

Note: the sender program cannot specify a response queue outside its qspace.

The `name_space_list` argument also controls the cache access as follows:

To lookup a local queue reference or queue name, specify both `OTMQ _TBL_GRP` and `OTMQ_TBL_PROC`. This causes the process cache to be checked before looking into the group cache.

To lookup a global queue reference, specify `OTMQ_TBL_BUS` (or `OTMQ _TBL_BUS_LOW` or `OTMQ _TBL_BUS_MEDIUM`), `OTMQ _TBL_GRP` and `OTMQ _TBL_PROC`. This causes the process cache to be checked. Then, the group cache is checked before looking into the global name space.

Note that to lookup all caches in the global name space before looking in the master database, specify `OTMQ _TBL_BUS_LOW` instead of `OTMQ _TBL_BUS`.

To lookup only the slower but more up-to-date caches in the global name space before looking in the master database, specify `OTMQ _TBL_BUS_MEDIUM` instead of `OTMQ _TBL_BUS`.

`name_space_list_len`:

Supplies the number of entries in the `name_space_list` argument. If the `name_space_list_len` argument is zero, uses `OTMQ_TBL_GRP` as the default in the `name_space_list` argument.

scope

Specifies the scope for `pName`.

The identifies for scope:

`NAME_SCOPE_P`: scope is process

NAME_SCOPE_L: scope is local

NAME_SCOPE_G: scope is global

For TP APIs `scope` and `namespace_list` arguments, if `scope` is NULL, `namespace_list` must not NULL too, and if `scope` has a data value, `namespace_list` will invalidate.

timeout

The number of OTMQ time units (1 second intervals) to allow for the attach to complete. If a null pointer is specified, the `BLOCKTIME` property of the group's attach service is used (configured in the `UBBCONFIG *SERVICES` section). If service-wide `BLOCKTIME` is not configured, system-wide `BLOCKTIME` specified in the `UBBCONFIG *RESOURCES` section is used (default to approximately 60 seconds).

Return Value(s)

Upon failure, `tpqlocate()` returns -1 and sets `tperrno` to indicate the error condition. Otherwise, the queue has been successfully located when `tpqlocate ()` returns.

Errors:

[TPEINVAL]

Invalid arguments were given.

[TPENOENT]

Cannot access the qspace because it is not available

[TPETIME]

This error code indicates that either a timeout has occurred or `tpqlocate()` has been attempted, in spite of the fact that the current transaction is already marked rollback only.

[TPEPROTO]

`tpqlocate()` was called improperly.

[TPESYSTEM]

An Oracle Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[TPEOS]

An operating system error has occurred.

tpenqplus()

Name

`tpenqplus()` —Locates the queue name for the specified queue name or queue alias.

Synopsis

```
#include <atmi.h>
#include <tmqentry.h>
#include <tmqreturn.h>
int tpenqplus (qspace, qname, ctl, data, len, flags)
```

Description

Sends a message to a target queue in target qspace using a set of standard OTMQ delivery modes.

The block and DIP argument of the TPQCTL struct can be used to guarantee message delivery if a system, process, or network fails. Recoverable messages are stored on disk by the message recovery system until they can be delivered to the target queue of the receiver program. When sending a recoverable message, you must specify the uma argument of TPQCTL structure if the message recovery cannot store the message. You must also supply the TPQCTL structure PSB argument to receive the operation return status.

The optional timeout argument lets you set a maximum amount of time for the send operation to complete before the function times out. The optional replyqueue of TPQCTL struct argument allows you to specify an alternate queue for receiving the response messages rather than directing responses to the primary queue of the sender program. In synchronous mode, fail to receive response message from reply queue will cause tpenqplus() return error, but the message may be still sent to target queue successfully.

To use OTMQ features, you must invoke tpeqattach before invoke tpenqplus. The tpenqplus qspace argument must be the same same as the tpeqattach qspace argument.

If the caller is within a transaction and the TPNOTRAN flag is not set, the message is queued in transaction mode. This has the effect that if tpenqplus() returns successfully and the caller transaction is committed successfully, then the message is guaranteed to be available subsequent to the transaction completing. If the caller transaction is rolled back either explicitly or as the result of a transaction timeout or some communication error, then the message will be removed from the queue (that is, the placing of the message on the queue is also rolled back). It is not possible to enqueue then dequeue the same message within the same transaction.

The message is not queued in transaction mode if either the caller is not in transaction mode, or the TPNOTRAN flag is set. Once tpenqplus() returns successfully, the submitted message is guaranteed to be in the queue. When not in transaction mode, if a communication error or a timeout occurs, the application will not know whether or not the message was successfully stored on the queue.

To use tpenqplus(), you must first invoke tpeqattach().

Table 1-6 lists `tpenqplus()` supported arguments:

Table 1-6 tpenqplus() Arguments

Argument	Data Type	Mechanism	Prototype	Access
qspace	char	reference	char *	passed
qname	char	reference	char *	passed
ctl	TPQCTL	reference	TPQCTL *	passed/returned
data	char	reference	char *	passed
len	long	reference	long	passed
flags	long	reference	long	passed

qspace

Supplies the queue space name for enqueueing the message. The max length is 15

qname

Supplies the queue name for enqueueing the message, the max length is 127

Control Parameter

The TPQCTL structure is used by the application program to pass and retrieve parameters associated with enqueueing the message. The flags element of TPQCTL is used to indicate what other elements in the structure are valid.

On input to `tpenqplus()`, the following elements may be set in the TPQCTL structure:

The element of TPQCTL used for OTMQ:

flags

The following is a list of valid bits for the flags parameter controlling input information for `tpenqplus()`.

TPNOFLAGS

TPQTOPI

TPQBEFOREMSGID

TPQTIME_ABS

TPQTIME_REL

TPQPRIORITY

TPQCORRID

TPQREPLYQ

TPQFAILUREQ
TPQDELIVERYQOS, TPQREPLYQOS
TPQEXPTIME_ABS
TPQEXPTIME_REL
TPQEXPTIME_NONE
TPQMSGID
OTMQ
TPQGETBYFILTER
TPQGETMSGA
TPQREADJRN
TPQENDIAN
TPQGETBYSEQNUM

priority

Supplies the priority level for selective message reception. Priority ranges from 0 (lowest priority) to 99 (highest priority).

msg_class

Supplies the class code of message being sent. OTMQ supports the use of symbolic names for class argument values. Symbolic class names should begin with `MSG_CLAS_`. For information on defining class symbols, see the `tmqsym.h` include file. On UNIX and Windows NT systems, the `tmqsym.h` include file cannot be edited. You must create an include file to define type and class symbols for use by your application.

Class symbols reserved by OTMQ are as follows:

Reserved Class Symbol	Value
<code>MSG_CLAS_XXX</code>	30000 through 32767 (except 31001-31003)

msg_type

Supplies the type code for the message being sent. OTMQ supports the use of symbolic names for `msg_type` argument values. Symbolic type names begin with `MSG_TYPE_`. For information on defining type symbols, see the `tmqsym.h` include file.

OTMQ has reserved the symbol value range -1 through -5000. A zero value for this argument indicates that no processing by message type is expected.

block

Supplies the delivery mode for the message using the following format:

`OTMQ_DEL_sn`-where `sn` is one of the following sender notification constants:

WF-Wait for completion.

AK-Asynchronous acknowledgment

NN-No notification

Note: NN mode does not support transaction, TPNOTRAN flag is set automatically for this mode.

DIP

Dip is one of the following delivery interest point constants:

"ACK-Read from target queue and explicitly acknowledged using the `tpqconfirmmsg` function. ACK can also be an implicit acknowledgement sent after the second `tpdeqplus` call by the receiving application.

"CONF-Delivered from the DQF and explicitly confirmed using the `tpqconfirmmsg` function (recoverable)

"DEQ-Read from the target queue

"DQF-Stored in the destination queue file (recoverable)

"MEM-Stored in the target queue

"SAF-Stored in the store and forward file (recoverable)

Note: If temporary queues are used, deleted, and reused quickly, it is possible in isolated cases for an implicit ACK response from a previous temporary queue to be placed on the new temporary queue.

If you set `OTMQ_DEL_WF` and `OTMQ_DIP_ACK`, the ACK message will be read in `tpenqplus`, client do not need to call another API to read the ACK message.

timeout

Supplies the maximum amount of time the `tpenqplus` function waits for a message to arrive before returning control to the application. If the timeout occurs before a message arrives, the status code `OTMQ__TIMEOUT` is returned. Specifying 0 as the timeout value sets the timeout to the default value of 30 seconds.

psb

Receives a value in the OTMQ Status Block specifying the final completion status. The `psb` argument is used when sending or receiving recoverable messages. The PSB structure stores the status information from the message recovery system and may be checked after sending or receiving a message as shown in [Listing 1-3](#).

Listing 1-3 PSB Structure

Listing 1-3 PSB Structure

```

struct psb_t {
    long type_of_psb;          /* PSB type */

    long del_psb_status;      /* The completion status of the function.
It contains the status from TuxMsgQ. It can also contain a value of TPSUCCESS
when the message is not sent recoverably. */

    long uma_psb_status;      /* The completion status of the
undeliverable message action (UMA). The PSB UMA status indicates if the UMA
was not executed or applicable. */

    long psb_reserved[6];    /* reserved filed */
};

typedef struct psb_t PSB;

```

Note: this structure is already defined at atmi.h.

uma

Supplies the action to be performed if the message cannot be stored at the specified -delivery interest point. The format of this argument is OTMQ_UMA_XXX where XXX is one of the following symbols:

Symbol:	Description
DISC:	Discard message
RTS:	Return to sender
SAF:	Store and Forward
DLJ:	Dead letter journal
DLQ:	Dead letter queue

replyqueue

Supplies a `q_name` to use as the alternate queue for receiving response messages from the receiver program. The sender program must be attached to the queue specified in the `replyqueue` argument to receive the response messages. To use `replyqueue`, flags must address TPQREPLYQ.

Note the sender program cannot assign a response queue outside its qspace.

correlation_id

Supplies the correlation id, a user-defined identifier stored as a 32-byte value

seq_number

If the value of `seq_number[0]` or `seq_number[1]` is not specified as "(long)0", OTMQ will not generate unique number for this message.

Data

If data is non-NULL, it must point to a buffer previously allocated by `tpalloc()` and `len` should specify the amount of data in the buffer that should be queued. Note that if data points to a buffer of a type that does not require a length to be specified (for example, an FML fielded buffer), then `len` is ignored. If data is NULL, `len` is ignored and a message is queued with no data portion. Consult `tpalloc()` for more details.

Len

The length of data, if data is non-NULL.

Flags

The following is a list of valid flags:

TPNOTRAN

If the caller is in transaction mode and this flag is set, the message is not queued within the caller transaction. A caller in transaction mode that sets this flag is still subject to the transaction timeout (and no other) when queuing the message. If message queuing fails, the caller's transaction is not affected.

TPNOBLOCK

The message is not enqueued if a blocking condition exists. If this flag is set and a blocking condition exists such as the internal buffers into which the message is transferred are full, the call fails and `tperrno` is set to `TPEBLOCK`. If this flag is set and a blocking condition exists because the target queue is opened exclusively by another application, the call fails, `tperrno` is set to `TPEDIAGNOSTIC`, and the diagnostic field of the `TPQCTL` structure is set to `QMESHARE`.

When `TPNOBLOCK` is not set and a blocking condition exists, the caller blocks until the condition subsides or a timeout occurs (either transaction or blocking timeout). If a timeout occurs, the call fails and `tperrno` is set to `TPETIME`.

TPNOTIME

Setting this flag signifies that the caller is willing to block indefinitely and wants to be immune to blocking timeouts. Transaction timeouts may still occur.

TPSIGRSTRT

If this flag is set and a signal interrupts any underlying system calls, the interrupted system call is reissued. If **TPSIGRSTRT** is not set and a signal interrupts a system call, `tpenqplus()` fails and `tperrno` is set to **TPGOTSIG**.

Return Value(s)

Upon failure, `tpenqplus()` returns -1 and sets `tperrno` to indicate the error condition. Otherwise, the message has been successfully enqueue when `tpenqplus()` returns.

Errors:

[TPEINVAL]

Invalid arguments were given.

[TPEOENT]

Cannot access the qspace because it is not available

[TPETIME]

This error code indicates that either a timeout has occurred or `tpenqplus()` has been attempted, in spite of the fact that the current transaction is already marked rollback only.

[TPEBLOCK]

A blocking condition exists and **TPNOBLOCK** was specified.

[TPGOTSIG]

A signal was received and **TPSIGRSTRT** was not specified.

[TPEPROTO]

`tpenqplus()` was called improperly.

[TPESYSTEM]

An Oracle Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[TPEOS]

An operating system error has occurred.

[TPEDIAGNOSTIC]

Enqueuing a message on the specified queue failed. The reason for failure can be determined by the diagnostic returned via `ctl`.

Diagnostic:

[QMEINVAL]

An invalid flag value was specified.

[QMEBADRMID]

An invalid resource manager identifier was specified.

[QMENOTOPEN]

The resource manager is not currently open.

[QMETRAN]

The call was not in transaction mode or was made with the `TPNOTRAN` flag set and an error occurred trying to start a transaction in which to enqueue the message. This diagnostic is not returned by queue managers from Oracle Tuxedo release 7.1 or later.

[QMEBADMSGID]

An invalid message identifier was specified.

[QMESYSTEM]

A system error occurred. The exact nature of the error is written to a log file.

[QMEOS]

An operating system error occurred.

[QMEABORTED]

The operation was aborted. When executed within a global transaction, the global transaction has been marked rollback-only. Otherwise, the queue manager aborted the operation.

[QMEPROTO]

An enqueue was done when the transaction state was not active.

[QMEBADQUEUE]

An invalid or deleted queue name was specified.

[QMENOSPACE]

Due to an insufficient resource, such as no space on the queue, the message with its required quality of service (persistent or non-persistent storage) was not enqueued.

[QMERELASE]

An attempt was made to enqueue a message to a queue manager that is from a version of the Oracle Tuxedo system that does not support a newer feature.

[QMBADDELIVERY]

Invalid delivery mode.

[QMBADPRIORITY]

Invalid priority value on send operation.

[QMBADPROCNUM]

Invalid target queue name specified.

[QMBADRESPQ]

Response queue not owned by process.

[QMBADUMA]

Undeliverable message action (UMA) is invalid.

[QMNOTSUPPORTED]

The combination of delivery mode and uma selected is not supported.

tpdeqplus()

Name

`tpdeqplus()`—Retrieves the next available message from a selected queue and moves it to the location specified in the data argument.

Synopsis

```
#include <atmi.h>
#include <tmqentry.h>
#include <tmqreturn.h>
int tpdeqplus (qspace, qname, ctl, data, len, flags)
```

Description

Retrieves the next available message from a selected queue and moves it to the location specified in the data argument. When no selection filter is specified, the function returns the next available message in first-in/first-out (FIFO) order based on message priority to the buffer specified in the data argument. Priority ranges from 0 (lowest priority) to 99 (highest priority). For example, priority 1 messages are always placed before priority 0 messages. Messages are placed in first-in/first out order by message priority. If a selection filter is specified, then only messages that meet the selection criteria are retrieved. If no messages are available or meet the selection criteria, then the return status is `QMENOMSG`.

Applications should check the PSB status field of each message to determine if the message was sent with a recoverable delivery mode. If an application receives a recoverable message, it must call the `tpqconfirmmsg` function to delete it from the message recovery journal disk storage. If receipt of a recoverable message is not confirmed, the message continues to be stored by the recovery system and will be re-delivered if the application detaches and then reattaches to the queue.

To use `tpdeqplus()`, you must first invoke `tpqattach()`.

[Table 1-7](#) lists `tpdeqplus()` supported arguments:

Table 1-7 tpdeqplus() Arguments

Argument	Data Type	Mechanism	Prototype	Access
qspace	char	reference	char *	passed
qname	char	reference	char *	passed
ctl	TPQCTL	reference	TPQCTL *	passed/returned
data	char	reference	char **	returned
len	long	reference	long *	passed/returned
flags	long	reference	long	passed

qspace

Supplies the queue space name for message dequeue. The max length is 15

qname

Supplies the queue name for message enqueue. The max length is 127

Data

Data is the address of a pointer to the buffer into which a message is read, and len points to the length of that message. *data must point to a buffer originally allocated by `tpalloc()`. If a message is larger than the buffer passed to `tpdeqplus`, the buffer is increased in size to accommodate the message. To determine whether a message buffer changed in size, compare its (total) size before `tpdeqplus()` was issued with *len. If *len is larger, then the buffer has grown; otherwise, the buffer has not changed size. Note that *data may change for reasons other than if the buffer size increased. If *len is 0 upon return, then the message dequeued has no data portion and neither *data nor the buffer it points to were modified. It is an error for *data or len to be NULL. Consult `tpalloc()` for more details.

The message is dequeued in transaction mode if the caller is in transaction mode and the `TPNOTRAN` flag is not set. This has the effect that if `tpdeqplus()` returns successfully and the caller transaction is committed successfully, then the message is removed from the queue. If the caller's transaction is rolled back either explicitly or as the result of a transaction timeout or some communication error, then the message will be left on the queue (that is, the removal of the message from the queue is also rolled back). It is not possible to enqueue and dequeue the same message within the same transaction.

The message is not dequeued in transaction mode if either the caller is not in transaction mode, or the `TPNOTRAN` flag is set. When not in transaction mode, if a communication

error or a timeout occurs, the application will not know whether or not the message was successfully dequeued and the message may be lost.

Len

The length of data, if data is non-NULL.

ctrl

The TPQCTL structure .

flags

The following is a list of valid bits for the flags parameter controlling input information for `tpdeqplus()`.

TPNOFLAGS
TPQGETBYMSGID
TPQGETBYCORRID
TPQWAIT
TPQPEEK
OTMQ
TPQGETBYFILTER
TPQGETMSGGA
TPQREADJRN
TPQENDIAN
TPQGETBYSEQNUM
TPQGETBYMSGCLASS
TPQGETBYMSGTYPE

priority

Supplies the priority level for selective message reception. Priority ranges from 0 (lowest priority) to 99 (highest priority).

msg_class

Supplies the class code of message being sent. OTMQ supports the use of symbolic names for class argument values. Symbolic class names should begin with `MSG_CLAS_`. For information on defining class symbols, see the `tmqsym.h` include file. On UNIX and Windows NT systems, the `tmqsym.h` include file cannot be edited. You must create an include file to define type and class symbols for use by your application.

Class symbols reserved by OTMQ are as follows:

Reserved Class	Symbol Value
<code>ACK_CLASS</code>	28
<code>MSG_CLAS_PAMS</code>	29

TUXEDO_MSG	31001
MSG_CLAS_TUXEDO_TPSUCCESS	31002
MSG_CLAS_TUXEDO_TPFFAIL	31003
MSG_CLAS_XXX	30000 through 32767 (except 31001-31003)

msg_type

Supplies the type code for the message being sent. OTMQ supports the use of symbolic names for `msg_type` argument values. Symbolic type names begin with `MSG_TYPE_`. For information on defining type symbols, see the `tmqsym.h` include file.

OTMQ has reserved the symbol value range -1 through -5000. A zero value for this argument indicates that no processing by message type is expected.

block

Supplies the delivery mode for the message using the following format:

"OTMQ_DEL_sn-where sn is one of the following sender notification constants:

"WF-Wait for completion

"AK-Asynchronous acknowledgment

"NN-No notification

Note: NN mode does not support transaction, `TPNOTRAN` flag is set automatically for this mode.

DIP

Dip is one of the following delivery interest point constants:

"ACK-Read from target queue and explicitly acknowledged using the `tpqconfirmmsg` function. ACK can also be an implicit acknowledgement sent after the second `tpdeqplus` call by the receiving application.

"CONF-Delivered from the DQF and explicitly confirmed using the `tpqconfirmmsg` function (recoverable)

"DEQ-Read from the target queue

"DQF-Stored in the destination queue file (recoverable)

"MEM-Stored in the target queue

"SAF-Stored in the store and forward file (recoverable)

Note: If temporary queues are used, deleted, and reused quickly, it is possible in isolated cases for an implicit ACK response from a previous temporary queue to be placed on the new temporary queue.

If set `OTMQ_DIP_ACK` and `OTMQ_DEL_WF`, the ACK message will be read in `tpenqplus, _client` do not need to call another API to read the ACK message.

timeout

Supplies the maximum amount of time the `tpenqplus` function waits for a message to arrive before returning control to the application. If the timeout occurs before a message arrives, the status code `OTMQ_TIMEOUT` is returned. Specifying 0 as the timeout value sets the timeout to the default value of 30 seconds.

psb

Receives a value in the OTMQ Status Block specifying the final completion status. The `psb` argument is used when sending or receiving recoverable messages. The PSB structure stores the status information from the message recovery system and may be checked after sending or receiving a message as shown in [Listing 1-4](#).

Listing 1-4 PSB Structure

```

struct psb_t {
    long type_of_psb;           /* PSB type */
    long del_psb_status;       /* The completion status of the function.
It contains the status from TuxMsgQ. It can also contain a value of TPSUCCESS
when the message is not sent recoverably. */
    long uma_psb_status;       /* The completion status of the
undeliverable message action (UMA). The PSB UMA status indicates if the UMA
was not executed or applicable. */
    long psb_reserved[6];     /* reserved filed */
};
typedef struct psb_t PSB;

```

Note: This structure is already defined at `atmi.h`.

Table 1-8 PSB Delivery

PSB Delivery Status	Platform	Description	API
OTMQ__CONFIRMREQ	All	Confirmation required for this message.	Tpdeqplus ()
OTMQ__DOWN	All	The specified OTMQ Queue Service Group is not running.	All
OTMQ__POSSDUPL	All	Message is a possible duplicate.	Tpdeqplus ()
OTMQ__NO_DQF	All	When DQF is disabled, message delivery with DQF mode will return PSB del_psb_status as OTMQ__NO_DQF	Tpenqplus ()
OTMQ__NO_SAF	All	When SAF/DQF journal is disabled, message delivery with SAF mode will return PSB del_psb_status as OTMQ__NO_SAF	Tpenqplus ()
OTMQ__SUCCESS	All	Indicates successful completion.	All

uma

Supplies the action to be performed if the message cannot be stored at the specified -delivery interest point. The format of this argument is OTMQ_UMA_XXX where XXX is one of the following symbols:

Symbol	Description
DISC	Discard message
RTS	Return to sender
SAF	Store and Forward
DLJ	Dead letter journal
DLQ	Dead letter queue

replyqueue

Supplies a q_name to use as the alternate queue for receiving response messages from the receiver program. The sender program must be attached to the queue specified in the replyqueue argument to receive the response messages. The sender program cannot assign a response queue outside its qspace.

correlation_id

Supplies the correlation id, a user-defined identifier stored as a 32-byte value

filter_idx

Get message by filter, `filter_idx` set as `index_handle` is created using the `tpqsetselect` function.

Data

If data is non-NULL, it must point to a buffer previously allocated by `tpalloc()` and `len` should specify the amount of data in the buffer that should be queued. Note that if data points to a buffer of a type that does not require a length to be specified (for example, an FML fielded buffer), then `len` is ignored. If data is NULL, `len` is ignored and a message is queued with no data portion. Consult `tpalloc()` for more details.

Len

The length of data, if data is non-NULL.

Flags

The following is

TPNOTRAN

If the caller is in transaction mode and this flag is set, the message is not queued within the caller transaction. A caller in transaction mode that sets this flag is still subject to the transaction timeout (and no other) when queuing the message. If message queuing fails, the caller's transaction is not affected.

TPNOBLOCK

The message is not enqueued if a blocking condition exists. If this flag is set and a blocking condition exists such as the internal buffers into which the message is transferred are full, the call fails and `tperrno` is set to `TPEBLOCK`. If this flag is set and a blocking condition exists because the target queue is opened exclusively by another application, the call fails, `tperrno` is set to `TPEDIAGNOSTIC`, and the diagnostic field of the `TPQCTL` structure is set to `QMESHAPE`.

When `TPNOBLOCK` is not set and a blocking condition exists, the caller blocks until the condition subsides or a timeout occurs (either transaction or blocking timeout). If a timeout occurs, the call fails and `tperrno` is set to `TPETIME`.

TPNOTIME

Setting this flag signifies that the caller is willing to block indefinitely and wants to be immune to blocking timeouts. Transaction timeouts may still occur.

TPSIGRSTRT

If this flag is set and a signal interrupts any underlying system calls, the interrupted system call is reissued. If `TPSIGRSTRT` is not set and a signal interrupts a system call, `tpenqplus()` fails and `tperrno` is set to `TPGOTSIG`.

Return Value(s)

Upon failure, `tpdeqplus()` returns -1 and sets `tperrno` to indicate the error condition. Otherwise, the queue has been successfully dequeued when `tpdeqplus()` returns.

Errors:

Upon failure, `tpdeqplus()` sets `tperrno` to one of the following values. (Unless otherwise noted, failure does not affect the caller's transaction, if one exists.)

[TPEINVAL]

Invalid arguments were given.

[TPENOENT]

Cannot access the qspace because it is not available.

[TPEOTYPE]

Either the type and subtype of the dequeued message are not known to the caller; or, `TPNOCHANGE` was set in flags and the type and subtype of `*data` do not match the type and subtype of the dequeued message. In either case, `*data`, its contents, and `*len` are not changed. When the call is made in transaction mode and this error occurs, the transaction is marked abort-only, and the message remains on the queue.

[TPETIME]

This error code indicates that either a timeout has occurred.

[TPEBLOCK]

A blocking condition exists and `TPNOBLOCK` was specified.

[TPGOTSIG]

A signal was received and `TPSIGRSTRT` was not specified.

[TPEPROTO]

`tpdeqplus()` was called improperly. There is no effect on the queue or the transaction.

[TPESYSTEM]

An Oracle Tuxedo system error has occurred. The exact nature of the error is written to a log file. There is no effect on the queue.

[TPEOS]

An operating system error has occurred. There is no effect on the queue.

[TPEDIAGNOSTIC]

Dequeuing a message from the specified queue failed. The reason for failure can be determined by the diagnostic value returned via `ctl` structure.

Diagnostic:

The following diagnostic values are returned during the dequeuing of a message:

[QMEINVAL]

An invalid flag value was specified.

[QMEBADRMID]

An invalid resource manager identifier was specified.

[QMENOTOPEN]

The resource manager is not currently open.

[QMEBADMSGID]

An invalid message identifier was specified for dequeuing.

[QMESYSTEM]

A system error has occurred. The exact nature of the error is written to a log file.

[QMEOS]

An operating system error has occurred.

[QMEABORTED]

The operation was aborted. When executed within a global transaction, the global transaction has been marked rollback-only. Otherwise, the queue manager aborted the operation.

[QMEPROTO]

A dequeue was done when the transaction state was not active.

[QMEBADQUEUE]

An invalid or deleted queue name was specified.

[QMENOMSG]

No message was available for dequeuing. Note that it is possible that the message exists on the queue and another application process has read the message from the queue. In this case, the message may be put back on the queue if that other process rolls back the transaction.

[QMBADPRIORITY]

Invalid priority value used for receive.

[QMNOTDCL]

Process has not been attached to OTMQ.

tpqpublish()

Name

`tpqpublish()`—Used to publish a topic data.

Synopsis

```
#include <atmi.h>
#include <tmqentry.h>
#include <tmqreturn.h>
int tppublish (topic_name, data, len, flags)
```

Description

The caller uses `tppublish()` to publish a topic data. The topic is named by `topic` and `data`, if not NULL, points to the data. The topic and its data are dispatched by the Oracle Tuxedo ATMI EventBroker to all subscribers whose subscriptions successfully evaluate against `topic` and whose optional filter rules successfully evaluate against `data`.

[Table 1-9](#) lists `tppublish()` supported arguments:

Table 1-9 tppublish() Arguments

Argument	Data Type	Mechanism	Prototype	Access
topicname	char	reference	char *	passed
data	char	reference	char *	passed
len	long	reference	long	passed
flags	long	reference	long	passed

topicname

`topicname` is a NULL-terminated string of at most 31 characters and start with "TMQ:<QNOT>:qspace name", such as "TMQ:QNOT:QSPACE:usertopic". The first `topicname` character cannot be a dot (".") as this character is reserved as the starting character for all events defined by the Oracle Tuxedo ATMI system itself. The `topicname` "TMQ:QNOT: qspace name" is the suffix for all user topic and can not be used as user topic name alone. "QNOT" is not an necessary string. But if `topicname` contain "QNOT", means the message which will be published is an AVAIL/UNAVAIL message.

Data

If `data` is non-NULL, it must point to a buffer previously allocated by `tpalloc()` and `len` should specify the amount of data in the buffer that should be posted with the event. Note that if `data` points to a buffer of a type that does not require a length to be specified (for example, an FML fielded buffer), then `len` is ignored. If `data` is NULL, `len` is ignored and the event is posted with no data.

Flags

If the publisher is within a transaction and the `TPNOTRAN` flag is not set, the publish topic goes to the EventBroker in transaction mode such that it dispatches the event as part of the publisher transaction. The broker dispatches transactional event notifications only to those service routine and stable-storage queue subscriptions that used the `TPEVTRAN` bit setting in the `ctl-?` flags parameter passed to `tpqsubscribe()`. Client notifications, and those service routine and stable-storage queue subscriptions that did not use the `TPEVTRAN` bit setting in the `ctl-?` flags parameter passed to `tpqsubscribe()`, are also dispatched by the EventBroker but not as part of the publisher process transaction.

If the publisher is outside a transaction, `tpqpublish()` is a one-way publish topic with no acknowledgement when the service associated with the event fails. This occurs even when `TPEVTRAN` is set for that event (using the `ctl.flags` parameter passed to `tpqsubscribe()`). If the publisher is in a transaction, then `tpqpublish()` returns `TPESVCFAIL` when the associated service fails in the event.

The following is a list of valid flags:

TPNOTRAN

If the caller is in transaction mode and this flag is set, then the event publishing is not made on behalf of the caller transaction. A caller in transaction mode that sets this flag is still subject to the transaction timeout (and no other) when posting events. If the event posting fails, the caller's transaction is not affected.

TPNOREPLY

Informs `tpqpublish()` not to wait for the EventBroker to process all subscriptions for topic before returning. When `TPNOREPLY` is set, `tpurcode()` is set to zero regardless of whether `tpqpublish()` returns successfully or not. When the caller is in transaction mode, this setting cannot be used unless `TPNOTRAN` is also set.

TPNOBLOCK

The topic is not published if a blocking condition exists. If such a condition occurs, the call fails and `tperrno` is set to `TPEBLOCK`. When `TPNOBLOCK` is not specified and a blocking condition exists, the caller blocks until the condition subsides or a timeout occurs (either transaction or blocking timeout).

TPNOTIME

This flag signifies that the caller is willing to block indefinitely and wants to be immune to blocking timeouts. Transaction timeouts may still occur.

TPSIGRSTRT

If a signal interrupts any underlying system calls, then the interrupted system call is reissued. When `TPSIGRSTRT` is not specified and a signal interrupts a system call, then `tpqpublish()` fails and `tperrno` is set to `TPGOTSIG`.

In a multithreaded application, a thread in the `TPINVALIDCONTEXT` state is not allowed to issue a call to `tpqpublish()`.

Return Value(s)

Upon failure, `tpqpublish()` returns -1 and sets `tperrno` to indicate the error condition. Otherwise, the message has been successfully broadcasted when `tpqpublish()` returns.

Errors:

[TPEINVAL]

Invalid arguments were given.

[TPENOENT]

Cannot access the TMQEVT

[TPETIME]

This error code indicates that either a timeout has occurred or `tpqpublish()` has been attempted, in spite of the fact that the current transaction is already marked rollback only.

[TPEBLOCK]

A blocking condition exists and `TPNOBLOCK` was specified.

[TPGOTSIG]

A signal was received and `TPSIGRSTRT` was not specified.

[TPEPROTO]

`tpqpublish()` was called improperly.

[TPESYSTEM]

An Oracle Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[TPEOS]

An operating system error has occurred.

tpqsubscribe()

Name

`tpqsubscribe()`—Used to subscribe to a topic.

Synopsis

```
#include <atmi.h>
#include <tmqentry.h>
#include <tmqreturn.h>
long tpqsubscirbe (topic, filter, ctl, flags)
```


Description

The caller uses `tpqsubscribe()` to subscribe to a topic. Topic is a NULL-terminated string of at most 255 characters containing a regular expression. If present, filter is a string containing a Boolean filter rule that must be evaluated successfully before the TMQEVT posts the topic. Upon receiving an topic to be published, the TMQEVT applies the filter rule, if one exists, to the publish topic string. If the data passes the filter rule, the TMQEVT invokes the notification method; otherwise, the broker does not invoke the associated notification method. The caller can subscribe to the same event multiple times with different filter rules.

Table 1-10 lists `tpqsubscribe()` supported arguments:

Table 1-10 tpqsubscribe() Arguments

Argument	Data Type	Mechanism	Prototype	Access
topic	char	reference	char*	passed
filter	char	reference	char*	passed
ctrl	TPEVCTL	reference	TPEVCTL *	passed
flags	long	reference	long	passed

topic

topic is a NULL-terminated string of at most 31 characters and starts with "TMQ:<QNOT>:qspace name" (for example "TMQ:QNOT:QSPACE:usertopic"). The first topic character cannot be a dot (".") as this character is reserved as the starting character for all events defined by the Oracle Tuxedo ATMI system itself. The topic "TMQ:QNOT:qspace name" is the suffix for all user topics and cannot be used as a user topic name alone."QNOT" is not a necessary string. But if a topic contains "QNOT", it indicates the published message is an AVAIL/UNAVAIL message.

filter

filter is a string containing a Boolean filter rule that must be evaluated successfully before the TMQEVT posts the topic. See Regular Expressions.

ctrl

The TPEVCTL structure as shown at `atmi.h`

This structure contains the following elements:

```
/* Subscription Control structure */
struct tpevctl_t {
```

```

long flags;
char name1[XATMI_SERVICE_NAME_LENGTH];
char name2[XATMI_SERVICE_NAME_LENGTH];
TPQCTL qctl;
};
typedef struct tpevctl_t TPEVCTL;

```

The following is a list of valid bits for the `ctl->flags` element controlling options for topic subscriptions:

TPEVQUEUE

Setting this flag indicates that the subscriber wants topic notifications to be enqueued to the queue space named in `ctl->name1` and the queue named in `ctl->name2`. That is, when an topic name is published that evaluates successfully against topic, the TMQEVT tests the published data against the filter rule associated with topic. If the data passes the filter rule or if there is no filter rule for the topic, then the TMQEVT enqueues a message to the queue space named in `ctl->name1` and the queue named in `ctl->name2` along with any data published with the topic. The queue space and queue name can be any valid Oracle Tuxedo ATMI system queue space and queue name, either of which may or may not exist at the time the subscription is made.

`ctl->qctl` can contain options further directing the TMQEVT enqueueing of the published topic. If no options are specified, then `ctl->qctl.flags` should be set to `TPNOFLAGS`. Otherwise, options can be set as described in the "Control Parameter" subsection of `topenqplus`. `TPEVQUEUE` are mutually exclusive flags. If `TPEVTRAN` is also set in `ctl->flags`, then if the process calling `tpqpublish()` is in transaction mode, the TMQEVT enqueues the published topic and its data such that it will be part of the publisher transaction. The TMQEVT must belong to a server group that supports transactions (see `UBBCONFIG` for details). If `TPEVTRAN` is not set in `ctl->flags`, then the TMQEVT enqueues the published topic and its data such that it will not be part of the publisher transaction.

TPEVTRAN

Setting this flag indicates that the subscriber wants the topic notification for this subscription to be included in the publisher transaction, if one exists. If the publisher is not a transaction, then a transaction is started for this topic notification. If this flag is not set, then any topics published for this subscription will not be done on behalf of any transaction in which the publisher is participating.

For subscriptions to stable-storage queues, the queue space, queue name, and correlation identifier are used, in addition to topic and filter, when determining

matches. The correlation identifier can be used to differentiate among several subscriptions for the same topic expression and filter rule, destined for the same queue. Thus, if the caller has set `ctl->flags` to `TPEVQUEUE`, and `TPQCOORD` is not set in `ctl->qctl.flags`, then `tpqsubscribe()` fails if topic, filter, the queue space name set in `ctl->name1`, and the queue name set in `ctl->name2` match those of a subscription (which also does not have a correlation identifier specified) already known to the `TMQEVT`. Further, if `TPQCOORD` is set in `ctl->qctl.flags`, then `tpqsubscribe()` fails if topic, filter, `ctl->name1`, `ctl->name2`, and `ctl->qctl.corrid` match those of a subscription (which has the same correlation identifier specified) already known to the `TMQEVT`.

TPEVPERSIST

By default, the OTMQ deletes subscriptions when the resource to which it is posting is not available (for example, the OTMQ cannot access a service routine and/or a queue space/queue name associated with an subscription). Setting this flag indicates that the subscriber wants this subscription to persist across such errors (usually because the resource will become available again in the future). When this flag is not used, the OTMQ will remove this subscription if it encounters an error accessing either the service name or queue space/queue name designated in this subscription.

If this flag is used with `TPEVTRAN` and the resource is not available at the time of event notification, then the `EventBroker` will return to the poster such that its transaction must be aborted. That is, even though the subscription remains intact, the resource's unavailability will cause the poster's transaction to fail.

flags

The following is a list of valid flags for `tpqsubscribe()`:

OTMQ

Note: OTMQ is must be set for `TPQCTL->flags`.

TPNOBLOCK

The subscription is not made if a blocking condition exists. If such a condition occurs, the call fails and `tperrno` is set to `TPEBLOCK`. When `TPNOBLOCK` is not specified and a blocking condition exists, the caller blocks until the condition subsides or a timeout occurs (either transaction or blocking timeout).

TPNOTIME

This flag signifies that the caller is willing to block indefinitely and wants to be immune to blocking timeouts. Transaction timeouts may still occur.

TPSIGRSTRT

If a signal interrupts any underlying system calls, then the interrupted system call is reissued. When **TPSIGRSTRT** is not specified and a signal interrupts a system call, then `tpqsubscribe()` fails and `tperrno` is set to **TPGOTSIG**.

In a multithreaded application, a thread in the **TPINVALIDCONTEXT** state is not allowed to issue a call to `tpqsubscribe()`.

Return Value(s)

Upon failure, `tpqsubscribe()` returns -1 and sets `tperrno` to indicate the error condition. Otherwise, the topic has been successfully subscribed when `tpqsubscribe()` returns and returns a subscription handle.

Errors:

[TPEINVAL]

Invalid arguments were given.

[TPENOENT]

Cannot access **TMQEVT**

[TPETIME]

This error code indicates that either a timeout has occurred or `tpqsubscribe()` has been attempted, in spite of the fact that the current transaction is already marked rollback only.

[TPEBLOCK]

A blocking condition exists and **TPNOBLOCK** was specified.

[TPGOTSIG]

A signal was received and **TPSIGRSTRT** was not specified.

[TPEPROTO]

`tpqsubscribe()` was called improperly.

[TPESYSTEM]

An Oracle Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[TPEOS]

An operating system error has occurred.

[TPELIMIT]

The subscription failed because the `EventBroker` maximum number of subscriptions has been reached.

tpqunsubscribe()

Name

`tpqunsubscribe()`—Used to remove a subscription.

Synopsis

```
#include <atmi.h>
#include <tmqentry.h>
#include <tmqreturn.h>
int tpqunsubscribe (subscription, flags)
```

Description

Used to remove a subscription.

[Table 1-11](#) lists `tpqunsubscribe()` supported arguments:

Table 1-11 tpqunsubscribe() Arguments

Argument	Data Type	Mechanism	Prototype	Access
subscription	long	reference	long	passed
flags	long	reference	long	passed

Subscription

subscription is an subscription handle returned by `tpqsubscribe()`. Setting subscription to the wildcard value, -1, directs `tpqunsubscribe()` to unsubscribe to all non-persistent subscriptions previously made by the calling process. Non-persistent subscriptions are those made without the `TPEVPERSIST` bit setting in the `ctl1-?flags` parameter of `tpqsubscribe()`. Persistent subscriptions can be deleted only by using the handle returned by `tpqsubscribe()`.

flags

The following is a list of valid flags for `tpqsubscribe()`:

TPNOBLOCK

The subscription is not made if a blocking condition exists. If such a condition occurs, the call fails and `tperrno` is set to `TPEBLOCK`. When `TPNOBLOCK` is not specified and a blocking condition exists, the caller blocks until the condition subsides or a timeout occurs (either transaction or blocking timeout).

TPNOTIME

This flag signifies that the caller is willing to block indefinitely and wants to be immune to blocking timeouts. Transaction timeouts may still occur.

TPSIGRSTRT

If a signal interrupts any underlying system calls, then the interrupted system call is reissued. When **TPSIGRSTRT** is not specified and a signal interrupts a system call, then `tpunsubscribe()` fails and `tperrno` is set to `TPGOTSIG`.

In a multithreaded application, a thread in the `TPINVALIDCONTEXT` state is not allowed to issue a call to `tpunsubscribe()`.

Return Value(s)

Upon failure, `tpqunsubscribe()` sets `tperrno` to one of the following values. (Unless otherwise noted, failure does not affect the caller's transaction, if one exists.)

[TPEINVAL]

Invalid arguments were given.

[TPENOENT]

Cannot access the OTMQ EventBroker.

[TPETIME]

This error code indicates that either a timeout has occurred

[TPEBLOCK]

A blocking condition exists and `TPNOBLOCK` was specified.

[TPGOTSIG]

A signal was received and `TPSIGRSTRT` was not specified.

[TPEPROTO]

`tpunsubscribe()` was called improperly.

[TPESYSTEM]

An Oracle Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[TPEOS]

An operating system error has occurred.

tpqconfirmmsg()

Name

`tpqconfirmmsg()`—Confirms receipt of a message that requires explicit confirmation.

Synopsis

```
#include <atmi.h>
#include <tmqentry.h>
#include <tmqreturn.h>
int tpqconfirmmsg (seq_number, force_j)
```

Description

Confirms receipt of a message that requires explicit confirmation. This can be a recoverable message sent to a queue that is configured for explicit confirmation or a message sent using the ACK delivery mode which must be explicitly confirmed upon receipt. Applications should examine the PSB status field of each message received to determine if the message requires explicit confirmation.

When a recoverable message is received, the application must call the `tpqconfirmmsg` function in order to delete it from the message recovery journal disk storage. If receipt of a recoverable message is not confirmed, the message continues to be stored by the recovery system and will be re-delivered if the application detaches and then reattaches to the queue.

OTMQ can confirm receipt of a recoverable message automatically when the next consecutive message in the recovery journal is delivered. This feature is called implicit confirmation.

All queues must be configured for implicit or explicit confirmation. For complete information on how to configure message queues, see `tmqadmin` in the Oracle Tuxedo Message Queue Command Reference Guide.

Successfully delivered recoverable messages can be recorded in the postconfirmation journal (PCJ). The `tpqconfirmmsg` function uses the `force_j` argument to write messages to the PCJ file if the system is not currently configured to store them. Note that successfully delivered recoverable messages cannot be written to the PCJ file unless they are explicitly confirmed using the `tpqconfirmmsg` function.

To use `tpqconfirmmsg()`, you must first invoke `tpqattach()`.

Note: `tpqconfirmmsg()` is an asynchronous invocation, when it returns, the recoverable message in the recovery journal disk storage may not have been deleted by the recovery system.

[Table 1-12](#) lists `tpqconfirmmsg()` supported arguments:

Table 1-12 `pqconfirmmsg()` Arguments

Argument	Data Type	Mechanism	Prototype	Access
<code>seq_number</code>	long	reference	long *	passed
<code>force_j</code>	int	reference	int	passed

seq_number

Supplies the message sequence number of the recoverable message being confirmed. The message sequence number is generated by the OTMQ message recovery system for each recoverable message. This value is passed to the receiver program in the PSB of the `tpdeqplus` function when it reads each recoverable message.

force_j

Supplies the journaling action for this message. Following are the predefined constants for this argument:

Symbol	Description
<code>OTMQ_DEFAULT_JRN</code>	Enables writing the message to the PCJ queue only if the journaling is enabled by <code>qspacecreate</code> command.
<code>OTMQ_FORCE_JRN</code>	Enables writing the message to the PCJ queue only if the journaling is enabled by <code>qspacecreate</code> command.
<code>OTMQ_NO_JRN</code>	Disables journaling regardless of whether journaling is configured.

Return Value(s)

Upon failure, `tpqconfirmmsg()` sets `tperrno` to one of the following values. (Unless otherwise noted, failure does not affect the caller's transaction, if one exists.)

[TPEINVAL]

Invalid arguments were given.

[TPENOENT]

Cannot access the qspace because it is not available.

[TPETIME]

This error code indicates that either a timeout has occurred.

[TPESYSTEM]

An Oracle Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[TPEOS]

An operating system error has occurred.

tpqsetselect()

Name

`tpqsetselect()`—Allows application developers to define complex selection criteria for message reception.

Synopsis

```
#include <atmi.h>
#include <tmqentry.h>
#include <tmqreturn.h>
int tpqsetselect (selection_array, num_masks, index_handle)
```

Description

Allows application developers to define complex selection criteria for message reception. The selection array specifies the queues to search, the priority order of message reception, two comparison keys for range checking, and an order key to determine the order in which messages are selected from the queue.

The `tpqsetselect` function creates an index handle that is used as the `sel_filter` argument of OTMQ functions for reading the message. When a selection index handle is passed to `tpdeqplus`, each message received is compared against comparison `key_1` and then comparison `key_2`. If the message matches both keys (a logical AND operation), the message is added to a set of matched messages. The order in which selected messages are delivered is determined by the order key.

To use `tpqsetselect()`, you must first invoke `tpqattach()`.

[Table 1-13](#) lists `tpqsetselect()` supported arguments:

Table 1-13 tpqsetselect() Arguments

Argument	Data Type	Mechanism	Prototype	Access
selection_array	selection_array_component_tp	reference	selection_array_component_tp *	passed
num_masks	short	reference	short *	passed
index_handle	Int	reference	int *	returned

selection_array

Supplies an array of selection records that contain the selection rules for each queue. The typedef structures define the C data structure for the selection array. The structure is defined in `tmqentry.h` as follows:

```
typedef struct _selection_array_component_tp {
    char qspace[TMQSNAMELEN+1];
    char queue[QNAMELEN+1];
    TM32I priority;
    TM32I key_1_offset;
    TM32I key_1_size;
    TM32I key_1_value;
    TM32I key_1_oper;
    TM32I key_2_offset;
    TM32I key_2_size;
    TM32I key_2_value;
    TM32I key_2_oper;
    TM32I order_offset;
    TM32I order_size;
    TM32I order_order;
    char key_value_qspace[TMQSNAMELEN+1];
    char key_value_queue[TMQNAMELEN+1];
    union {
        otmq_correlation_id correlation_id;
        otmq_sequence_number sequence_number
    } extended_key
} selection_array_component_tp;
```

The `selection_array_component_tp` data structure has the following components:

Table 1-14 selection_array_component_tp Data Structure Components

Component	Description
QSpace, Queue and Priority	Allows the application to specify the queue number and priority.
Comparison Key 1	Defines the components of the first comparison key used to enable range checking of messages.
Comparison Key 2	Defines the components of the second comparison key used to enable range checking of messages.
Order Key	Contains the information required to provide selection of messages by FIFO, Minimum Value, or Maximum Value.
Key Value for Offset "OTMQ_SOURCE"	Special case for setting key offset as "OTMQ_SOURCE", the source QSpace name and queue name should be specified in these 2 fields.

[Table 1-15](#) specifies the valid values that can be applied to the arguments in this part of the `Select_Queue` structure:

Table 1-15 Queue and Priority

Field	Values	Description
QSpace	QSpace Name	Specifies the QSpace name to be searched.
Queue	Queue Name	Specifies the queue name to be searched. The queue name can be any message queue for which the application has read access. The queue name can be obtained from the <code>q_attached</code> argument of the <code>tpqattach</code> function or <code>qname</code> of the <code>tpqlocate</code> function. A char of 0 for this argument specifies the application's primary queue. The queue must be attached before do this except MRQ, if MRQ is not attached, this api will attach the MRQ first
Priority		Specifies the priority, using either an integer between 0 and 99 inclusive or a variable. (Using the direct interger value is the preferred method of specifying priority.) This argument also accepts the following predefined constants which are set by the application. When the priority is set as 0, it will read priority 0 messages only.

Table 1-15 Queue and Priority

Field	Values	Description
	OTMQ_PRI_AN Y	Read priority 1 before reading priority 0 messages.
	OTMQ_PRI_P0	Read priority 0 messages only.
	OTMQ_PRI_P1	Read priority 1 messages only.

Table 1-16 specifies the arguments and valid values that can be applied to this part of the `selection_array_component_tp` structure:

Table 1-16 Comparison Keys

Field	Values	Description
Offset		Contains a value that specifies where the information to be compared begins inside the message. The following predefined constants apply:
	n	User message byte number (0 relative).
	OTMQ_CLASS	Class of the message.
	OTMQ_TYPE	Type of the message.
	OTMQ_SOURCE	Source queue address of the message.
	OTMQ_CORRELATION_ID	Correlation ID of the message. May be used for <code>key_1_offset</code> or <code>key_2_offset</code> but not both. If this symbol is specified, the <code>Size</code> field must be set to <code>OTMQ_CORRELATION_ID_SIZE</code> (or 32 bytes).
	OTMQ_SEQUENCE_NUMBER	Message sequence number acquired from the OTMQ Status Buffer. If this symbol is specified, the <code>Size</code> field must be set to <code>OTMQ_SEQUENCE_NUMBER_SIZE</code> (or 8 bytes).
Size		Specifies data type of the key to be compared. Note: For the special case of offset "OTMQ_SOURCE", the size can be any of following values except "0".

Table 1-16 Comparison Keys

Field	Values	Description
	0	Disable use of key.
	1	Byte (8 bits).
	2	Word (16 bits).
	4	int32 (32 bits).
	OTMQ_SEQUENCE_NUMBER_SIZE	8 bytes
	OTMQ_CORRELATION_ID_SIZE	32 bytes
Value	n	Contains the value for message field comparison field that is formatted as an integer of 32 bits.
oper		Relational operator comparison.
	OTMQ_OPER_EQ	Message field = value.
	OTMQ_OPER_NEQ	Message field <> value.
	OTMQ_OPER_GTR	Message field > value.
	OTMQ_OPER_LT	Message field < value.
	OTMQ_OPER_GTRE	Message field > or = value.
	OTMQ_OPER_LTE	Message field < or = value.

The Order Key part contains variables described in the following table:

Table 1-17 Order Key

Field	Values	Description
Offset		Byte offset of the message field. The offset variable contains a value that specifies where the information to be compared begins inside the message.
	n	User message byte number (0 relative).(only support memory queue)
	OTMQ_CLASS	Class of the message.
	OTMQ_TYPE	Type of the message.
	OTMQ_CORRELATION_ID	Correlation ID of the message. If this symbol is specified, the Size field must be set to OTMQ_CORRELATION_ID_SIZE (or 32 bytes).
	OTMQ_SEQUENCE_NUMBER	Message sequence number acquired from the OTMQ Status Buffer. If this symbol is specified, the Size field must be set to OTMQ_SEQUENCE_NUMBER_SIZE (or 8 bytes).
Size		Size of the comparison. The size variable specifies the data type of the key to be compared.
	0	Disable use of key.
	1	Byte.
	2	Word.
	4	int32 (32 bits).
	OTMQ_SEQUENCE_NUMBER_SIZE	8 bytes
	OTMQ_CORRELATION_ID_SIZE	32 bytes
Order		Order operator. The order variable specifies the sequence in which the select process is to be performed.

Table 1-17 Order Key

Field	Values	Description
	OTMQ_ORDER_FIFO	1,Read priority 1 before reading priority 0 messages 2,SQL SYNTAX: priority DESC 3,First pending.
	OTMQ_ORDER_MIN	1,Minimum value of all pending. 2,SQL SYNTAX: (offset is OTMQ_CLASS) class ASC, priority ASC 3,Read the last matched message
	OTMQ_ORDER_MAX	1,Maximum value of all pending. 2,SQL SYNTAX: (offset is OTMQ_CLASS) class DESC, priority ASC 3,Read the last matched message

Correlation ID

The correlation ID is a 32-byte user-defined identifier associated with a message. If OTMQ_CORRELATION_ID is supplied as the value for either the key_1_offset or key_2_offset field, the correlation ID value is used to match messages with the specified correlation ID. Since there is a single correlation ID per message, OTMQ_CORRELATION_ID should only be specified for one of the comparison keys; specifying the correlation ID for both keys results in a TPEINVAL error.

If OTMQ_CORRELATION_ID is supplied as the value for the order_offset field, messages with the specified correlation ID are returned in the order specified by the order_order field.

Sequence Number

The message sequence number is a unique value for each message. The sequence number is stored in the PSB. Applications should acquire the message sequence number from the PSB and not modify it in any way.

Note: An application may specify only one of the two keys to select by correlation identifier or by sequence number.

num_masks

Supplies the number of records in the selection array. This argument allows a minimum of 1 record to a maximum of 256 records in the selection array.

index_handle

Receives a variable containing the index handle for the selection mask as follows:

* The high-order word contains OTMQ_BY_MASK.

* The low-order word contains the index to the selection array.

The `index_handle` is passed as the `sel_filter` argument of `TPQCTL` in `tpdeqplus`, `tpqgetmsga`, and `tpqcancelselect` functions. OTMQ implementations offer up to 32767 index handles.

Return Value(s)

Upon failure, `tpqsetselect()` sets `tperrno` to one of the following values.

[TPEINVAL]

Invalid arguments were given.

[TPENOENT]

Cannot access the qspace because it is not available.

[TPETIME]

This error code indicates that either a timeout has occurred.

[TPESYSTEM]

An Oracle Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[TPEOS]

An operating system error has occurred.

tpqcancelselect()

Name

`tpqcancelselect()`—Releases the selection array and index handle associated with a previously generated selection mask.

Synopsis

```
#include <atmi.h>
#include <tmqentry.h>
```



```
#include <tmqreturn.h>
int tpqcancelselect (index_handle)
```

Description

Releases the selection array and index handle associated with a previously generated selection mask. An `index_handle` and associated selection mask are created using the `tpqsetselect` function. When the selection mask is used with asynchronous read requests, this function also cancels any pending `tpqgetmsga` requests that use the referenced `index_handle`.

To use `tpqcancelselect()`, you must first invoke `tpqattach()`.

[Table 1-18](#) lists supported `tpqcancelselect()` arguments:

Table 1-18 tpqcancelselect() Arguments

Argument	Data Type	Mechanism	Prototype	Access
<code>index_handle</code>	<code>int</code>	Refence	<code>int*</code>	passed

`index_handle`

Returned by `tpqsetselect`.

Return Value(s)

Upon failure, `tpqcancelselect()` sets `tperrno` to one of the following values.

[TPEINVAL]

Invalid arguments were given.

[TPENOENT]

Cannot access the qspace because it is not available.

[TPETIME]

This error code indicates that either a timeout has occurred.

[TPESYSTEM]

An Oracle Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[TPEOS]

An operating system error has occurred

tpqreadjrn()

Name

tpqreadjrn()—Reads a message from an OTMQ local group journal.

Synopsis

```
#include <atmi.h>
#include <tmqentry.h>
#include <tmqreturn.h>
int tpqreadjrn (qspace, qname, ctl, type, odata, olen, timeout)
```

Description

Reads a message from an OTMQ journal. Before using `tpqreadjrn`, you must first invoke `tpqattach()`.

[Table 1-19](#) lists supported `tpqreadjrn` arguments:

Table 1-19 tpqreadjrn Arguments

Argument	Data Type	Mechanism	Prototype	Access
qspace	char	reference	char *	passed
qname	char	reference	char *	passed
ctl	TPQCTL	Reference	TPQCTL*	passed/returned
type	int	Reference	int	passed
odata	char	reference	char *	returned
olen	long	reference	long *	returned
timeout	long	reference	long	passed

qspace

Supplies the queue space name as the filter to read the journal messages.

For SAF/DLQ/DLJ journal queue, queue space is the source queue space of the journal message that the client wants to read. For PCJ/DQF journal, queue space is the target queue space of the journal message that the client wants to read.

qname

Supplies the name of the permanent queue as the filter to read journal message.

For SAF/DLQ/DLJ journal queue, the queue name is the source queue of the journal messages that the client wants to read. For PCJ/DQF journal, queue name is the target queue name of the journal messages that the client wants to read

Control Parameter**flags**

For `tpqreadjrn()`, `ctl->flags` must contain OTMQ| TPQREADJRN.

type

Supplies the type of the journal to read. The values only be validate as:

SAF_HANDLE

DQF_HANDLE

DLQ_HANDLE

DLJ_HANDLE

PCJ_HANDLE

odata

Receives the contents of the message retrieved from the selected message recovery journal. This argument contains either the name of a memory region or a message handle where OTMQ writes.

olen

Supplies the size of the buffer (in bytes) for messages

timeout

Supplies the maximum amount of time the `tpqreadjrn` function waits for a message to arrive before returning control to the application. If the timeout occurs before a message arrives, the status code `OTMQ_TIMEOUT` is returned. Specifying 0 as the timeout value sets the timeout to the default value of 30 seconds.

Return Value(s)

Upon failure, `tpqreadjrn()` sets `tperrno` to one of the following values. Otherwise, the successfully read messages when `tpqreadjrn()` returns. you can use `tperrno()` and `tpqstrerror()` to get the detail OTMQ error number and detail string error message

[TPEINVAL]

Invalid arguments were given.

[TPENOENT]

Cannot access the qspace because it is not available.

[TPEOTYPE]

Either the type and subtype of the dequeued message are not known to the caller; or, TPNOCHANGE was set in flags and the type and subtype of *data do not match the type and subtype of the dequeued message. In either case, *data, its contents, and *len are not changed. When the call is made in transaction mode and this error occurs, the transaction is marked abort-only, and the message remains on the queue.

[TPETIME]

This error code indicates that either a timeout has occurred.

[TPEBLOCK]

A blocking condition exists and TPNOBLOCK was specified.

[TPGOTSIG]

A signal was received and TPSIGRSTRT was not specified.

[TPEPROTO]

tpdeqplus() was called improperly. There is no effect on the queue or the transaction.

[TPESYSTEM]

An Oracle Tuxedo system error has occurred. The exact nature of the error is written to a log file. There is no effect on the queue.

[TPEOS]

An operating system error has occurred. There is no effect on the queue.

[TPEDIAGNOSTIC]

Dequeuing a message from the specified queue failed. The reason for failure can be determined by the diagnostic value returned via ctl structure.

Diagnostic:

The following diagnostic values are returned during the dequeuing of a message:

[QMEINVAL]

An invalid flag value was specified.

[QMEBADRMID]

An invalid resource manager identifier was specified.

[QMESYSTEM]

A system error has occurred. The exact nature of the error is written to a log file.

[QMEOS]

An operating system error has occurred.

[QMEABORTED]

The operation was aborted. When executed within a global transaction, the global transaction has been marked rollback-only. Otherwise, the queue manager aborted the operation.

[QMEPROTO]

A dequeue was done when the transaction state was not active.

[QMEBADQUEUE]

An invalid or deleted queue name was specified.

[QMENOMSG]

No message was available for dequeuing. Note that it is possible that the message exists on the queue and another application process has read the message from the queue. In this case, the message may be put back on the queue if that other process rolls back the transaction.

[QMBADPRIORITY]

Invalid priority value used for receive.

[QMNOTDCL]

Process has not been attached to OTMQ.

tpqshowpending()

Name

`tpqshowpending()`—Requests the number of pending messages for a list of selected queues.

Synopsis

```
#include <atmi.h>
#include <tmqentry.h>
#include <tmqreturn.h>
int tpqshowpending ( qspace, count, in_q_list, out_pend_list)
need to follow the style of others API
```

Description

Requests the number of pending messages for a list of selected queues. To use the `tpqshowpending` function, specify the number of message queues for which you want to obtain a pending message count and the list of queue names for which you want to obtain a pending message count. The value returned by this function contains the total number of messages in each queue.

To use `tpqshowpending()`, you must first invoke `tpqattach()`.

Notes: You must allocate the `out_pend_list` array.

The number of queue that need to be listed is depend on "count", you must insure the array of queue list is properly allocated.

[Table 1-20](#) lists supported `tpqshowpending()` arguments.

Table 1-20 `tpqshowpending()` Arguments

Argument	Data Type	Mechanism	Prototype	Access
<code>qspace</code>	<code>char</code>	reference	<code>char *</code>	passed
<code>count</code>	<code>int</code>	reference	<code>int</code>	passed
<code>in_q_list</code>	<code>char</code>	reference	<code>char **</code>	passed
<code>out_pend_list</code>	<code>long</code>	reference	<code>long*</code>	passed/returned

`qspace`

Supplies the queue space name for enqueue the message. The max length is 15

`count`

Supplies the number of queue entries in the `in_q_list` argument (the number of indexes in the array). The maximum allowed value is 32,000.

`In_q_list`

Supplies an array containing the queue name for which the pending message count is requested

`Out_pend_list`

Receives the pending message count for each selected queue.

Return Value(s)

Upon failure, `tpqshowpending()` sets `tperrno` to one of the following values. You can use `tpqerrno()` and `tpqstrerror()` to get the detail OTMQ error number and detail string error message

[`TPEINVAL`]

Invalid arguments were given.

[`TPENOENT`]

Cannot access the `qspace` because it is not available.

[TPETIME]

This error code indicates that either a timeout has occurred.

[TPESYSTEM]

An Oracle Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[TPEOS]

An operating system error has occurred.

tpqgetmsga()

Name

tpqgetmsga()—Requests asynchronous notification of a message arrival.

Synopsis

```
#include <atmi.h>
#include <tmqentry.h>
#include <tmqreturn.h>
int tpqgetmsga (qspace, qname, ctl, data, len, uact, uact_param, uflag,
flags)
```

Description

Requests asynchronous notification of a message arrival. In OTMQ, the `tpqgetmsga` function is implemented using unsolicited message of Tuxedo. When calling `tpqgetmsga`, it registers its own UNSOL message handler. When a message arrives in that queue that fits the selection condition of `tpqgetmsga`, the message being dequeued will be sent through an UNSOL message to the application. Also if an UNSOL message has arrived, if the application setup its own message handling callback function, or an flag, the user callback function will be called, or the flag will be set to "1", to inform the application that an message has been dequeued by `tpqgetmsga`.

Since UNSOL message is the infrastructure of `tpqgetmsga`, to call this API, user must NOT set the NOTIFY to "IGNORE" in UBB. Also for some limitation, "SIGNAL" of notification mode is not supported on OpenVMS.

If a queue has been sent a recoverable message, the receiver program can confirm receipt of the message using the `tpqconfirmmsg` function. The `tpqconfirmmsg` function enables the successfully delivered message to be deleted from the message recovery system.

To use `tpqgetmsga()`, you must first invoke `tpqattach()`.

Table 1-21 lists supported `tpqgetmsga()` arguments:

Table 1-21 `tpqgetmsga()` Arguments

Argument	Data Type	Mechanism	Prototype	Access
<code>qspace</code>	<code>char</code>	reference	<code>char *</code>	passed
<code>qname</code>	<code>char</code>	reference	<code>char *</code>	passed
<code>ctl</code>	<code>TPQCTL</code>	reference	<code>TPQCTL *</code>	passed
<code>data</code>	<code>char</code>	reference	<code>char **</code>	Passed/returned
<code>len</code>	<code>long</code>	reference	<code>long *</code>	Passed/returned
<code>uact</code>	<code>long</code>	reference	<code>long *</code>	Passed
<code>uact_param</code>	<code>long</code>	reference	<code>long *</code>	Passed
<code>uflag</code>	<code>long</code>	reference	<code>long *</code>	Passed
<code>flags</code>	<code>long</code>	reference	<code>long</code>	Passed

`qspace`

Supplies the queue space name for enqueue the message. The max length is 15

`qname`

Supplies the queue name for enqueue the message, the max length is 127

`ctl`

The `TPQCTL` structure

`data`

For static buffer-style messaging, receives the name of a memory region where OTMQ writes the contents of the retrieved message.

`len`

Supplies the size of the buffer (in bytes) for static message buffers.

`uact`

user action

`uact_param`

user action parameters

uflag

Supplies the int value for the flag number to be set when the `tpqgetmsga` function completes. When the `tpqgetmsga` function executes, it clears this flag. If this argument value is not supplied, no flag is used.

flags

The flags supported by `tpdeqplus` must be set as `TPNOTIME`

Return Value(s)

Upon failure, `tpqgetmsga()` returns -1 and sets `tperrno` to indicate the error condition. you can use `tpqerrno()` and `tpqstrerror()` to get the detail OTMQ error number and detail string error message.

`tpqgetmsga()` sets `tperrno` to one of the following values upon failure. (Unless otherwise noted, failure does not affect caller transaction, if one exists.)

[TPEINVAL]

Invalid arguments were given.

[TPENOENT]

Cannot access the qspace because it is not available.

[TPETIME]

This error code indicates that either a timeout has occurred.

[TPESYSTEM]

An Oracle Tuxedo system error has occurred. The exact nature of the error is written to a log file. There is no effect on the queue.

[TPEOS]

An operating system error has occurred. There is no effect on the queue.

[TPEDIAGNOSTIC]

Dequeuing a message from the specified queue failed. The reason for failure can be determined by the diagnostic value returned via `ctl` structure.

Diagnostic:

The following diagnostic values are returned during the dequeuing of a message:

[QMEINVAL]

An invalid flag value was specified.

[QMEBADRMID]

An invalid resource manager identifier was specified.

[QMENOTOPEN]

The resource manager is not currently open.

[QMEBADMSGID]

An invalid message identifier was specified for dequeuing.

[QMESYSTEM]

A system error has occurred. The exact nature of the error is written to a log file.

[QMEOS]

An operating system error has occurred.

[QMEABORTED]

The operation was aborted. When executed within a global transaction, the global transaction has been marked rollback-only. Otherwise, the queue manager aborted the operation.

[QMEPROTO]

A dequeue was done when the transaction state was not active.

[QMEBADQUEUE]

An invalid or deleted queue name was specified.

[QMENOMSG]

No message was available for dequeuing. Note that it is possible that the message exists on the queue and another application process has read the message from the queue. In this case, the message may be put back on the queue if that other process rolls back the transaction.

[QMBADPRIORITY]

Invalid priority value used for receive.

[QMNOTDCL]

Process has not been attached to OTMQ.

tpqcancelget()

Name

`tpqcancelget()`—Cancels all pending `tpqgetmsga` requests that match the value specified in the `sel_filter` argument.

Synopsis

```
#include <atmi.h>
#include <tmqentry.h>
#include <tmqreturn.h>
int tpqcancelget (qspace, sel_filter , flags)
```

Description

Cancels all pending `tpqgetmsga` requests that match the value specified in the `sel_filter` argument. When a pending `tpqgetmsga` request is canceled, the OTMQ Status Block (PSB) delivery status is set to `OTMQ__CANCEL` and the specified action routine is queued. The `tpqcancelget` function waits until completion to allow for proper synchronization between the `tpqcancelget` function and the request for `tpqgetmsga` functions.

Any outstanding `tpqgetmsga` function requests are canceled by the `tpqexit` function or at image exit.

To use `tpqcancelget()`, must first invoke `tpqattach()`.

Notes: `tpqcancelget` can cancel those `tpqgetmsga` which set priority ≥ 0 but `sel_filter == NULL` by setting its parameter `sel_filter` to `(0 | 0)`.

`tpqcancelget` with "sel_filter" being set as `(OTMQ_PQ_PRI | x)` CANNOT cancel those `tpqgetmsga` which set "priority" to x and "sel_filter" set to NULL.

Table 1-22 lists supported `tpqcancelget()` arguments:

Table 1-22 tpqcancelget () Arguments

Argument	Data Type	Mechanism	Prototype	Access
qspace	char	reference	char *	passed
sel_filter	long	reference	long	passed
flags	long	reference	long	passed

qspace

Supplies the queue space whose requests asynchronous notification of a message arrival.

sel_filter

Supplies the criteria that enables the application to selectively cancel outstanding `tpqgetmsga` requests. For a description of the `sel_filter` argument, see the `tpdeqplus` function. For a description of how to create a complex selection filter, see the `tpqsetselect` function.

Flags

If `tpqcancelget()` is within a transaction and the `TPNOTRAN` flag is not set, the api goes into transaction mode.

The following is a list of valid flags:

TPNOTRAN

If the caller is in transaction mode and this flag is set, then the event publishing is not made on behalf of the caller's transaction. A caller in transaction mode that sets this flag is still subject to the transaction timeout (and no other) when posting events. If the event posting fails, the caller's transaction is not affected.

TPNOREPLY

Informs `tpqcancelget()` not to wait for the EventBroker to process all subscriptions for topic before returning. When **TPNOREPLY** is set, regardless of whether `tpqpublish()` returns successfully or not. When the caller is in transaction mode, this setting cannot be used unless **TPNOTRAN** is also set.

TPNOBLOCK

The topic is not published if a blocking condition exists. If such a condition occurs, the call fails and `tperrno` is set to **TPEBLOCK**. When **TPNOBLOCK** is not specified and a blocking condition exists, the caller blocks until the condition subsides or a timeout occurs (either transaction or blocking timeout).

TPNOTIME

This flag signifies that the caller is willing to block indefinitely and wants to be immune to blocking timeouts. Transaction timeouts may still occur.

Return Value(s)

Upon failure, `tpqcancelget()` returns -1 and sets `tperrno` to indicate the error condition. You can use `tpqerrno()` and `tpqsterror()` to get the detail OTMQ error number and detail string error message

`tpqcancelget()` sets `tperrno` to one of the following values upon failure.

[TPEINVAL]

Invalid arguments were given.

[TPEOENT]

Cannot access the `qspace` because it is not available.

[TPETIME]

This error code indicates that either a timeout has occurred.

[TPESYSTEM]

An Oracle Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[TPEOS]

An operating system error has occurred.

tpqerrno()

Name

tpqerrno()—Gets OTMQ system call errno.

Synopsis

```
#include <atmi.h>
#include <tmqentry.h>
#include <tmqreturn.h>
Int tpqerrno(void);
```

Description

tpqerrno() is used to retrieve the error code of OTMQ system call.

Return Value(s)

Upon success, tpqstrerror() returns a pointer to a string that contains the error message text.

If err is an invalid error code, tpqstrerror() returns a NULL.

tpqexit()

Name

tpqexit()—Terminates all attachments between the application and the OTMQ queue service.

Synopsis

```
#include <atmi.h>
#include <tmqentry.h>
#include <tmqreturn.h>
int tpqexit(void);
```

Description

Terminates all attachments between the application and the OTMQ queue service. All pending messages in queues which are not unlimited and multi-resource queues are discarded. To retain messages in permanently active queues, call tpdetachq with option TUXMSGQ_NOFLUSH_Q before calling tpqexit.

Return Value(s)

Upon failure, tpqcancelget() returns -1 and sets tperno to indicate the error condition.

[TPEPROTO]

`tpqexit()` was called improperly.

[TPESYSTEM]

An Oracle Tuxedo system error has occurred. The exact nature of the error is written to a log file. There is no effect on the queue.

[TPESVCERR]

A service routine encountered an error in `tpqexit`. If either `SVCTIMEOUT` in the `UBBCONFIG` file or `TA_SVCTIMEOUT` in the `TM_MIB` is non-zero, `TPESVCERR` is returned when a service timeout occurs.

tpqstrerror()

Name

`tpqstrerror()`—Gets Oracle Tuxedo Message Queue error message string details.

Synopsis

```
#include <atmi.h>
#include <tmqentry.h>
#include <tmqreturn.h>
Void tpqstrerror(int tpqerrno);
```

Description

`tpqstrerror()` is used to retrieve the text of an error message from `TMQ_CAT`. `err` is the error code set in `tpqerrno` when an OTMQ system function call returns a -1 or other failure value.

You can use the pointer returned by `tpqstrerror()` as an argument to `userlog()` or the UNIX function `fprintf()`.

[Table 1-23](#) lists `tpqstrerror()` supported arguments:

Table 1-23 tpqstrerror() Arguments

Argument	Data Type	Mechanism	Prototype	Access
<code>tpqerrno</code>	<code>int</code>	reference	<code>int</code>	passed

tpqerrno()

Gets OTMQ system call `errno`.

tpqstrerror()

Return Value(s)

Upon success, `tpqstrerror()` returns a pointer to a string that contains the error message text.

If `err` is an invalid error code, `tpqstrerror()` returns a `char *`.

Oracle Tuxedo Message Queue Command Reference

Table 1 Oracle Tuxedo Message Queue UBB Commands

Name	Description
<code>buildqclient</code>	Used to construct an OTMQ client module.
<code>buildqserver</code>	Used to construct an OTMQ load module which can run as an Oracle Tuxedo application server.
<code>ConvertQSPACE</code>	Utility used to upgrade/migrate existing /Q Qspace to OTMQ Qspace.
<code>tmqadmin</code>	Queue manager administration program

buildqclient

Name

`buildqclient`—Used to construct an OTMQ client module.

Synopsis

```
buildqclient [ -v ] [ { -r rmname | -w } ] [ -o name ] [ -f firstfiles ] [ -l lastfiles ]
```

Description

buildqclient is used to construct an OTMQ client module. The command combines the files supplied by the -f and -l options with the standard Oracle Tuxedo ATMI libraries to form a load module. The load module is built by buildqclient using the default C language compilation command defined for the operating system in use. The default C language compilation command for the UNIX system is the cc command described in UNIX system reference manuals.

It takes the following options:

-v

Specifies that buildqclient should work in verbose mode. In particular, it writes the compilation command to its standard output.

-w

Specifies that the client is to be built using the workstation libraries. The default is to build a native client if both native mode and workstation mode libraries are available. This option cannot be used with the -r option.

-r rmname

Specifies the resource manager associated with this client. The value rmname must appear in the resource manager table located in \$TUXDIR/udataobj/RM. Each line in this file is of the form:

```
rmname:rmstructure_name:library_names.
```

Using the rmname value, the entry in \$TUXDIR/udataobj/RM is used to include the associated libraries for the resource manager automatically and to set up the interface between the transaction manager and resource manager properly. Other values can be specified as they are added to the resource manager table. If the -r option is not specified, the default is that the client is not associated with a resource manager.

-o name

Specifies the filename of the output load module. If not supplied, the load module is named a.out.

-f firstfiles

Specifies one or more user files to be included in the compilation and link edit phases of buildqclient first, before the Oracle Tuxedo ATMI libraries and OTMQ libraries. If more than one file is specified, filenames must be separated by white space and the entire list must be enclosed in quotation marks. This option may be specified multiple times. The CFLAGS environment variable, described below, should be used to include any compiler options and their arguments.

-l lastfiles

Specifies one or more user files to be included in the compilation and link edit phases of buildqclient last, after the Oracle Tuxedo ATMI libraries and OTMQ libraries. If more than one file is specified, filenames must be separated by white space and the entire list must be enclosed in quotation marks. This option may be specified multiple times.

Environment Variables

TUXDIR

buildqclient uses the environment variable TUXDIR to find the Oracle Tuxedo ATMI libraries and include files to use during compilation of the client process.

CC

buildqclient normally uses the default C language compilation command to produce the client executable. The default C language compilation command is defined for each supported operating system platform and is defined as cc(1) for UNIX system. In order to allow for the specification of an alternate compiler, buildqclient checks for the existence of an environment variable named CC. If CC does not exist in the buildqclient environment, or if it is the string "", buildqclient will use the default C language compiler. If CC does exist in the environment, its value is taken to be the name of the compiler to be executed.

CFLAGS

The environment variable CFLAGS is taken to contain a set of arguments to be passed as part of the compiler command line. This is in addition to the command line option "-I\${TUXDIR}/include" passed automatically by buildqclient. If CFLAGS does not exist in buildqclient's environment, or if it is the string "", no compiler command line arguments are added by buildqclient.

LD_LIBRARY_PATH (UNIX systems)

The environment variable LD_LIBRARY_PATH indicates which directories contain shared objects in addition to the Oracle Tuxedo system shared objects. Some UNIX systems require different environment variables: for HP-UX systems, use the SHLIB_PATH environment variable; for AIX, use LIBPATH.

LIB (Windows NT systems)

Indicates a list of directories within which to find libraries. A semicolon (;) is used to separate the list of directories.

buildqserver

Name

`buildqserver`—Used to construct an OTMQ load module which can run as a Tuxedo application server.

Synopsis

```
buildqserver [-s { @filename | service[,service . . . ] [:func]| :func } ]  
[-v] [-o outfile] [-f firstfiles] [-l lastfiles] [-r rname] [-t]
```

Description

`buildqserver` is used to construct an OTMQ load module which can run as a Tuxedo application server. The command combines the files supplied by the `-f` and `-l` options with the standard server main routine and the standard OTMQ libraries to form a load module. The load module is built by the `cc` command, which `buildqserver` invokes.

It takes the following options:

-v

Specifies that `buildqserver` should work in verbose mode. In particular, it writes the compilation command to its standard output.t.

-o outfile

Specifies the name of the file the output load module is to have. If not supplied, the load module is named `SERVER`.

-f firstfiles

Specifies one or more user files to be included in the compilation and link edit phases of `buildqserver` first, before the Oracle Tuxedo ATMI libraries and OTMQ libraries. If more than one file is specified, filenames must be separated by white space and the entire list must be enclosed in quotation marks. This option may be specified multiple times. The `CFLAGS` environment variable, described below, should be used to include any compiler options and their arguments.

-l lastfiles

Specifies one or more user files to be included in the compilation and link edit phases of `buildqserver` last, after the Oracle Tuxedo ATMI libraries and OTMQ libraries. If more than one file is specified, filenames must be separated by white space and the entire list must be enclosed in quotation marks. This option may be specified multiple times.

-r rmname

Specifies the resource manager associated with this server. The value `rmname` must appear in the resource manager table located in `$TUXDIR/udataobj/RM`. Each line in this file is of the form:

```
rmname:rmstructure_name:library_names.
```

Using the `rmname` value, the entry in `$TUXDIR/udataobj/RM` is used to include the associated libraries for the resource manager automatically and to set up the interface between the transaction manager and resource manager properly. Other values can be specified as they are added to the resource manager table. If the `-r` option is not specified, the default is to use the null resource manager.

-s { @filename | service[,service...][:func] | :func }

Specifies the names of services that can be advertised when the server is booted. Service names (and implicit function names) must be less than or equal to 127 characters in length. An explicit function name (that is, a name specified after a colon) can be up to 128 characters in length. Names longer than these limits are truncated with a warning message. When retrieved by `tadmin` or `TM_MIB`, only the first 15 characters of a name are displayed. All functions that can be associated with a service must be specified with this option. In the most common case, a service is performed by a function that carries the same name; that is, the `x` service is performed by function `x`. For example, the following specification builds the associated server with services `x`, `y`, and `z`, each to be processed by a function of the same name: `"-s x,y,z"`.

In other cases, a service (or several services) may be performed by a function of a different name. The following specification builds the associated server with services `x`, `y`, and `z`, each to be processed by the function `abc`: `"-s x,y,z:abc"`. Spaces are not allowed between commas. Function name is preceded by a colon. In another case, the service name may not be known until run time. Any function that can have a service associated with it must be specified to `buildqserver`. To specify a function that can have a service name mapped to it, put a colon in front of the function name. For example, the following specification builds the server with a function `pqr`, which can have a service association. `tpadvertise` could be used to map a service name to the `pqr` function. `"-s :pqr"`. A filename can be specified with the `-s` option by prefacing the filename with the `'@'` character. Each line of this file is treated as an argument to the `-s` option. You may put comments in this file. All comments must start with the `'#'` character. This file can be used to specify all the functions in the server that may have services mapped to them. The `-s` option may appear several times. Note that services beginning with the `'.'` character are reserved for system use, and `buildqserver` will fail if the `-s` option is used to include such a service in the server.

-t

Specifies multithreading. If you want your servers to be multithreaded, this option is mandatory. If this option is not specified and you try to boot a server with a configuration file in which the value of MAXDISPATCHTHREADS is greater than 1, a warning message is printed in the user log and the server reverts to single-threaded operation.

The purpose of this option is to prevent an administrator from trying to boot, as a multithreaded server, a server that is not programmed in a thread-safe manner.

Environment Variables

TUXDIR

buildclient uses the environment variable TUXDIR to find the Oracle Tuxedo ATMI libraries and include files to use during compilation of the client process.

CC

buildqclient normally uses the default C language compilation command to produce the client executable. The default C language compilation command is defined for each supported operating system platform and is defined as cc(1) for UNIX system. In order to allow for the specification of an alternate compiler, buildqclient checks for the existence of an environment variable named CC. If CC does not exist in the buildqclient environment, or if it is the string "", buildqclient will use the default C language compiler. If CC does exist in the environment, its value is taken to be the name of the compiler to be executed.

CFLAGS

The environment variable CFLAGS is taken to contain a set of arguments to be passed as part of the compiler command line. This is in addition to the command line option "-I\${TUXDIR}/include" passed automatically by buildqclient. If CFLAGS does not exist in buildqclient's environment, or if it is the string "", no compiler command line arguments are added by buildqclient.

LD_LIBRARY_PATH (UNIX systems)

The environment variable LD_LIBRARY_PATH indicates which directories contain shared objects in addition to the Oracle Tuxedo system shared objects. Some UNIX systems require different environment variables: for HP-UX systems, use the SHLIB_PATH environment variable; for AIX, use LIBPATH.

LIB (Windows NT systems)

Indicates a list of directories within which to find libraries. A semicolon (;) is used to separate the list of directories.

ConvertQSPACE

Name

ConvertQSPACE—Queue manager administration program.

Synopsis

```
ConvertQSPACE -d device -s qspace -i ipckey -h
```

Description

Upgrade to OTMQ to have many new benefits, saying faster enqueue/dequeue speed, asynch communication, recovery message storage, online and/or offline trade, pub/sub mode, auto failed message handling, and so on.

This utility allows you to upgrade existed /Q Qspace to OTMQ Qspace, so that the data can be migrated smoothly too. The existing /Q queues is mapped to unlimited queues to keep consistency with old behavior.

Notes: Pay close attention to the following:

- The QMCONFIG environment variable must be configured as /Q device.
- The given /Q and OTMQ devices must not be used by other processes at the same time
- The conversion is read-only for /Q device. So if the process is interrupted unexpectedly, the user can re-do it until success.
- The transaction cannot be migrated since the TMS server is changed.

It takes the following options:

-d device

Specifies the name of new OTMQ device. If existed, we will try to reuse it unless it has been already opened.

-s qspace

Specifies the name of the /Q Qspace to be converted to OTMQ Qspace.

-i ipckey

Specifies the ipc key for the new OTMQ Qspace, which cannot be the same as the original /Q Qspace ipc key.

-h

Prints this usage.

tmqadmin

Name

tmqadmin—Queue manager administration program.

Synopsis

```
[QMCONFIG=<device>] tmqadmin [<device>]
```

Description

With the commands listed in this entry, tmqadmin supports the creation, inspection, and modification of message queues. The universal device list (UDL) maps the physical storage space on a machine on which the Oracle Tuxedo ATMI system is run. An entry in the UDL points to the disk space in which the queues and messages of a queue space are stored. The name of the device (file) on which the UDL resides (or will reside) for the queue space may be specified either as a command line argument or via the environment variable QMCONFIG. If both are specified, the command option is used.

As a system-provided command, tmqadmin does not undergo normal initialization, so it does not pick up the value of ULOGPFX from the UBBCONFIG file. As a result, any log entries generated by tmqadmin commands are written to the current working directory. This is corrected by setting and exporting the ULOGPFX environment variable to the pathname of the directory in which the userlog is located.

tmqadmin uses the greater than sign (>) as a prompt. Arguments are entered separated by white space (tabs and/or spaces). Arguments that contain white space may be enclosed within double quotes; if an argument enclosed within double quotes contains a double quote, the internal double quote must be preceded with a backslash. Commands prompt for required information if it is not given on the command line. A warning message is displayed and the prompt shown again, if a required argument is not entered. Commands do not prompt for information on optional parameters.

A user can exit the program by entering q or <CTRL-d> when prompted for a command. Output from a command may be terminated by pressing BREAK; the program then prompts for the next command. Hitting return when prompted for a command repeats the previously executed command, except after a break.

Note that there is no way to effectively cancel a command once you press RETURN; hitting BREAK only terminates output from the command, if any. Therefore, be sure that you type a command exactly as you intended before pressing RETURN.

Output from tmqadmin commands is paginated according to the pagination command in use (see the `paginate` subcommand below).

When tmqadmin is initially entered, no queue space is opened. To create a queue space, run `qspacecreate`; to open it, run `qopen`. The `qaborttrans`, `qclear`, `qclose`, `qchangeprio`, `qchangequeue`, `qchangetime`, `qchangeexptime`, `qcommittrans`, `qchange`, `qcreate`, `qdeletemsg`, `qinfo`, `qlist`, `qprinttrans` and `qset` commands can be executed only when a queue space is open.

It takes the following commands:

- [General Commands](#)
- [Queue Space Commands](#)
- [Queue Commands](#)
- [Message Commands](#)
- [Transaction Commands](#)

General Commands

- !** `shellcommand`
Escapes to shell and execute shellcommand.
- !!**
Repeats previous shell command.
- #** [`text`]
Lines beginning with # are comment lines and are ignored.
- <CR>**
Repeats the last command.
- echo** (`e`) [`{off | on}`]
Echoes input command lines when set to on. If no option is given, the current setting is toggled, and the new setting is printed. The initial setting is off.
- help** (`h`) [`{command | all}`]
Prints help messages. If a command is specified, the abbreviation, arguments, and description for that command are printed. The `all` argument causes a description of all commands to be displayed.

If no arguments are specified on the command line, the syntax of all commands is displayed.

paginate (page) [{off|on}]

Paginates output. If no option is given, the current setting is toggled, and the new setting is printed. The initial setting is on, unless either standard input or standard output is a non-terminal device. Pagination may be turned on only when both standard input and standard output are terminal devices.

The default paging command is the pager indigenous to the native operating system environment. The command `pg`, for example, is the default command on the UNIX operating system. The shell environment variable `PAGER` may be used to override the default command used for paging output.

quit (q)

Terminates the session.

verbose (v) [{off | on}]

Produces output in verbose mode. If no option is given, the current setting is toggled, and the new setting is printed. The initial setting is off.

Queue Space Commands

chdl [dlindex [newdevice]]

Changes the name for a universal device list entry. The first argument is the index of the device on the universal device list that is to be changed (device indexes are returned by `lidl`). The program prompts for it if it is not provided on the command line.

The second argument is the new device name. If a device name is not provided on the command line, the program prints the current device name and then prompts for a new one. The name is limited to 64 characters in length. Use this command cautiously; files and data are not accessible via the old name after the device name is changed. For more information about printing the Universal Device List (UDL) and Volume Table of Contents (VTOC), see *Administering an Oracle Tuxedo Application at Run Time*.

crdl [device [offset [size]]]

Creates an entry in the universal device list. Note: The first entry in the device list must correspond to the device that is referenced by `QMCONFIG` and must have an offset of 0. If arguments are not provided on the command line, the program prompts for them.

The arguments are the device name, the block number at which space may begin to be allocated, and the number of physical pages (disk sectors) to be allocated. More than one extent can be allocated to a given file. You can, for example, allocate `/app/queues/myspace 0 500`, and then allocate `/app/queues/myspace 1000 500`, for a total of 1000 blocks allocated with blocks 500 through 999 not being

used. Several blocks from the first device entry are used by the device list and table of contents. Up to 25 entries may be created on the device list.

For OTMQ, this command can support large size. The max value of this command is 2147483647, the basic unit is block size.

dsdl [-y] [dlindex]

Destroys an entry found in the universal device list. The dlindex argument is the index on the universal device list of the device that is to be removed from the device list. If it is not provided on the command line, the program prompts for it. Entry 0 cannot be removed until all VTOC files and other device list entries are destroyed. (Because entry 0 contains the device that holds the device list and table of contents, destroying it also destroys these two tables.) VTOC files can be removed only by removing the associated entities (for example, by destroying a queue space that resides on the device). The program prompts for confirmation unless -y is specified.

ipcrm [-f] [-y] [queue_space_name]

Removes the IPC data structures used for the specified queue space. If a queue space name is not provided on the command line, the program prompts for one. If the specified queue space is open in tmqadmin, it will be closed. ipcrm knows all IPC resources used by the queue space and is the only way that the IPC resources should be removed.

tmqadmin ensures that no other processes are attached to the queue space before removing the IPC resources. The -f option can be specified to force removal of IPC resources even if other processes are attached. This command prompts for confirmation before execution if the -f option is specified, unless the -y option is specified. All non-persistent messages in the specified queue space are permanently lost when this command completes successfully.

ipcs [queue_space_name]

Lists the IPC data structures used for a queue space, if any (none may be used if the queue space is not opened by any process). If a queue space name is not provided on the command line, the program prompts for one.

lidl [dlindex]

Prints the universal device list. For each device the following is listed: the index, the name, the starting block, and the number of blocks on the device. In verbose mode, a map is printed that shows free space (starting address and size of free space). If dlindex is specified, only the information for that device list entry is printed.

livtoc

Prints information for all VTOC table entries. The information printed for each entry includes the name of the VTOC table, the device on which it is found, the offset of the VTOC table from the beginning of the device and the number of pages allocated for that table. There are a maximum of 100 entries in the VTOC.

qaddext [**queue_space_name** [**pages**]]

Adds an extent to the queue space. The queue space must not be active (no processes can be attached to the queue space). If a queue space name and the number of additional physical pages to allocate for the queue space are not specified on the command line, the program prompts for them. If the specified queue space is open in tmqadmin, it will be closed. The number of physical pages is rounded down to the nearest multiple of four pages (see qspacecreate for clarification and examples). Space is allocated from extents defined in the UDL associated with the QMCONFIG device. Each new queue space extent uses an additional entry in the VTOC (a maximum of 100 entries are available). The queue manager names the extents such that they can be identified quickly and associated with the queue space. All non-persistent messages in the specified queue space are permanently lost when this command completes successfully.

qclose

Closes the currently open queue space. All non-persistent messages in the specified queue space are permanently lost when this command completes successfully.

qopen [**queue_space_name**]

Opens and initializes the internal structures for the specified queue space. If a queue space is not specified on the command line, the program prompts for it. If a queue space is already open in tmqadmin, it is closed.

qsize [-A actions] [-H handles] [-C cursors] [-O owners] [-Q tmp_queues] [-f filter_memory] [-n nonpersistent_msg_memory[b,B]] [-o overflow_memory][pages [queues [transactions [processes [messages]]]]]

Computes the size of shared memory needed for a queue space with the specified size in pages, queues, (concurrent) transactions, processes, and (queued) messages. If the values are not provided on the command line, the program prompts for them. The number of system semaphores needed is also printed. Valid values for the remaining options are described in the qspacecreate option.

qspacechange (qspch) [-A actions] [-C cursors] [-H handles] [-O owners] [-Q temp_queues] [-f filter_memory] [-T first_temp_queue_No.] [-N queue_alias_file] [-n nonpersistent_msg_memory[b,B]] [-o

```
overflow_memory] [queue_space_name [ipkey [trans [procs [messages
[errorq [inityn [blocking]]]]]]]
```

Changes the parameters for a queue space. The queue space must not be active (that is, no processes can be attached to it). If the required information is not provided on the command line, the program prompts for it. Valid values are described in the `qspacecreate` section of this page. If the specified queue space is open in `tmqadmin`, it is closed. To add new extents, `qaddext` must be used. The number of queues cannot be modified.

```
qspacecreate (qspc) [-A actions] [-n nonpersistent_msg_memory[b,B]]
[-o overflow_memory][-C max cursor][-O max owner][-f maximum filter
memory size][-H max handle] [-Q qNum] [-T first temp queue] [-L max
linkdriver table][-R max linkdirver route table][-N alias
file][queue_space_name [ipkey [pages [queues [trans [procs [messages
[errorq [inityn [blocking [SAF_and_DQF_queue [PCJ_journaling
[Dead_Letter_Journal ]]]]]]]]]]]]
```

Creates a queue space for queued messages. If not provided on the command line, the program prompts for the following information: the queue space name, the `ipkey` for the shared memory segment and semaphore; number of physical pages to allocate for the queue space; the number of queues; the number of concurrent transactions; the number of processes concurrently attached to the queue space; the number of messages that may be queued at one time; the name of an error queue for the queue space; whether or not to initialize pages on new extents for the queue space; the blocking factor for doing queue space initialization and warm start disk input/output; SAF and DQF queue to create or not (by default, it is); PCJ Journaling to enable or not (by default, it is); and Dead letter Journal to enable or not (by default it is).

The number of physical pages requested is rounded down to the nearest multiple of four pages. For example, a request of 50 pages results in a memory allocation of 48 pages, and a request of 52 pages results in a memory allocation of 52 pages. The error queue is used to hold messages that have reached the maximum number of retries (they are moved from their original queue to the error queue). The administrator is responsible for ensuring that this queue is drained.

The number of physical pages allocated must be large enough to hold the overhead for the queue space (one page plus one page per queue). If the initialization option is specified as 'y' or 'Y,' the space used to hold the queue space is initialized and this command may run for a while. In verbose mode, a period (.) is printed to the standard output after completing initialization of each 5% of the queue space.

If the initialization option is not turned on but the underlying device is not a character special device, the file will be initialized if it is not already the size specified for the extent (that is, the file will be grown to allocate the specified

space). When reading and writing blocks during creation of the queue space and during warm start (restart of the queue space), the size of input and output operations will be calculated as a multiple of the disk page size as specified by the blocking factor.

SAF (Store And Forward) and DQF (Destination Queue File) are message recovery journals respectively on the local system and on remote system. If SAF and DQF queues are created by default, the recoverable delivery feature using SAF and DQF are enabled.

PCJ (Post Confirmation Journal) stores successfully confirmed recoverable messages. PCJ is created by default as permanent active queue.

DLJ ((Dead Letter Journal) provides disk storage for messages that could not be stored for automatic recovery. DLJ is created by default as permanent active queue.

- A actions

Specifies the number of additional actions that the Queuing Services component can handle concurrently.

When a blocking operation is encountered and additional actions are available, the blocking operation is set aside until it can be satisfied. After setting aside the blocking operation, another operation request can be handled. When the blocking operation completes, the action associated with the operation is made available for a subsequent operation. An operation fails if a blocking operation is requested and cannot be immediately satisfied and there are no actions available.

The system reserves actions equivalent to the number of processes that can attach to a queue space so that each queue manager process may have at least one blocking action. Beyond the system-reserved number of blocking actions, the administrator may configure the system to be able to accommodate additional blocking actions beyond the reserve.

If the - A actions option is not specified, the default is zero. If the - A option is not specified, the program does not prompt for it.

-n nonpersistent_msg_memory

Specifies the size of the area to reserve in shared memory for non-persistent messages for all queues in the queue space. The size may be specified in bytes (b) or blocks (B), where the block size is equivalent to the disk block size. The [bB] suffix is optional and, if not specified, the default is blocks. The [bB] max value is 2147483647, If the -n option is not specified, the memory size defaults to zero (0).

Also, if the -n option is not specified, the program does not prompt for it. If the value is specified in bytes (b) for `nonpersistent_msg_memory`, the system

divides the specified value by the number of bytes per page (page size is equivalent to the disk page size), rounds down the result to the nearest integer, and allocates that number of pages of memory. For example, assuming a page size of 1024 bytes (1KB), a requested value of 2000b results in a memory allocation of 1 page (1024 bytes), and a requested value of 2048b results in a memory allocation of 2 pages (2048 bytes).

Requesting a value less than the number of bytes per page results in an allocation of 0 pages (0 bytes). If the value is specified in blocks (B) for `nonpersistent_msg_memory` and assuming that one block of memory is equivalent to one page of memory, the system allocates the same value of pages. For example, a requested value of 50B results in a memory allocation of 50 pages. If the `nonpersistent_msg_memory` for a queue space is zero (0), no space is reserved for non-persistent messages.

In this case, attempts to enqueue a non-persistent message fail. Persistent and non-persistent storage are not interchangeable. If a non-persistent message cannot be enqueued due to an exhausted or fragmented memory area, the enqueueing operation fails, even if there is sufficient persistent storage for the message. If a persistent message cannot be enqueued due to an exhausted or fragmented disk, the enqueueing operation fails, even if there is sufficient non-persistent storage for the message.

- o overflow_memory

Specifies the size of the memory area to reserve in shared memory to accommodate peek load situations where some or all of the allocated shared memory resources are exhausted. The memory size is specified in bytes. Additional objects will be allocated from this additional memory on a first-come-first-served basis.

When an object created in the additional memory is closed or destroyed, the memory is released for subsequent overflow situations. If the `- o overflow_memory` option is not specified, the default is zero. If the `- o` option is not specified, the program does not prompt for it. This additional memory space may yield more objects than the configured number, but there is no guarantee that additional memory is available for any particular object at any given point in time. Currently, only actions, handles, cursors, owners, temporary queues, timers, and filters use the overflow memory.

-C

Specifies the maximum cursor.

-f

Specifies the maximum filter memory size. If not specified, the default value 32768 is used.

-H

Specifies the maximum handle.

-Q

Specifies the maximum temporary queues. If not specified, the default value 100 is used.

-T

Specifies the first temporary queue. If not specified, the default value 200 is used.

-L

Specifies the maximum linkdriver table. If not specified, the default value 200 is used.

-R

Specifies the maximum linkdriver route table. If not specified, the default value 200 is used.

qspacedestroy (qspds) [-f] [-y] [queue_space_name]

Destroys the named queue space. If not provided on the command line, the program will prompt for it. If the specified queue space is open in tmqadmin, it will be closed. By default, an error is returned if processes are attached to the queue space or if requests exist on any queues in the queue space. See the qdestroy command for destroying queues that contain requests. The -f option can be specified to "force" deletion of all queues, even if they may have messages or processes are attached to the queue space. This command prompts for confirmation before proceeding unless the -y option is specified. All non-persistent messages in the specified queue space are lost when this command completes successfully

qspacelist [queue_space_name]

Lists the creation parameters for the queue space. If it is not specified on the command line, the program will prompt for it. If a queue space name is not entered, the parameters for the currently open queue space are printed. (An error occurs if a queue space is not open and a value is not entered.) In addition to printing the values for the queue space (as set when creating the queue space with qspacecreate or when they were last changed with qspacechange), this command shows the sizes for all queue space extents. It also shows the amount of system-reserved memory as well as the total amount of configured shared memory. The amount of memory allocated for shared memory resources may not match the amount requested when the amount of memory is requested in bytes (b); see the -n nonpersistent_msg_memory option in qspacecreate for clarification and examples.

Queue Commands

```
qchange [-d default_delivery_policy] [-n mhigh,mlow,cmd][-e
default_relative_expiration_time][-t default_queue_property] [-o
owner] [-a PERM_ACTIVE] [-c confirm_style] [-f][queue_name [qorder
[out-of-order [retries [delay [high [low [cmd]]]]]]]]]
```

Modifies a queue in the currently open queue space. The required arguments may be given on the command line or the program will prompt for them. These are the queue name, whether out-of-order enqueueing is allowed (not allowed, top of queue, or before a specified `msgid`); the number of retries and delay time in seconds between retries; and the high and low limits for execution of a threshold command and the threshold command itself for persistent messaging.

The out-of-order values are `none`, `top`, and `msgid`. Both `top` and `msgid` may be specified, separated by a comma. The threshold values are used to allow for automatic execution of a command when a threshold is reached for persistent messages.

The high limit specifies when the command is executed. The low limit must be reached before the command is executed again when the high limit is reached. For example, if the limits are 100 and 50 messages, the command is executed when 100 messages are on the queue, and it is not executed again until the queue is drained down to 50 messages and is filled again to 100 messages. The queue capacity can be specified in bytes or blocks used by the queue (number followed by a `b` or `B` suffix), percentage of the queue space used by the queue (number followed by a `%`), or total number of messages on the queue (number followed by an `m`). The threshold type for the high and low threshold values must be the same. It is optional whether or not the type is specified on the low value, but if specified, it must match the high value type.

The message (`m`) suffix spans both persistent and non-persistent messages. The other threshold suffixes apply only to persistent messages. Use the `-n` option to specify threshold values for non-persistent messages. When specified on the command line, the threshold command should be enclosed in double quotation marks if it contains white space.

The retry count indicates how many times a message can be dequeued and the transaction rolled back, causing the message to be put back on the queue. A delay between retries can also be specified. When the retry count is reached, the message is moved to the error queue defined for the queue space. If no error queue has been defined, the message is dropped.

The queue ordering values for the queue cannot be changed. Low-priority messages are dequeued after every ten messages, even if the queue still contains high-priority messages. The `-d` option specifies the default delivery policy for the

queue. The valid values for the `-d` option are `persist` and `nonpersist`. When the default delivery policy is `persist`, enqueued messages with no explicitly specified delivery mode are delivered using the persistent (disk-based) delivery method. When the policy is `nonpersist`, enqueued messages with no explicitly specified delivery mode are delivered using the non-persistent (in memory) delivery method.

If the `-d` option is not specified, the system does not prompt for information and the default delivery policy is unchanged. When the default delivery policy is modified, the delivery quality of service is not changed for messages already in the queue. If the queue being modified is the reply queue named for any messages currently in the queue space, the reply quality of service is not changed for those messages as a result of changing the default delivery policy of the queue. If a non-persistent message cannot be enqueued due to an exhausted or fragmented memory area, the enqueueing operation fails, even if there is sufficient persistent storage for the message.

If a persistent message cannot be enqueued due to an exhausted or fragmented disk, the enqueueing operation fails, even if there is sufficient non-persistent storage for the message. If the amount of memory reserved for non-persistent messages in a queue space is zero (0), no space is reserved for non-persistent messages. (For more information, see `qspacecreate` and `qspacechange` for specifying the non-persistent message memory area.) In this case, attempts to enqueue a non-persistent message fail. This includes messages with no specified delivery quality of service for which the target queue has a default delivery policy of `nonpersist`.

The `-n` option specifies the threshold values used for automatic execution of a command when a non-persistent storage area threshold is reached. The `nhigh` limit specifies when the command `ncmd` is executed. The `nlow` limit must be reached before the command will be executed again when the `nhigh` limit is reached. If the `-n` option is specified, the `nhigh`, `nlow`, and `ncmd` values must all be supplied, or the command fails. The `ncmd` value may be specified as an empty string. If the `-n` option is not specified, the program does not prompt for information.

The memory capacity (amount of non-persistent data in the queue) can be specified as one of the following threshold types: bytes (b), blocks (B), or percentage (number followed by %). The threshold type for the `nhigh` and `nlow` values must be the same. For example, if `nhigh` is set to 100%, then `nlow`, if specified, must also be specified as a percentage. The threshold type of the `nlow` value is optional. If the `-n` option is not specified, the default threshold values for non-persistent messaging are unchanged. If `ncmd` contains white space, it must be enclosed in

double quotation marks. The `m` suffix of the `[. . . [high[low[cmd]]] . . .]` thresholds applies to all messages in a queue, including both persistent and non-persistent messages, and therefore is not available with `nhigh` and `nlow`. The `[. . . [high[low[cmd]]] . . .]` thresholds specified without the `-m` suffix apply to persistent (disk-based) messages only.

The `-e default_relative_expiration_time` option sets an expiration time for all messages enqueued to the queue that do not have an explicitly specified expiration time. The expiration time may be either a relative expiration time or none. When the expiration time is reached and the message has not been dequeued or administratively deleted, the message is removed from the queue, all resources associated with the message are reclaimed by the system, and statistics are updated.

If the expiration time is before the message availability time, the message is not available for dequeuing unless either the availability or expiration time is changed so that the availability time is before the expiration time. In addition, these messages are removed from the queue at expiration time even if they were never available for dequeuing.

If a message expires during a transaction, the expiration does not cause the transaction to fail. Messages that expire while being enqueued or dequeued within a transaction are removed from the queue when the transaction ends. There is no notification when a message has expired.

If the `-e` option is not specified, the default expiration time of the queue is not changed. When the queue expiration time is modified using `qchange`, the expiration times for messages already in the queues are not modified. If the `-e` option is not specified, the program does not prompt for it. The format of a relative `default_relative_expiration_time` is `+seconds` where `seconds` is the number of seconds from the time that the queue manager successfully completes the operation to the time that the message expires. A value of zero (0) indicates immediate expiration.

The value of `default_relative_expiration_time` may also be set to the string `none`. The `none` string indicates that messages that are enqueued with no explicit expiration time will not expire unless an expiration time is explicitly assigned to them.

The valid values for the `-t "type"` option are `"PQ"`, `"SQ"` and `"MRQ"`. The default value is `"unlimited queue"`.

The `-o "owner"` option is used for secondary queues. And defines the primary queue with which this queue is to be associated. The valid values for it is primary queue name.

The valid values for the `-a "active"` option are "Y" or "N". The default value is "N". This is to define if the queue is permanent active. If yes, then the queue always can receive and store message unless the quota is exceeded. If not, the queue cannot receive and store message before it is attached, and will report invalid queue in the sender side. This feature doesn't make impact for MRQ and unlimited queue (inherited from /Q).

-c [confirm style]

Set queue property of confirm style, the valid values for `-c "confirm style"` option are EO (means confirm un-order) and II (implicit confirm).

-f:

Some queue name are reserved for internal use, which will be prevented by `qcreate: 74-76,90-100,150-199,4000-6000`, unless the `-f` parameter is specified.

When create these internal queue name by `qcreate -f` or MIB, warning is given in ULOG.

-e[exptime]

The default expiration time specified by `-e` option does not take effect for recoverable message (SAF/DQF/CONF). Instead, the SAF/DQF queue default expiration time (none by default) takes effect.

-q [nhigh[m/b/a/n],nlow,ncmd]

If the `-q` option is specified, the `nhigh`, `nlow`, and `ncmd` values must all be supplied, or the command fails. The `ncmd` value may be specified as an empty string.

qclear [-y]

Clears attached client information on a queue and resets the queue to unattached status. The queue is specified using the `qset` command.

This command prompts for confirmation unless the '-y' option is specified.

qcreate (qcr) [-d default_delivery_policy] [-n mhigh,mlow,ncmd][-e default_relative_expiration_time][-t default_queue_property] [-o owner] [-a PERM_ACTIVE] [-c confirm_style] [-f][queue_name [qorder [out-of-order [retries [delay [high [low [cmd]]]]]]]]]

Creates a queue in the currently open queue space. The required arguments may be given on the command line or the program will prompt for them. These are the queue name, the queue ordering (fifo or lifo, by expiration time, by priority, by time); whether out-of-order enqueueing is allowed (not allowed, top of queue, before a specified msgid); the number of retries and delay time in seconds between retries; the high and low limits for execution of a threshold command; and the threshold command itself for persistent messages.

The queue ordering values are `fifo`, `lifo`, `priority`, `expiration`, and `time`. When specifying the queue ordering, the most significant sort value must be specified first, followed by the next most significant sort value, and so on; `fifo` or `lifo` can be specified only as the least significant (or only) sort value.

If neither `fifo` or `lifo` is specified, the default is `fifo` within whatever other sort criteria are specified. If `expiration` is specified, messages with no expiration time are dequeued after all messages with an expiration time. Multiple sort values may be specified separated by commas. The out-of-order values are `none`, `top`, or `msgid`. Both `top` and `msgid` may be specified, separated by a comma. The threshold values are used to allow for automatic execution of a command when a threshold is reached for persistent messages. The high limit specifies when the command is executed.

The low limit must be reached before the command will be executed again when the high limit is reached. For example, if the limits are 100 and 50 messages, the command will be executed when 100 messages are on the queue, and will not be executed again until the queue has been drained below 50 messages and has filled again to 100 messages. The queue capacity can be specified in bytes or blocks used by the queue (number followed by a `b` or `B` suffix), percentage of the queue space used by the queue (number followed by a `%`), or total number of messages on the queue (number followed by an `m`).

The threshold type for the high and low threshold values must be the same. The message (`m`) suffix spans both persistent and non-persistent messages. The other threshold suffixes apply only to persistent messages. Use the `-n` option to specify threshold values for non-persistent messages. It is optional whether or not the type is specified on the low value, but if specified, it must match the high value type. When specified on the command line, the threshold command should be enclosed in double quotation marks if it contains white space.

The retry count indicates how many times a message can be dequeued and the transaction rolled back, causing the message to be put back on the queue. A delay between retries can also be specified. When the retry count is reached, the message is moved to the error queue defined for the queue space. If an error queue has not been defined, the message is dropped. Low-priority messages are dequeued after every ten messages, even if the queue still contains high-priority messages.

`qcreate` supports the following options:

-d

Specifies the default delivery policy for the queue. The valid values for the `-d` option are `persist` and `nonpersist`. When the default delivery policy is `persist`, enqueued messages with no explicitly specified delivery mode are delivered using the persistent (disk-based) delivery method. When the policy is `nonpersist`,

enqueued messages with no explicitly specified delivery mode are delivered using the non-persistent (in memory) delivery method. If the `-d` option is not specified, the system does not prompt for information and the default delivery policy for the queue is `persist`.

When the default delivery policy is modified, the delivery quality of service is not changed for messages already in the queue. If a non-persistent message cannot be enqueued due to an exhausted or fragmented memory area, the enqueueing operation fails, even if there is sufficient persistent storage for the message. If a persistent message cannot be enqueued due to an exhausted or fragmented disk, the enqueueing operation fails, even if there is sufficient non-persistent storage for the message.

If the amount of memory reserved for non-persistent messages in a queue space is zero (0), no space is reserved for non-persistent messages. (See `qspacecreate` and `qspacechange` for information on specifying the non-persistent message memory area.) In this case, attempts to enqueue a non-persistent message fail. This includes messages with no specified delivery quality of service for which the target queue has a default delivery policy of `nonpersist`.

-n

Specifies the threshold values used for automatic execution of a command when a non-persistent storage area threshold is reached. The `nhigh` limit specifies when the command `ncmd` is executed. The `nlow` limit must be reached before the command will be executed again when the `nhigh` limit is reached. If the `-n` option is specified, the `nhigh`, `nlow`, and `ncmd` values must all be supplied, or the command fails. The `ncmd` value may be specified as an empty string. If the `-n` option is not specified, the program does not prompt for information.

The memory capacity (amount of non-persistent data in the queue) can be specified as one of the following threshold types: bytes (b), blocks (B), or percentage (number followed by %). The threshold type for the `nhigh` and `nlow` values must be the same. For example, if `nhigh` is set to 100%, then `nlow`, if specified, must also be specified as a percentage. The threshold type of the `nlow` value is optional. If the `-n` option is not specified, the default threshold values are used (100% for `nhigh` and 0% for `nlow`) and `ncmd` is set to " ".

If `ncmd` contains white space, it must be enclosed in double quotation marks. The `m` suffix of the [. . . [high[low[cmd]]] . . .] thresholds applies to all messages in a queue, including both persistent and non-persistent messages, and therefore is not available with `nhigh` and `nlow`. The [. . . [high[low[cmd]]] . . .] thresholds specified without the `-m` suffix apply to persistent (disk-based) messages only.

-e default_relative_expiration_time

Sets an expiration time for all messages enqueued to the queue that do not have an explicitly specified expiration time. The expiration time may be either a relative expiration time or none. When the expiration time is reached and the message has not been dequeued or administratively deleted, the message is removed from the queue, all resources associated with the message are reclaimed by the system, and statistics are updated.

If the expiration time is before the message availability time, the message is not available for dequeuing unless either the availability or expiration time is changed so that the availability time is before the expiration time. In addition, these messages are removed from the queue at expiration time even if they were never available for dequeuing. If a message expires during a transaction, the expiration does not cause the transaction to fail. Messages that expire while being enqueued or dequeued within a transaction are removed from the queue when the transaction ends. There is no notification when a message has expired.

If the `-e` option is not specified, the default expiration time of the queue is set to none. When the queue's expiration time is modified using `qchange`, the expiration times for messages already in the queues are not modified. If the `-e` option is not specified, the program does not prompt for it. The format of a relative `default_relative_expiration_time` is `+seconds` where `seconds` is the number of seconds from the time that the queue manager successfully completes the operation to the time that the message expires. A value of zero (0) indicates immediate expiration. The value of `default_relative_expiration_time` may also be set to the string `none`. The `none` string indicates that messages that are enqueued with no explicit expiration time will not expire unless an expiration time is explicitly assigned to them.

-t "type"

The valid values for the `-t "type"` option are `PQ` (primary queue), `SQ` (secondary queue) and `MRQ` (`MRQ`). The default value is `"unlimited queue"`.

-o "owner"

is used for secondary queues. And defines the primary queue with which this queue is to be associated. The valid values for it is primary queue's name.

The valid values for the `-a "active"` option are `"Y"` or `"N"`.

The default value is `"N"`. This is to define if the queue is permanent active. If yes, then the queue always can receive and store message unless the quota is exceeded. If not, the queue cannot receive and store message before it is attached, and will report invalid queue in the sender side. This feature doesn't make impact for `MRQ` and unlimited queue (inherited from `/Q`).

c [confirm style]

Set queue property of confirm style, the valid values for `-c "confirm style"` option are `EO` (confirm un-order), `II` (implicit confirm).

-f:

Some queue name are reserved for internal use, which will be prevented by `qcreate: 74-76, 90-100, 150-199, 4000-6000`, unless the `-f` parameter is specified.

When create these internal queue name by `qcreate -f` or `MIB`, warning is given in `ULOG`.

-e[exptime]

The default expiration time specified by `-e` option does not take effect for recoverable message (`SAF/DQF/CONF`). Instead, the default expiration time of `SAF/DQF` queue (`none` by default) will take effect.

-q [nhigh[m/b/a/n],nlow,ncmd]

If the `-q` option is specified, the `nhigh`, `nlow`, and `ncmd` values must all be supplied, or the command fails. The `ncmd` value may be specified as an empty string.

qdestroy (qds) [{ -p | -f }] [-y] [queue_name]

Destroys the named queue. By default, an error is returned if requests exist on the queue or a process is attached to the queue space. The `-p` option can be specified to "purge" any messages from the queue and destroy it, if no processes are attached to the queue space. The `-f` option can be specified to "force" deletion of a queue, even if messages or processes are attached to the queue space; if a message is currently involved in a transaction the command fails and an error is written to the userlog. This command prompts for confirmation before proceeding unless the `-y` option is specified.

qinfo [queue_name]

Lists information for associated queue or for all queues. This command lists the following: the number of messages on the specified queue (or all queues if no argument is given); the amount of space used by the messages associated with the queue (both persistent and non-persistent); the number of messages being delivered persistently and non-persistently; the total number of messages in the specified queues, and the amount of space used by the persistent and non-persistent messages. In verbose mode, this command also lists the queue creation parameters for each queue, the default expiration for the queue (if any), the sort criteria, and the default delivery policy for the queue.

Message Commands

qchangeexp (qce) -y [newtime]

Changes the expiration time for messages on a queue. When a message expires, it is removed from the queue, all resources used by the message are reclaimed by the system, and the relevant statistics are updated. If the expiration time is before the message availability time, the message is not available for dequeuing unless either the availability or expiration time is changed so that the availability time is before the expiration time. In addition, these messages are removed from the queue at expiration time even if they were never available for dequeuing. If a message expires during a transaction, the expiration does not cause the transaction to fail. Messages that expire while being enqueued or dequeued within a transaction are removed from the queue when the transaction ends. There is no notification when a message has expired.

The queue for which an expiration time is set is selected using the `qset` command. Selection criteria for limiting the messages to be updated are set with the `qscan` command. If no selection criterion is set, all messages on the queue are changed. By default, a confirmation is requested before the expiration time is set. The `-y` option specifies no prompt for confirmation. The `newtime` value can be relative to either the current time, an absolute value, or none. If the `newtime` value is not provided on the command line, the program prompts for it. Messages enqueued by versions of the Oracle Tuxedo ATMI system that do not support message expiration cannot be modified to have an expiration time even when the queue manager responsible for changing the value supports message expiration. If messages affected by `qchangeexp` have been enqueued by one of these versions of the Oracle Tuxedo ATMI system, an error message indicates that some of the selected messages were not modified due to this limitation. A relative expiration time is relative to when the request arrives at the queue manager process. The format of a relative `newtime` is `+seconds` where `seconds` is the number of seconds from the time that the queue manager successfully completes the operation to the time that the message expires. If `seconds` is set to zero (0), messages expire immediately. An absolute expiration time is determined by the clock on the machine where the queue manager process resides. The format of an absolute `newtime` is `YY[MM[DD[HH[MM[SS]]]]]` as described in `qscan`. The value of `newtime` may also be set to the string `none`, which indicates that affected messages never expire.

qchangeprio (qcp) [-y] [newpriority]

Changes the priority for messages on a queue. The queue that is affected is set using the `qset` command and the selection criteria for limiting the messages to be updated are set using the `qscan` command.

If no selection criteria are set, all messages on the queue are changed: confirmation is requested before this is done unless the `-y` option is specified. It is recommended that the `qlist` command be executed to see what messages will be modified (this reduces typographical errors). The `newpriority` value specifies the new priority which will be used when the message(s) are forwarded for processing. It must be in the range 1 to 100, inclusive. If not provided on the command line, the program will prompt for it.

qchangequeue (qcq) [-y] [newqueue]

Moves messages to a different queue within the same queue space. The queue from which messages are moved is set using the `qset` command and the selection criteria for limiting the messages to be moved are set using the `qscan` command. If no selection criteria are set, all messages on the queue are moved: confirmation is requested before this is done unless the `-y` option is specified. It is recommended that the `qlist` command be executed to see what messages will be moved (this reduces typographical errors). The `newqueue` value specifies the name of the queue to which messages will be moved. If `newqueue` is not specified on the command line, the program prompts for it. The delivery quality of service of a message is not changed to match the default delivery policy of `newqueue`.

When messages with an expiration time are moved, the expiration time is considered an absolute expiration time in the new queue, even if it was previously specified as a relative expiration time.

qchangetime (qct) [-y] [newtime]

Changes the message availability time for messages on a queue. The queue is specified using the `qset` command. The selection criteria for limiting the messages to be updated are set using the `qscan` command.

If no selection criteria are set, all messages on the queue are changed: confirmation is requested before this is done unless the `-y` option is specified. It is recommended that the `qlist` command be executed to see what messages will be modified (this reduces typographical errors). The `newtime` value can be either relative to the current time or an absolute value. If not provided on the command line, the program will prompt for it. The format of a relative onetime is `+seconds` where `seconds` is the number of seconds from now that the message is to be executed (0 implies immediately). The format of an absolute `newtime` is `YY[MM[DD[HH[MM[SS]]]]]`, as described in `qscan`.

qdeletemsg (qdltm) [-y]

selection criteria for limiting the messages to be deleted are set using the `qscan` command. If no selection criteria are set, all messages on the queue are deleted: confirmation is requested before this is done. It is recommended that the `qlist`

command be executed to see what messages will be deleted (this reduces typographical errors). This command prompts for confirmation unless the `-y` option is specified.

qlist (`ql`)

Lists messages on a queue. The queue is specified using the `qset` command. The selection criteria for limiting the messages to be listed are set using the `qscan` command. If no selection criteria are set, all messages on the queue will be listed.

For each message selected, the message identifier is printed along with the message priority, the number of retries already attempted, message length, delivery quality of service, the quality of service for any replies, and the expiration time (if any). The message availability time is printed if one is associated with the message, or for messages that have a scheduled retry time (due to rollback of a transaction). The correlation identifier is printed if present and verbose mode is on.

```
qscan [{ [-t time1[-time2]] [-p priority1[-priority2]] [-m msgid] [-i
corrid][-d delivery_mode] [-e time1[-time2]] | none }
```

Sets the selection criteria used for the `qchangeprio`, `qchangequeue`, `qchangetime`, `qdeletemsg`, and `qlist` commands. An argument of `none` indicates no selection criteria; all messages on the queue will be affected. Executing this command with no argument prints the current selection criteria values.

When command line options give a value range (for example, `-t`, `-e`, or `-p`) the value range may not contain white space. The `-t` option can be used to indicate a time value or a time range. The format of `time1` and `time2` is:

YY[MM[DD[HH[MM[SS]]]]] specifying the year, month, day, hour, minute, and second. Units omitted from the date-time value default to their minimum possible values. For example, "7502" is equivalent to "750201000000." The years 00-37 are treated as 2000-2037, years 70-99 are treated as 1970-1999, and 38-69 are invalid. The `-p` option can be used to indicate a priority value or a priority range. Priority values are in the range 1 to 100, inclusive.

The `-m` option can be used to indicate a message identifier value, assigned to a message by the system when it is enqueued. The message identifier is unique within a queue and its value may be up to 32 characters in length. Values that are shorter than 32 characters are padded on the right with nulls (0x0). Backslash and non-printable characters (including white space characters such as space, newline, and tab) must be entered with a backslash followed by a two-character hexadecimal value for the character (for example, space is `\20`, as in "hello\20world").

The `-i` option can be used to indicate an correlation identifier value associated with a message. The identifier value is assigned by the application, stored with the

enqueued message, and passed on to be stored with any reply or error message response such that the application can identify responses to particular requests. The value may be up to 32 characters in length. Values that are shorter than 32 characters are padded on the right with nulls (0x0). Backslash and non-printable characters (including white space characters such as space, newline, and tab) must be entered with a backslash followed by a two-character hexadecimal value for the character (for example, space is `\20`, as in `my\20ID\20value`).

The valid values for the `-d delivery_mode` option are `persist` and `nonpersist`. This option specifies the delivery mode of messages selected by `qscan` so that an operator can take action based on the delivery method. The `-e` option can be used to indicate an expiration time or an expiration time range. The format of `time1` and `time2` is the same as `time1` and `time2` for the `-t` option.

qset [`queue_name`]

Sets the queue name that is used for the `qchangeprio`, `qchangequeue`, `qchangetime`, `qdeletemsg`, and `qlist` commands. Executing this command with no argument prints the current queue name.

Transaction Commands

qaborttrans (`qabort`) [`-y`] [`tranindex`]

Heuristically aborts the precommitted transaction associated with the specified transaction index, `tranindex`. If the transaction index is not specified on the command line, the program prompts for it. If the transaction is known to be decided and the decision was to commit, `qaborttrans` fails.

The index is taken from the previous execution of the `qprinttrans` command. Confirmation is requested unless the `-y` option is specified. This command should be used with care.

qcommittrans (`qcommit`) [`-y`] [`tranindex`]

Heuristically commits the precommitted transaction associated with the specified transaction index `tranindex`. The program will prompt for the transaction index if not specified on the command line. If the transaction is known to be decided and the decision was to abort, `qcommittrans` will fail. The index is taken from the previous execution of the `qprinttrans` command. Confirmation is requested unless the `-y` option is specified. This command should be used with care.

qprinttrans (`qpt`)

Prints transaction table information for currently outstanding transactions. The information includes the transaction identifier, an index used for aborting or committing transactions with `qaborttrans` or `qcommittrans`, and the transaction status.

Oracle Tuxedo Message Queue UBB Server Reference

Table 1 Oracle Tuxedo Message Queue UBB Servers

Name	Description
TuxMsgQLD	OTMQ Link Driver Server.
TuxMsgQ	Message Queue Manager.
TuxMQFWD	Message Queue Off-line trade driver.
TMQ_NA	Message Queue Naming Server.
TMS_TMQM	TMS server for OTMQ resource manager.
TMQ_EVT	TMQ event reporting process.
TMQFORWARDPLUS	Message Forwarding server.

TuxMsgQLD

TuxMsgQLD—OTMQ Link Driver Server

Synopsis

```
TuxMsgQLD SRVGRP="identifier" SRVID="number"
[CLOPT="[-A] [servopts options]
[-g grp_id][ -l //hostname:port][ -f config_filename]" ]
```

Description

The TuxMsgQLD (Link Driver Server) of OTMQ is ported from traditional OMQ Link Driver to achieve message level compatibilities between OTMQ and OMQ applications.

Also the OTMQ Link Driver Server provides the routing functionality like traditional OMQ but with limitations.

Configuration File

To configure this Link Driver Server, a new configuration file should be created. This configuration file should be under APPDIR. The configuration file is ported from %XGROUP and %ROUTE section of traditional OMQ configuration file. .

[Listing 3-1](#) an example of the Link Driver Server configuration file:

Listing 3-1 Link Driver Server Configuration File:

```
# Define cross-group connections with remote OMQ,
# only the remove OMQ group info should be listed here
%XGROUP
# Group   Group   Node/   Init-   Thresh   Buffer   Recon-   Window   Trans-   End-
# Name    Number  Host    iate    old      Pool    nect     Delay    Size (Kb)  port  point
GRP_11    11     host1.abc.com Y       -        -        30      10       250      TCP/IP 10001
GRP_12    12     host2.abc.com Y       -        -        30      10       250      TCP/IP 10002

%EOS
%ROUTE
#-----
# Target   Route-through
# Group    Group
#-----
      2         11
      3         12

%EOS
%END
```

%XGROUP Section

Following configuration attributes are mandatory for OTMQ to setup XGROUP connection to the remote OMQ group:

- "Group Name -- Remote OMQ group name by which the remote OMQ group is known to the local OTMQ group
- "Group Number -- Remote OMQ group number
- "Node/Host -- Network address of remote OMQ group
- "Endpoint -- The internet port number of the remote OMQ link listener process

Following attributes are optional. If not set, default values will be used:

- "Initiate -- "Y", "N" or "D". indicating whether connections to this node should be initiated (connect when local group startup) or whether connections to this node is enabled. Default is "N".
- "Reconnect -- Interval, in seconds, between reconnect attempts when this cross-group link is not connected. Default is 60.
- "Window Delay -- Delay, in seconds, that a sender must wait before using a new window when the receiver is congested. Default is 10.
- "Windows Size -- Maximum number of messages a group can send to another group before requesting permission to send more. Default is 250.
- "Transport: Network protocol stack used. Only "TCPIP" is supported.

Following attributes are not supported by OTMQ Link Driver Server, but to keep align with traditional OMQ XGROUP settings, just keep them here:

- "Buffer Pool -- Not supported by OTMQ for now.
- "Threshold -- Not supported by OTMQ for now.

%ROUTE

Following configuration attributes are mandatory for OTMQ to setup ROUTE info for the remote OMQ/OTMQ groups that can't be connected directly:

- "Target Group -- OMQ/OTMQ group for which traffic is being routed to.
- "Route-through Group -- OMQ group to which traffic for the target group will be routed through.

Note: Route-through Groups should be the OMQ groups that this OTMQ group has direct connection to (i.e. defined in the %XGROUP section).

Cannot define multiple routing entry for the same Target Group.

Limitations

Direct Connection

- Only support DISC and RTS UMA.

Routing

- Only support AK and NN modes.
- Only support DISC UMA.
- Only support MEM, DEQ and ACK DIPs when protocol exchange is involved more than once, such as sending message from OMQ to OMQ through OTMQ or from OTMQ to OTMQ through OMQ.

TuxMsgQ

TuxMsgQ—Message Queue Manager.

Synopsis

TuxMsgQ

SRVGRP="identifier"

SRVID="number" CLOPT=" [-A][servopts options] -- [-t timeout][-i interval]"

Description

The message queue manager is an Oracle Tuxedo system-supplied server that enqueues and dequeues messages on behalf of programs calling `tpenqplus()` and `tpdeqplus()`, respectively. The application administrator enables message enqueueing and dequeuing for the application by specifying this server as an application server in the `*SERVERS` section.

To configure default attach timeout for a Qspace, set `BLOCKTIME` property in the `*SERVICES` section for the service named `TuxMQATH[qspace]`, where `[qspace]` is the service name that `TuxMsgQ` advertised with routine `TuxMsgQ`.

The location, server group, server identifier and other generic server related parameters are associated with the server using the already defined configuration file mechanisms for servers. The following additional command-line option is available for customization.

It takes the following options:

-t timeout

Used to indicate the timeout to be used for queuing operations when not in transaction mode (for example, `tpenqueue()` / `tpenqplus()` or `tpdequeue()` / `tpdeqplus()` are called when the caller is not in transaction mode or with the `TPNOTRAN` flag). This value also has an impact on dequeue requests with the `TPQWAIT` option since the operation will timeout and an error will be sent back to the requester based on this value. If not specified, the default is 30 seconds.

A TuxMsgQ server is booted as part of an application to facilitate application access to its associated queue space; a queue space is a collection of queues.

Any configuration condition that prevents the TuxMsgQ from enqueueing or dequeuing messages will cause the TuxMsgQ to fail at boot time. The `SRVGRP` must have `TMSNAME` set to `TMS_QM`, and must have `OPENINFO` set to indicate the associated device and queue space name.

-i interval

To configure the sanity check interval in TuxMsgQ server, set "-i [interval]" in `CLOPT`. The "interval" number means the TuxMsgQ server will do sanity check per receiving this number of messages.

Example(s)

Listing 3-2 TuxMsgQ Example

```
*SERVERS
TuxMsgQ SRVGRP="TUXMSGQGRP" SRVID=1000 RESTART=Y GRACE=0
CLOPT="-s myqueue:TuxMsgQ"
```

See Also

[tpdeqplus\(\)](#), [tpenqplus\(\)](#), [servopts\(5\)](#), [TMQFORWARD\(\)](#), [UBBCONFIG\(5\)](#)

Setting Up an Oracle Tuxedo Application

Administering an Oracle Tuxedo Application at Run Time

Programming an Oracle Tuxedo ATMI Application Using C

TuxMQFWD

Name

TuxMQFWD—Message Queue Off-line trade driver.

Synopsis

```
TuxMQFWD
SRVGRP="identifier"
SRVID="number" CLOPT=" [-A][servopts options] -- [-f delay time][-t
timeout][-i idle time]"
```

Description

The message queue off-line trade driver is an Oracle Tuxedo system-supplied server.

The location, server group, server identifier and other generic server related parameters are associated with the server using the already defined configuration file mechanisms for servers. It takes the following options:

-f delay time

Parameter used to indicate the amount of time (in seconds) when off-line trade driver try again after the last time failure

-i idletime

Parameter used to indicate the amount of time (in seconds) that the server remains idle after draining the queue(s) that it is reading. A negative value indicates an amount of time in milliseconds. For example if you specify `-i -10`, the idle time will be 10 milliseconds.

If a value of zero is specified, a 1 second value is used, the server reads the queue(s) continually, which can be inefficient if the queues do not continually have messages. If a value of zero is specified, a 1 second value is used.

-t timeout

Parameter used to indicate the transaction timeout value (in seconds) used on `tpbegin()` for transactions that dequeue messages and forward them to application servers. If not specified, the default is 60 seconds.

Any configuration condition that prevents TuxMQFWD from dequeuing or forwarding messages will cause the server to fail to boot. These conditions include the following:

- The SRVGRP must have TMSNAME set to TMS_TMQM.
- OPENINFO must be set to indicate the associated device and queue name.

- The SERVER entry must not be part of an MSSQ set.
- REPLYQ must be set to N.
- The server must not advertise any services (TuxMQFWD is built out without any service, so -s option will be ignored). You cannot configure more than one TuxMQFWD process in a group.

Example(s)

```
TuxMQFWD
SRVGRP=QGRP1 SRVID=51 GRACE=0 RESTART=Y CONV=N MAXGEN=10
CLOPT="-- -i 1"
```

TMQ_NA

Name

TMQ_NA—Message Queue Naming Server

Synopsis

```
TMQ_NA
SRVGRP="identifier"
SRVID="number" CLOPT=" [-A][servopts options] -- [-g OTMQ group] [-r OMQ
group]"
```

Description

TMQ_NA is an OTMQ system server. It can provide naming and runtime binding of queue aliases to queue names. It supports the following options:

-g xxx
Parameter used to indicate this naming service is provided by OTMQ group xxx.

-r yyy
Parameter used to indicate this naming service is provided by OMQ group yyy

Note: the sequence of parameters determines which Naming server is primary and which one is backup,

Example(s)

```
TMQ_NA
SRVGRP=QGRP1 SRVID=51 GRACE=0 RESTART=Y CONV=N MAXGEN=10
CLOPT=" -- -g 1 "
```

TMS_TMQM

Name

TMS_TMQM—TMS server for OTMQ resource manager.

Synopsis

Description

OTMQ also provides a separate Tuxedo Transaction Management Server (TMS). TMS_TMQM should be configured in the TuxMsgQ and/or TuxMQFWD group.

Example(s)

```
*GROUPS
QGRP1
    LMID=L1 GRPNO=1 TMSNAME=TMS_TMQM TMSCOUNT=2
    OPENINFO="TUXEDO/TMQM:/dev/device2:myqueue"
```

TMQ_EVT

Name

TMQ_EVT—OTMQ event reporting process.

Synopsis

```
TMQ_EVT SRVGRP="identifier" SRVID="number"
    [CLOPT="[-A] [servopts options]
    [-- [-S] [-p poll-seconds] [-f control-file]]"]
```

Description

TMQ_EVT is an Oracle Tuxedo system provided server that processes event report message buffers from tppublish, and acts as an EventBroker to filter and distribute them.

Filtering and notification rules are stored in control-file, which defaults to `${APPDIR}/tmqevt.dat`. Control file syntax is defined in `EVENT_MIB(5)`; specifically, the attributes of the classes in `EVENT_MIB` can be set to activate subscriptions under the full range of notification rules.

It is possible to boot one or more secondary `TMQ_EVT` processes for increased availability. Additional servers must be booted with the `-S` command-line option, which indicates a "secondary" server.

When the `EVENT_MIB(5)` configuration is updated, the primary `TMQ_EVT` server writes to its control file. Secondary servers poll the primary server for changes and update their local control file if necessary. The polling interval is controlled by the `-p` option, and is 30 seconds by default.

Notes: If you are setting up an MP configuration that includes more than one release of the Oracle Tuxedo system and you want to run the `TMQ_EVT` server, you must run these servers on the node with the highest available release of the system.

`TMQ_EVT` must not delopoy with the same group of `TMS_TMQM`.

`TMQ_EVT` must run on an Oracle Tuxedo release 6.0 or later machine.

To migrate the primary `TMQ_EVT` server to another machine, the system administrator must provide a current copy of control-file. Each secondary `TMQ_EVT` server automatically maintains a recent copy.

If `tpqpublish()` is called in transaction mode, all `TMQ_EVT` server groups must have transactional capability (a TMS process).

The `TMQ_EVT` server environment variables must be set so that FML field tables and viewfiles needed for message filtering and formatting are available. They could be set in the machine's or server's environment file.

Example(s)

```
*SERVERS
TMQ_EVT SRVGRP=ADMIN1 SRVID=100 RESTART=Y MAXGEN=5 GRACE=3600
CLOPT="-A --"
TMQ_EVT SRVGRP=ADMIN2 SRVID=100 RESTART=Y MAXGEN=5 GRACE=3600
CLOPT="-A -- -S -p 120"
```

See Also

`tpqpublish()`, `tpqsubscribe()`, `EVENTS(5)`, `EVENT_MIB(5)`,

TMQFORWARDPLUS

Name

TMQFORWARDPLUS—Message Forwarding server.

Synopsis

```
TMQFORWARDPLUS SRVGRP="identifier" SRVID="number" REPLYQ=N CLOPT="  
[-A] [servopts options] -- -q queueName[,queueName...]  
[-t trantime ] [-i idletime] [-b timeout] [-e] [-d] [-n] [-f delay] "
```

Description

The message forwarding server is an Oracle Tuxedo system-supplied server that forwards messages that have been stored using `tpenqplus()` for later processing. The application administrator enables automated message processing for the application servers by specifying this server as an application server in the `*SERVERS` section.

The location, server group, server identifier and other generic server related parameters are associated with the server using the already defined configuration file mechanisms for servers. It supports the following options:

-q queueName[,queueName...]

Used to specify the names of one or more queues/services for which this server forwards messages. Queue and service names are strings limited to 15 characters. This option is required.

-t trantime

Used to indicate the transaction timeout value used on `tpbegin()` for transactions that dequeue messages and forward them to application servers. If not specified, the default is 60 seconds.

-i idletime

Used to indicate the amount of time (in seconds) that the server remains idle after draining the queue(s) that it is reading. A negative value indicates an amount of time in milliseconds. For example if you specify `-i -10`, the idle time will be 10 milliseconds.

If a value of zero is specified, the server will read the queue(s) continually, which can be inefficient if the queues do not continually have messages. If no value is specified, the default is 30 seconds.

-b timeout

Used to limit nontransaction block waiting time, in seconds, for a forwarded service to complete. The `-b` option can only be used with the `-f` option.

-e

Used to cause the server to exit if it finds no messages on the queue(s). This, combined with the threshold command associated with the queue(s), can be used to start and stop the `TMQFORWARDPLUS` server in response to fluctuations of messages that are enqueued.

-d

Used to cause messages that result in service failure and have a reply message (non-zero in length) to be deleted from the queue after the transaction is rolled back. That is, the original request message is deleted from the queue-not put back on the queue-if the service fails and a reply message (non-zero in length) is received from the server.

The reply message is enqueued to the failure queue, if one is associated with the message and the queue exists. If the message is to be deleted at the same time as the retry limit configured for the queue is reached, the original request message is put into the error queue.

-n

Used to cause messages to be sent using the `TPNOTRAN` flag. This flag allows for forwarding to server groups that are not associated with a resource manager.

-f delay

Used to cause the server to forward the message to the service instead of using `tpcall`. The message is sent such that a reply is not expected from the service. The `TMQFORWARDPLUS` server does not block waiting for the reply from the service and can continue processing the next message from the queue. To throttle the system such that `TMQFORWARDPLUS` does not flood the system with requests, the delay numeric value can be used to indicate a delay, in seconds, between processing requests; use zero for no delay.

Messages are sent to a server providing a service whose name matches the queue name from which the message is read. The message priority is the priority specified when the message is enqueued, if set. Otherwise, the priority is the priority for the service, as defined in the configuration file, or the default (50).

Messages are dequeued and sent to the server within a transaction. If the service succeeds, the transaction is committed and the message is deleted from the queue. If the message is associated with a reply queue, any reply from the service is enqueued to the reply queue, along with the returned `tprcode`. If the reply queue does not exist, the reply is dropped.

An application may be able to specify the quality of service for a reply to a message when the original message is enqueued. If a reply quality of service is not specified, the default delivery policy specified for the reply queue is used. Note that the default delivery policy is determined when the reply to a message is enqueued. That is, if the default delivery policy of the reply queue is modified between the time that the original message is enqueued and the reply to the message is enqueued, the policy used is the one in effect when the reply is finally enqueued.

If the service fails, the transaction is rolled back and the message is put back on the queue, up to the number of times specified by the retry limit configured for the queue. When a message is put back on the queue, the rules for ordering and dequeuing that applied when it was first put on the

queue are (in effect) suspended for delay seconds; this opens up the possibility, for example, that a message of a lower priority may be dequeued ahead of the restored message on a queue ordered by priority.

If the `-d` option is specified, the message is deleted from the queue if the service fails and a reply message is received from the server, and the reply message (and associated `tpurcode`) are enqueued to the failure queue, if one is associated with the message and the queue exists. If the message is to be deleted at the same time as the retry limit for the queue is reached, the original request message is put into the error queue.

Any configuration condition that prevents `TMQFORWARDPLUS` from dequeuing or forwarding messages will cause the server to fail to boot. These conditions include the following:

- The `SRVGRP` must have `TMSNAME` set to `TMS_TMQM`.
- `OPENINFO` must be set to indicate the associated device and queue name.
- The `SERVER` entry must not be part of an `MSSQ` set.
- `REPLYQ` must be set to `N`.
- The `-q` option must be specified in the command-line options.
- The server must not advertise any services (that is, the `-s` option must not be specified).

Example(s)

Listing 3-3 `TMQFORWARDPLUS` Example

```
*GROUPS # For Windows, :myqueue becomes ;myqueue
TUXMSGQGRP LMID=lmid GRPNO=1 TMSNAME=TMS_TMQM
OPENINFO="TUXEDO/TMQM:/dev/device:myqueue"
# no CLOSEINFO is required

*SERVERS # recommended values RESTART=Y GRACE=0
TMQFORWARDPLUS SRVGRP="TUXMSGQGRP" SRVID=1001 RESTART=Y GRACE=0
CLOPT=" -- -qservice1,service2" REPLYQ=N
```

See Also

[tpongplus\(\)](#), [servopts\(5\)](#), [UBBCONFIG\(5\)](#)

