Oracle® Tuxedo Message Queue (OTMQ)

Programming Guide

12c Release 2 (12.1.3)

December 2014

Oracle® Tuxedo Message Queue (OTMQ) Programming Guide, 12c Release 2 (12.1.3)

Oracle Tuxedo Message Queue Programming Guide

Oracle Tuxedo Message Queue PAMS Programming Guide

CHAPTER 1

Oracle Tuxedo Message Queue Programming Guide

This chapter contains the following topics:

Programmer Tasks

Oracle Tuxedo Message Queue (OTMQ) provides the following features to Oracle Tuxedo application programmers:

- A set of application programming interfaces to enqueue messages for subsequent process. The queuing service guarantees to execute the enqueue request successfully. A serial of application programming interfaces are provided to dequeue messages in synchronous or asynchronous way.

- The application program can use the same application programming interface as P2P messaging to do publish/subscribe operations. For more information, see Using Publish/Subscribe.
- Besides the message order pre-defined for one queue, the application program can filter the messages being dequeued from the queue by setting filters. For more information, see Using Filters.
- The application program can choose to ensure message delivery to the target queue. For more information, see Using Recoverable Messaging.
- Also the OTMQ supports flexible way to bind queue name and alias, which allows the programmer to decouple the programming and the configurations of queues. For more information, see Using Naming.

Sending and Receiving Messages

OTMQ provides the basic queuing features.

- Application should first attach to a queue using `tpqattach(3c)` before using queuing features and other advanced features provided by OTMQ.
- For message sending, application calls standard enqueue API `tpenqplus(3c)` with specified block, DIP and UMA, to determine whether messaging is synchronous or asynchronous, recoverable or not, and action to take when delivery failed as shown in Listing 1-1.

Listing 1-1 Synchronous OTMQ Queue and Enqueue Message Attachment

```
#define MSG_CLAS_EXAMPLES            2000
#define MSG_TYPE_CLIENT_REQ             1

TPQCTL ctl;
Q_ATTACH_CTL qattachctl;
char q_space[16] = "QSPACE";
char q_name[128] = "myqueue1";
long flags;

/* join the application */
if (tpinit(NULL) == -1)
```

```
{
     (void) fprintf(stderr, "failed to join application: %s\n",
                    tpstrerror(tperrno));
     exit(1);
}

memset(&qattachctl, 0x0, sizeof(qattachctl));
qattachctl.attachmode = TMQ_ATTACH_BY_NAME;
qattachctl.qtype = TMQ_ATTACH_PQ;
qattachctl.namespace_list = NULL;
qattachctl.namespace_list_len = 0;
qattachctl.timeout = 30;

memset(&ctl, 0x0, sizeof(ctl));
ctl.flags |= OTMQ;
flags = TPNOTRAN;

if (tpqattach(q_space, q_name, &ctl, &qattachctl, flags) == -1)
{
     (void) fprintf(stderr, "failed to attach q[%s.%s]: %s\n", q_space,
                    q_name, tpstrerror(tperrno));
     (void) tpterm();
     exit(1);
}

/* get request buffer */
if ((reqstr = tpalloc("STRING", NULL, len)) == NULL)
{
     (void) fprintf(stderr, "unable to allocate STRING buffer: %s",
                    tpstrerror(tperrno));
     exit(1);
}

ctl.msg_class = MSG_CLAS_EXAMPLES;  /* user defined message class */
ctl.msg_type = MSG_TYPE_CLIENT_REQ; /* user defined message type */
ctl.block = OTMQ_DEL_WF;              /* use synchronous way */
ctl.DIP = OTMQ_DIP_MEM;               /* interest point */
```

```
ctl.uma = OTMQ_UMA_RTS;             /* undelivered message action - return
to sender */
ctl.timeout = 30;

/* enqueue the message into the destination queue */
if (tpenqplus(target_qspace, target_qname, &ctl, reqstr, 0, 0) == -1)
{
     (void) fprintf(stderr, "Failure to enqueue  %s\n",
     tpstrerror(tperrno)); if (tperrno == TPEDIAGNOSTIC)
     {
            (void) fprintf(stderr, "Diagnostic code=[%d]\t",
            ctl.diagnostic);
}
     tpfree((char *) reqstr);
     (void) tpterm();
     exit(1);
}

/* detach from queue */
/* tpqdetach() */
…
```

- For synchronous message receiving, application calls standard dequeue API tpdeqplus(3c) as shown in Listing 1-2.

Listing 1-2 Synchronous Message Dequeue

```
char qspacename[16] = "QSPACE";
char qname[128] = "myqueue2";

/* call tpinit to join the application */
/* tpinit() */
…
/* attach to the queue to dequeue message from, then do the dequeue action */
```

```
/* tpqattach() */
…

memset(&ctl, 0x0, sizeof(ctl));
ctl.flags |= OTMQ;
flags = TPNOTRAN;

/* get request buffer, allocate a buffer to store the dequeued message */
len = 100;
if ((reqstr = tpalloc("STRING", NULL, len)) == NULL)
{
     (void) fprintf(stderr, "unable to allocate STRING buffer: %s",
            tpstrerror(tperrno));
     (void) tpterm();
     exit(1);
}

/* dequeue the message from the queue */
ctl.timeout = 30;
if (tpdeqplus(qspacename, qname, &ctl, &reqstr, &len, 0) == -1)
{
     if (tperrno == TPEDIAGNOSTIC)
     {
            (void) fprintf(stderr, "Diagnostic code=[%d]\t",
            ctl.diagnostic);
     } else
     {
           (void) fprintf(stderr, "Failure to dequeue  %s\n",
           tpstrerror(tperrno));
     }
     tpfree((char *) reqstr);
     (void) tpterm();
     exit(1);
}

/* detach from queue */
```

```
/* tpqdetach() */
…
```

- For asynchronous message receiving, application calls `tpqgetmsga(3c)`as shown in Listing 1-3.

Listing 1-3 Asynchonous Dequeue Message

```
/* first define the user action to be done when message arrive */
int gotMessage = 0;

int msgAction(long * flag)
{
     printf("Get asynchronous message [%s],flag=0x%X\n",reqstr,flag);
     gotMessage = 1;
}

int main(int argc, char **argv)
{
     char qspacename[16] = "QSPACE";
     char qname[128] = "myqueue1";
...

     /* join the application */
     /* tpinit() */
     …
     /* attach to the queue to dequeue message from */
     /* tpqattach() */
     …
     memset(&qctl,0,sizeof(qctl));
     qctl.flags |= OTMQ;
     qctl.filter_idx = -1;   /* no message filter designated, get the first
     available message in queue */

     size_user_data=100;
```

```
    if( tpqgetmsga(qspacename,
           qname,
           (TPQCTL *)&qctl,
           (char **)&reqstr,
           (long *)&size_user_data,
           (long *)&msgAction,
           (long *)0,
           (long *)0,
           TPNOTIME)  != 0)
     {
       /* print out the error message string or diagnostic code */
       …
       tpfree((char *) reqstr);
       (void) tpterm();
       exit(1);
       }

       /* continue to do other actions, when message arrived in queue,
       * user action "msgAction" will be called */

       …
}
```

- If received message requires confirmation, application calls `tpqconfirmmsg(3c)` to confirm receipt of the message as shown in Listing 1-4.

Listing 1-4 Explicit Confirmation for a Dequeued message

```
/* join the application */
/* tpinit() */
…
/* attach to the queue to dequeue message from, then do the dequeue action */
/* tpqattach() */
…
/* dequeue message */
/* tpdeqplus() */
```

```
…
/* check the message delivery status stored in TPQCTL */
if( ctl.status_block.del_psb_status == OTMQ__CONFIRMREQ)
{
     /* This is a message need to be confirmed explicitly,
     * use the dequeued message sequence to confirm */
     if(tpqconfirmmsg(ctl.seq_number, 0) < 0)
       {
            /* print out the error message string or diagnostic code */
            …
            tpfree((char *) reqstr);
            (void) tpterm();
            exit(1);
     }
}
```

Using Filters

OTMQ provides message filter which allows user to retrieve message that matching the selection criteria defined by the message filter. Application can designate message filter when calling standard dequeue API `tpdeqplus(3c)`, or when calling subscription API `tpqsubscribe(3c)`.

Filter Type

OTMQ supports two types of message filter: simple filter and compound filter. Simple filter has lifecycle of only one-time operation (dequeue or subscription). While the compound filter can be pre-defined and re-used in the subsequent dequeue operations.

Simple Filter

- Filter per subscription

  Message filter can be specified when subscribing the user broadcast message. It only impacts the messages retrieved according to this subscription.

- Filter per operation

  Message filter can be specified when executing a tpdeqplus/tpdequeue. It only impacts the result of this operation itself. For simple filter use, you must set `filter_idx=-1` and `flags|=TPQGETBYFILTER`, otherwise it reports an error.

Compound Filter

- Filter per queue

  Message filter can be defined or canceled via a pair of APIs: `tpqsetselect` /`tpqcancelselect`. Once a filter is defined, the user can use it in a serial of dequeue or subscription operations.

Filter Format

Different types of message filter have different kinds of format. Following sections describe the selection criteria that can be specified for the simple filter or the compound filter.

Simple Filter

For simple filter, it consists of one of the following selection criteria:

- Default Selection

  Enables application to read messages from the queue based on the order in which they arrived. Applications using default selection will read the next pending message from the message queue. Messages are stored by pre-defined queue orders (FIFO, LIFO, priority, etc.).

- Selection by Message Attribute

  Enables the application to select messages based on the message type, message class or message priority, etc.

Table 1-1 shows how the selection criteria can be defined as select mode and value pairs.

Table 1-1 Select Mode

Selection Mode	Selection Variable	Mode Description
OTMQ_PQ_TYPE	Message type value	Selects the first pending message from the attached Primary Queue (PQ) that matches the message type value being specified in the selection variable. TPQCTL->flags must set OTMQ|TPQGETBYFILTER|TPQGETBYMSGTYPE
OTMQ_PQ_CLASS	Message class value	Selects the first pending message from the attached Primary Queue (PQ) that matches the message class value being specified in the selection variable. TPQCTL->flags must set OTMQ|TPQGETBYFILTER|TPQGETBYMSGCLASS
OTMQ_PQ_PRI	• Integer value between 0 and 99 • OTMQ_PRI_ANY • OTMQ_PRI_P0 • OTMQ_PRI_P1	Selects the first pending message with a priority equal to an integer between 0 and 99 inclusive (or equal to the selection variable value) from the attached Primary Queue (PQ). Specifying the direct integer value is the preferred method of selecting message by priority Using OTMQ_PRI_ANY enables the reading of any pending messages of all priorities. Using OTMQ_PRI_P0 enables the application to retrieve pending messages of priority 0 only. Using OTMQ_PRI_P1 enables the strict retrieval of pending messages with a priority of 1.

Listing 1-5 Dequeue Message with Simple Message Filter

```
#define MSG_CLAS_EXAMPLES           2000
#define MSG_TYPE_CLIENT_REQ            1

TPQCTL ctl;
```

```
….
/* join the application */
/* tpinit() */
…

/* attach to the Qspace */
/* tpqattach() */
…

/* select by message attributes */
ctl.flags |= TPQGETBYMSGCLASS;
ctl.msg_class = MSG_CLAS_EXAMPLES;
ctl.flags |= TMQGETBYMSGTYPE;
ctl.msg_type = MSG_TYPE_CLIENT_REQ;
ctl.flags |= TPQGETBYPRIORITY;
ctl.priority = 50;
…
/* call tpdeqplus to dequeue a message with
* message class is "MSG_CLAS_EXAMPLES",
* message type is "MSG_TYPE_CLIENT_REQ" and
* message priority is 50 */
if (tpdeqplus(qspacename, qname, &ctl, &reqstr, &len, 0) == -1)
{
     /* deal with failed scenario */
     ……
}
…
/* detach from Qspace */
/* tpqdetach() */
…
```

Compound Filter

The compound filter allows application to define complex selection criteria for message reception. The selection criteria array can 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.

Application calls `tpqsetselect(3c)` function first to define a filter and gets an index handle as return, which can be used later in the standard dequeue API to retrieve messages.

Also the application can call `tpqcancelselect(3c)` to cancel the compound filter defined before as shown in Listing 1-6.

Listing 1-6 Dequeue Message Using Compund Message Filter

```
char qspacename[16] = "QSPACE";
char qname[128] = "myqueue1";
char src_qname[128] = "from_que";
TPQctl ctl;
selection_array_component_tp  selection_array;
short num_masks = 1;
int index_handle = -1;

/* join the application */
/* tpinit() */

/* attach to the Qspace */
/* tpqattach() */

/* set complex filter to dequeue message with specific message class,
  * and from specific queue*/
memset(&selection_array, 0x0, sizeof(selection_array));
selection_array.key_1_offset = OTMQ_CLASS;
selection_array.key_1_size = 4;
selection_array.key_1_value = MSG_CLAS_EXAMPLES;
selection_array.key_1_oper = OTMQ_OPER_EQ;
selection_array.key_2_offset = OTMQ_SOURCE;
selection_array.key_2_size = 4;
selection_array.key_value_qspace = qspacename;
selection_array.key_value_queue = src_qname;
selection_array.key_2_oper = OTMQ_OPER_EQ;

if(tpqsetselect(&selection_array, &num_masks, &index_handle ) == -1)
{
```

```
     /* deal with failed scenario */
     …
}

ctl.filter_idx = index_handle; /* using the filter to dequeue */

if(tpdeqplus(qspacename, qname, &ctl, &reqstr, &len, 0) == -1)
{
     /* deal with failed scenario */
     …
}

/* need to cancel the filter using the index */
if(tpqcancelselect(&index_handle)== -1)
{
     /* deal with failed scenario */
     …
}

/* detach from Qspace */
/* tpqdetach() */
…
```

For more information, see `tpqsetselect()` and `tpqcancelselect()` in the *Oracle Tuxedo Message Queue Reference Guide*.

Using Publish/Subscribe

The publisher broadcast a message by sending the message to a special "topic". Each topic represents a broadcast stream. A broadcast stream is the set of target queues registered to receive messages directed to a particular topic. The subscriber should register first for a topic to receive the specific broadcasting messages.

The OTMQ Message Queue Manager Server is responsible for maintaining lists of user processes that are interested in the specific topic. In addition, the server is responsible for maintaining the various user definable rules (also known as pub/sub filter) that can be used to selectively extract messages from the broadcast stream that are set by the application using the `tpqsubscribe(3c)`.

A publisher can send a broadcast message using `tpqpublish(3c)`, and a subscriber can retrieve the subscribed message from its attached queue.

Sending Broadcast Messages

To broadcast a message, a publisher program directs the message to the topic that identifies the broadcast stream to use for message distribution. When the application issues the `tpqpublish(3c)` function, OTMQ Message Queue Manager Server deals with the `tpqpublish(3c)` call and transparently redirects the message to all corresponding recipients.

OTMQ Message Queue Manager Server delivers only one copy of each message on the broadcast stream to each target queue, regardless of how many selection matches are made by separate subscription rule entries.

Broadcast messages cannot be stored for automatic message recovery.

Receiving Broadcast Messages

To receive broadcast messages, applications use a standard API `tpqsubscribe(3c)` to register for receipt with the local or remote OTMQ Message Queue Manager Server.

The following topics describe:

- Subscribing to Receive Broadcast Messages
- Subscribing to Receive Selected Broadcast Messages
- Subscription Acknowledgement
- Reading Broadcast Messages
- Unsubscribing Receiving Broadcast Messages

Subscribing to Receive Broadcast Messages

To receive broadcast messages, an application registers a queue with a broadcast stream (topic) that managed by the OTMQ Message Queue Manager Server.

The receiver/subscribing applications register for broadcast messages using the function `tpqsubscribe(3c)`. The registration contains the string formatted topic plus any selection criteria (filter) related to messages that the application wishes to receive.

The application subscribe the broadcast messages using the function `tpqsubscribe(3c)` supplied with the following information:

- The topic: the broadcast stream that wants to subscribe
- The target: the target OTMQ Message Queue Manager Server, and the special flag which means Pub/Sub service.
- The source: the queue name which the requesting application is attaching.

Subscribing to Receive Selected Broadcast Messages

Use the message filter of `tpqsubscribe(3c)` to register for selective reception of broadcast messages. This subscription request registers a target queue to receive a copy of all messages on a broadcast stream that meet a single selection rule.

Filter is a string containing a Boolean filter rule that must be evaluated successfully before the OTMQ Message Queue Server distributing the broadcast messages to the subscriber. Filter rules are specific to the types buffers to which they are applied. For messages of string type, the filter rule is a regular expression. For messages of FML buffers, the filter rule is a string that can be passed to the FML Boolean compiler (see Tuxedo ATMI FML function `Fboolco`).

Table 1-2 shows how the Filter Regular Expression Rules can be defined.

Table 1-2 Regular Expression Rules

Rule	Matching Text
character	Itself (character is any ASCII character except the special ones mentioned below).
\ character	Itself except as follows: \\-newline \\t-tab \\b-backspace \\r-carriage return \\f-formfeed
\ special-character	Its un-special self. The special characters are `. * + ? | ( ) [ {` and \\. -Any character except the end-of-line character (usually newline or NULL). ^-Beginning of the line. $-End-of-line character.

Table 1-2 Regular Expression Rules

Rule	Matching Text
[class]	any character in the class denoted by a sequence of characters and/or ranges. A range is given by the construct character-character. For example, the character class, [a-zA-Z0-9_], will match any alphameric character or "_". To be included in the class, a hyphen, "-", must be escaped (preceded by a "\\") or appear first or last in the class. A literal "]" must be escaped or appear first in the class. A literal "^" must be escaped if it appears first in the class.
[^ class]	Any character in the complement of the class with respect to the ASCII character set, excluding the end-of-line character.
RE RE	The sequence. (catenation)
RE | RE	Either the left RE or the right RE. (left to right alternation)
RE *	Zero or more occurrences of RE.
RE +	One or more occurrences of RE.
RE ?	Zero or one occurrences of RE.
RE { n }	n occurrences of RE. n must be between 0 and 255, inclusive.
RE { m, n }	m through n occurrences of RE, inclusive. A missing m is taken to be zero. A missing n denotes m or more occurrences of RE.
(RE)	Explicit precedence/grouping.
(RE) $ n	The text matching RE is copied into the nth user buffer. n may be 0 through 9. User buffers are cleared before matching begins and loaded only if the entire pattern is matched.

There are three levels of precedence. In order of decreasing binding strength they are:

- catenation closure (*,+,?,{...})
- catenation
- alternation (|)

As indicated above, parentheses are used to give explicit precedence.

Subscription Acknowledgement

The `tpqsubscribe(3c)` returns a subscription handle back to the subscriber. This handle should be used to unsubscribe the specific subscription.

Reading Broadcast Messages

When a message is sent to a broadcast stream, the OTMQ Message Queue Manager Server will determine which applications have registered to receive that kind of message. Then it automatically sends these messages to the distribution of all matching applications. The receiving application reads the broadcast message from its target queue using the standard dequeue functions. The source queue address of the sender is also provided to the receiving application in the message.

Unsubscribing Receiving Broadcast Messages

An application can withdraw subscribing messages from a broadcast stream by calling the `tpqunsubscribe(3c)`. This action removes the subscription entry from the internal registration tables as shown in Listing 1-7

Listing 1-7 Subscribe and Unsubscribe Messages

```
TPEVCTL evctl;
long evt_handle = 0L ;  /* Event Subscription handles */
…
/* join the application */
/* tpinit() */
…

/* attach to the Qspace */
/* tpqattach() */
…
evctl.flags = TPEVQUEUE ;
(void)strcpy (evctl.name1, "QSPACE") ;
(void)strcpy (evctl.name1, "myqueue1") ;
evctl.qctl.flags |= OTMQ;

/* Subscribe */
```

```
evt_handle = tpqsubscribe ("TMQ:QNOT:QSPACE:mytopic",
   NULL,&evctl,TPSIGRSTRT) ;
if (evt_handle == -1L) {
     /* deal with failed scenario */
     …
}
…
/* dequeue subscribed message */
if(tpdeqplus(qspacename, qname, &ctl, &reqstr, &len, 0) == -1)
{
     /* deal with failed scenario */
     …
}

/* Unsubscribe to the subscribed topic */
if (tpqunsubscribe (evt_handle, TPSIGRSTRT) == -1)
{
     /* deal with failed scenario */
     …
}
```

For more information, see `tpqsubscribe()` and `tpqunsubscribe()` in the *Oracle Tuxedo Message Queue Reference Guide*.

Using Recoverable Messaging

Applications send messages using the OTMQ standard enqueue function `tpenqplus(3c)` and one of two types of delivery modes: recoverable or non-recoverable. If a message is sent as non-recoverable, the message is lost if it cannot be delivered to the target queue unless the application incorporates an error recovery procedure. If the message is sent as recoverable, OTMQ Message Queue Manager Server and Offline Trade Driver can automatically guarantee delivery to the target queue in spite of system, process, and network failures.

To ensure guaranteed delivery, the OTMQ Message Queue Manager Server writes recoverable messages to nonvolatile storage on the sender or receiver system. Then, if a message cannot be

delivered due to an error condition, the OTMQ Offline Trade Driver attempts redelivery of the message by reading it from the recovery journal and redelivering the message to the target queue until delivery is confirmed.

Application developers determine which messages should be sent as recoverable depending upon the needs of the application. Because recoverable messaging requires the extra step of storing the messages on disk, it requires additional processing time and power. To maximize performance, recoverable messaging should only be used when it is critical to application processing.

The OTMQ recoverable messaging feature offers the following benefits:

- Reduces development time by eliminating the need for designing applications to recover messages that cannot be delivered.
- Prevents applications from losing data when applications, systems, or network links fail.
- Simplifies the implementation of an event-driven store and forward capability in networked applications.

OTMQ also offers error recovery features for non-recoverable messages such as the dead letter queue (DLQ) and the ability to return a message to the sender if the message cannot be delivered. This topic describes all of the OTMQ delivery modes to enable you to understand the right choice for your application.

This section contains the following topics

Choosing a Message Delivery Mode

The choice between recoverable and non-recoverable delivery is based upon the needs of the application. To determine the appropriate method for sending a message, the application developer decides:

- Does the application need to know if the message arrived at the target queue?
- If notification is required, how far must the message get before the sender program receives notification that the message has arrived?

- Should the application wait for notification or should it continue processing and receive notification through an asynchronous acknowledgment message?
- If the message is designated as recoverable, does the application need to know if the message has been stored by the recovery system?
- If the message is designated as recoverable, what should happen if it cannot be stored by the recovery system?

The delivery mode specified in `tpenqplus(3c)` function determines:

- Whether the message is sent as recoverable or non-recoverable
- Whether a blocking or non-blocking mode is selected
- Whether the sender program receives notification and how it is received
- The point in the message flow at which the notification is sent

OTMQ uses message recovery journals to store messages that are designated as recoverable. The message recovery journal on the local system is called the store and forward (SAF). The message recovery journal on the remote system is called the destination queue file (DQF). If a recoverable message cannot be delivered, it is stored in either the SAF or DQF queue and is automatically re-sent once communication with the target group is restored.

OTMQ uses auxiliary journals to provide additional message recovery capabilities. The dead letter queue (DLQ) is a memory-based queue for storing undeliverable messages. The dead letter journal (DLJ) provides disk storage for messages that could not be stored for automatic recovery. Undelivered messages stored in the DLQ or DLJ can be retrieved under user or application control by using OTMQ's Journal API `tpqreadjrn(3c)`.

The post confirmation journal (PCJ) stores successfully confirmed recoverable messages.

If the OTMQ message recovery system is unable to store the message, the undeliverable message action (UMA) is taken. Some UMAs enable the message to be recovered at a later time under user or application control.

Choosing Recoverable or Non-recoverable Delivery Mode

The delivery mode consists of two components, the block type (block) and the delivery interest point (DIP). Specify the block and DIP in the TPQCTL structure.

- block - indicates how the sender program wants to receive information about the delivery of the message. You can either wait for the operation to complete (WF), or receive

notification in an asynchronous message (AK), or choose not to receive notification (NN) at all.

- DIP - determines whether the message is designated as recoverable. When the message reaches the delivery interest point, a notification message is sent (if requested) and the call returns control to the sender program or OTMQ Message Queue Manager Server delivers the asynchronous acknowledgment message.

Non-recoverable delivery interest points enable the sender program to receive notification when the message is stored in the target queue (MEM), when the message is read from the target queue (DEQ), or when the message is read from the target queue and explicitly confirmed by the receiver program using the tpqconfirmmsg(3c) function (ACK).

When a recoverable delivery interest point is selected, the message is stored on disk for automatic recovery. Recoverable delivery interest points enable the sender program to store the message in the local recovery journal (SAF), store the message in the remote recovery journal (DQF), or store the message in the remote recovery journal and receive notification when the message is confirmed by the target application (CONF).

OTMQ does not support all possible combinations of block type and delivery interest points. Table 1-3 lists the valid delivery modes and their meanings.

Table 1-3 Delivery Modes

Delivery Mode	Description
(Recoverable Delivery Modes)	
block = OTMQ_DEL_AK DIP = OTMQ_DIP_CONF	Send acknowledgment message when the message recovery system confirms message delivery from the remote recovery journal.
block = OTMQ_DEL_AK DIP = OTMQ_DIP_DQF	Send acknowledgment message when the message is stored in the remote recovery journal.
block = OTMQ_DEL_AK DIP = OTMQ_DIP_SAF	Send acknowledgment message when the message is stored in the local recovery journal.
block = OTMQ_DEL_NN DIP = OTMQ_DIP_DQF	Deliver message to the remote recovery journal but do not block and do not send notification.
block = OTMQ_DEL_NN DIP = OTMQ_DIP_SAF	Deliver message to the local recovery journal but do not block and do not send notification.

Table 1-3 Delivery Modes

Delivery Mode	Description
block = OTMQ_DEL_WF DIP = OTMQ_DIP_CONF	Block until the message is stored in the remote recovery journal and confirmed by the target application.
block = OTMQ_DEL_WF DIP = OTMQ_DIP_DQF	Block until the message is stored in the remote recovery journal.
block = OTMQ_DEL_WF DIP = OTMQ_DIP_SAF	Block until the message is stored in the local recovery journal.
(Non-Recoverable Delivery Modes)	
block = OTMQ_DEL_AK DIP = OTMQ_DIP_ACK	Send acknowledgment message when the receiver program explicitly confirms delivery using tpqconfirmmsg(3c).
block = OTMQ_DEL_AK DIP = OTMQ_DIP_DEQ	Send acknowledgment message when the message is removed from the target queue.
block = OTMQ_DEL_AK DIP = OTMQ_DIP_MEM	Send acknowledgment message when the message is stored in the target queue.
block = OTMQ_DEL_NN DIP = OTMQ_DIP_MEM	Deliver message to the target queue but do not block and do not send notification.
block = OTMQ_DEL_WF DIP = OTMQ_DIP_ACK	Block until the receiver program explicitly confirms delivery using tpqconfirmmsg(3c)
block = OTMQ_DEL_WF DIP = OTMQ_DIP_DEQ	Block until the message is removed from the target queue.
block = OTMQ_DEL_WF DIP = OTMQ_DIP_MEM	Block until the message is stored in the target queue.

Non-recoverable message delivery is the fastest and most efficient way to send messages. Use non-recoverable delivery modes if:

- High messaging rates are required by the application (hundreds or thousands of messages per second).

- The message content has a finite lifetime; therefore, the value of the information is stale if not received and processed quickly.
- The message is sent locally between two applications in the same message queuing group that tightly cooperate in the processing of an event.
- The message is a control message that causes a component of an application to change state.

Recoverable message delivery is the safest way to send a message; however, it adds significant processing overhead because each message must be stored before it is sent. Use recoverable delivery modes if:

- It is useful to know that the message has arrived; however, the sender does not need to know the state of the receiver.
- The message content should not be lost by the application system.
- The application can tolerate the increased system load and slower messaging rate caused by sending the message

Choosing an Undeliverable Message Action

Using the `tpenqplus(3c)` function in conjunction with the delivery argument, you can use the UMA argument to specify what should happen to the message if it cannot be delivered to the delivery interest point.

With non-recoverable messaging, the UMA indicates the action to be taken if the message cannot be stored in target queue. If a UMA is not specified, OTMQ will take the default action of discarding the message.

With recoverable messaging, the UMA indicates the action to be taken if the message cannot be stored in either the SAF or DQF queues. You must specify a UMA with recoverable delivery modes because your application must perform the exception processing when the message cannot be guaranteed for delivery by OTMQ Message Queue Manager Server.

With recoverable messaging, the UMA may be taken when:

- OTMQ is unable to write to the local SAF queue where the message is designated to be recoverable.
- The cross-group connection to the remote target group is down and the message designated as recoverable on the remote node (DQF) cannot be stored.
- The system resources used by the message recovery system are exhausted.

Table 1-4 lists the five valid UMAs.

Table 1-4 UMAs

UMA	Description
OTMQ_UMA_DISC	Discard - the message is deleted.
OTMQ_UMA_RTS	Return to sender - the message is delivered to the sender's response queue.
OTMQ_UMA_SAF	Store and forward - the message is written to the message recovery journal on the sender system.
OTMQ_UMA_DLQ	Dead letter queue - the message is written to the dead letter queue.
OTMQ_UMA_DLJ	Dead letter journal - the message is written to the DLJ.

How to Send a Recoverable Message

To send a recoverable message, use the `tpenqplus(3c)` function supplying the appropriate block type, DIP and UMA in the TPQCTL structure.

In addition, the application should:

- Specify a timeout value when sending recoverable messages with blocking delivery modes.
- Verify the delivery outcome of a send operation from PSB in TPQCTL structure. If the message was failed to be stored by the OTMQ Message Queue Manager Server, the application must check to make sure that the UMA was successfully executed.

The message flow for sending a recoverable message is:

- The application sends a message using tpenqplus(3c) function and with the appropriate block, DIP and UMA arguments.
- The OTMQ Message Queue Manager Server first writes the message to the recovery journal queue on the local (SAF) or remote system (DQF) depending upon the delivery mode specified.
- If the sender is blocked, it continues processing once the message reaches the delivery interest point. If the sender requests notification, it received an acknowledgement message once the message reaches the delivery interest point.

For more information, see `tpenqplus()` in the Oracle Tuxedo Message Queue Reference Guide.

How to Receive a Recoverable Message

To receive a recoverable message, use the `tpdeqplus(3c)` function. When a recoverable message is delivered to the target queue, the application must perform the following:

- Confirm message receipt, which allows the Offline Trade Driver (`TuxMQFWD(5)`) to delete the message being stored in the recovery journal queue before delivery.
- Check for duplicate messages via return code. Duplicate messages may be sent by the Offline Trade Driver if the application didn't confirm message receipt in time. For more information, see `tpdeqplus()` in the *Oracle Tuxedo Message Queue Reference Guide*.

The message flow for receipt of a recoverable message is as follows:

- A message is read from the message recovery journal queue by the Offline Trade Driver and sent to the target queue of the receiver.
- The receiver reads the message by calling tpdeqplus(3c) function.
- If the queue is configured for explicit confirmation, the receiver should call the `tpqconfirmmsg(3c)` function to acknowledge receipt of the recoverable message using the message sequence number assigned by the OTMQ Message Manager Server when the message was sent. If the queue is configured for implicit confirmation, the acknowledge message will be sent by `tpdeqplus(3c)` automatically after the recoverable message is dequeued successfully. For more information, see `tmqadmin` in the *Oracle Tuxedo Message Queue Reference Guide*.
- The `tpqconfirmmsg(3c)` function sends acknowledge notification to the Offline Trade Driver to notify the successful message delivery, which allows the Offline Trade Driver to remove the message from the message recovery journal queues.

Using UMAs for Exception Processing

Using Discard UMA

When the DISC UMA is used, the message is discarded if it cannot be delivered to the delivery interest point specified in the delivery mode argument. The DISC UMA is used when the sender program will handle each exception as it occurs. OTMQ can discard the undeliverable message because the message content is still available in the context of the sender program.

Using the Return-to-Sender UMA

When the RTS UMA is used, the message is redirected to the response queue of the sender program if it cannot be delivered to the delivery interest point specified in the delivery mode argument. The RTS UMA is used when the sender program does not want to process each exception as it occurs. Instead, the sender program redirects undeliverable messages to its main input stream for error handling.

The advantage of using the RTS UMA is that the sender program attaches to one queue and acts upon each message as it is read. The sender program must read the PSB status delivery value of each message to determine if the message is new or an undeliverable message. Messages that could not be stored by the message recovery system and require error handling have a return status of OTMQ__MSGUNDEL.

Using the SAF UMA

When the SAF UMA is used, the message is stored in the local journal queue if the message recovery system is unable to store it in the remote journal queue. The SAF UMA can be used with recoverable delivery interest points of DQF and CONF; however, it does not work with the WF_SAF delivery mode.

Use of the SAF UMA helps to manage the flow control between the sender and receiver systems. If the message cannot be written to the remote journal queue due to insufficient resources or a cross-group link failure, the message will be written to the local journal queue.

Note: SAF here means nearly the same as "SAF" DIP. Once message cannot be delivered, store message into SAF queue.

Using the Dead Letter Queue UMA

When the DLQ UMA is used, the message is redirected to the dead letter queue if it cannot be delivered to the delivery interest point specified in the delivery mode argument. The DLQ UMA is used when the sender program wants to centralize error handling for undeliverable messages in a designated queue while allowing each message to be handled separately.

A dead letter queue is part of the standard group configuration for each OTMQ message queuing group. It provides memory-based storage of all undeliverable messages for the group that could not be stored for automatic recovery. The dead letter queue is defined as queue number 96 It is created automatically by `tmqadmin(1) qspacecreate` command.

To use the dead letter queue, the sender program calls the `tpenqplus(3c)` function specifying the appropriate delivery argument and using OTMQ_UMA_DLQ as the UMA argument. Any

messages that cannot be delivered to the receiver program are written to the dead letter queue of the sender's group. An application program can use `tpqreadjrn(3c)` function to retrieve undelivered messages and use the `tpenqplus(3c)` function to attempt redelivery.

An advantage of using the dead letter queue is the ability to recover undeliverable messages on a one-by-one basis. The sender program or another process within the application can attach to the DLQ and handle error recovery for each undeliverable message. A disadvantage of using the dead letter queue is the lack of disk storage for undelivered messages. A system failure on the sending node will cause all undelivered messages in the dead letter queue to be lost.

Using the Dead Letter Journal

When the DLJ UMA is used, the message is written to an auxiliary journal, queue number is 196 (the dead letter journal) if it cannot be delivered to the delivery interest point specified in the delivery mode argument. This UMA can only be used for recoverable messages. The DLJ UMA is used when the sender program needs to centralize error handling procedures and the application can support the resending of many messages from DLJ queue at a delayed interval. Storing undeliverable messages in DLJ queue ensures that they will not be lost if the system goes down, and allows redelivery attempts under user or application control.

The dead letter journal provides disk storage for messages that could not be stored for automatic recovery. It is created automatically by `tmqadmin(1)` `qspacecreate` command.

To use the dead letter journal, the sender program uses the `tpenqplus(3c)` function specifying the appropriate delivery argument and OTMQ_UMA_DLJ as the UMA argument. Any messages that cannot be stored by the message recovery system are written to the dead letter journal of the sender's group. An application program can use `tpqreadjrn(3c)` function to retrieve undelivered messages and use the `tpenqplus(3c)` function to attempt redelivery as shown in Listing 1-8.

Listing 1-8 Using UMA with Undelivered Message Example

```
TPQCTL ctl;
int type;
…

/* join the application */
/* tpinit() */

/* attach to the QSpace */
```

```
/* tpqattach() */

/* get request buffer */
if ((reqstr = tpalloc("STRING", NULL, len)) == NULL)
{
     (void) fprintf(stderr, "unable to allocate STRING buffer: %s",
               tpstrerror(tperrno));
     exit(1);
}

ctl.block = OTMQ_DEL_WF;              /* use synchronous way */
ctl.DIP = OTMQ_DIP_SAF;               /* interest point */
ctl.uma = OTMQ_UMA_DLJ;              /* undelivered message action - Dead
Letter Journal*/

/* enqueue the message into the destination queue */
if (tpenqplus(target_qspace, target_qname, &ctl, reqstr, 0, 0) == -1)
{
     /* deal with failed scenario */
     …
}
…
…
/* done other works, handle failed message in DLJ before exit */
ctl.flags |=OTMQ;
ctl.flags |= TPQREADJRN;
type = DLJ_HANDLE;
if (tpqreadjrn(myqspace, myqueue, &ctl, &rcv_buf, &len, 0) == -1)
{
     /* deal with failed scenario */
     …
}

/* enqueue the failed message again */
if (tpenqplus(target_qspace, target_qname, &ctl, rcv_buf, 0, 0) == -1)
{
     /* deal with failed scenario */
     …
```

```
}
…
/* detach from the Qspace */
/* tpqdetach() */
…
```

The DIP and UMA Support List

Table 1-5 lists the valid delivery modes and UMA combinations.

Table 1-5 DIP and UMA Support List

	UMA				
Delivery Mode	SAF	DLJ	DLQ	RTS	DISC
block = OTMQ_DEL_NN DIP = OTMQ_DIP_MEM	N	N	Y	Y	Y
block = OTMQ_DEL_WF DIP = OTMQ_DIP_MEM	N	N	Y	Y	Y
block = OTMQ_DEL_AK DIP = OTMQ_DIP_MEM	N	N	Y	Y	Y
block = OTMQ_DEL_AK DIP = OTMQ_DIP_DEQ	N	N	Y	Y	Y
block = OTMQ_DEL_WF DIP = OTMQ_DIP_ACK	N	N	Y	Y	Y
block = OTMQ_DEL _AK DIP = OTMQ_DIP_ACK	N	N	Y	Y	Y
block = OTMQ_DEL _WF DIP = OTMQ_DIP_DEQ	N	N	Y	Y	Y
block = OTMQ_DEL _AK DIP = OTMQ_DIP_SAF	N	Y	Y	Y	Y

Table 1-5 DIP and UMA Support List

				UMA	
block = OTMQ_DEL _WF DIP = OTMQ_DIP_SAF	N	Y	Y	Y	Y
block = OTMQ_DEL _NN DIP = OTMQ_DIP_SAF	N	Y	Y	Y	Y
block = OTMQ_DEL _AK DIP = OTMQ_DIP_CONF	Y	Y	Y	Y	Y
block = OTMQ_DEL _WF DIP = OTMQ_DIP_CONF	Y	Y	Y	Y	Y
block = OTMQ_DEL _NN DIP = OTMQ_DIP_DQF	Y	Y	Y	Y	Y
block = OTMQ_DEL _WF DIP = OTMQ_DIP_DQF	Y	Y	Y	Y	Y
block = OTMQ_DEL _AK DIP = OTMQ_DIP_DQF	Y	Y	Y	Y	Y

Using Naming

In OTMQ configuration, each message queue gets a name when created, and also can get an alias at runtime. Naming is a powerful feature that enables OTMQ applications to identify message queues by name/alias whether they reside on the local system or on another system.

Application developers use the OTMQ naming feature to separate their applications from the underlying OTMQ environment configuration. By referring to message queues by name/alias in the applications, developers do not have to modify their applications code when the OTMQ environment configuration changes.

The user must configure `TMQ_NA(5)` server in UBB to take advantage of the naming service.

Naming Service

Naming service is provided by the OTMQ Naming Server. It can access and manage both the local namespace and global namespace for the runtime queue lookup when an application refers to a queue by alias, or dynamic binding a queue alias to a specified queue name. OTMQ Naming Server provides two services: one for the local scope alias lookup (Local Naming Service), another for the global's (Global Naming Service).

Name Scope

When a name or alias is defined for message queue, it is assigned a name scope. Queue name or alias can have a local (intra-qspace) or global (inter-qspace) scope. A local alias can be used by applications running in the same queue space in which the alias was defined. A global alias can be used by any applications.

Name Space

A name space is the repository where message queue alias and their associated message queue address (queue space and queue name) are stored. OTMQ Naming Server must look up the alias in the name space to find its associated actual queue address in order to send a message to the named queue.

OTMQ Naming Server uses three levels of name spaces: process, local (qspace-wide) and global (cross qspace wide). When OTMQ Naming Server start up, the local scope alias will be stored in local name space. The global scope alias will be stored in global name space. The process name space is an application cache used to improve performance. The alias can be cached at different level name space, user can bypass caching when using `tpqlocate(3c)` if they prefer accuracy over performance.

Process Level Name Space

When application attaches to the OTMQ, application automatically creates the empty process name space. When this process locates/binds an alias for the first time, it is cached in the process name space.

Local Name Space

When OTMQ Naming Server starts up, it automatically creates the local name space. Also local scope queue alias defined by applications will be cached in the local name space.

Global Name Space

To create the global name space, use a flat file system by creating the directory in which the OTMQ naming service will maintain the name space.

To use global naming, you must create a global namespace on the nodes on which the Naming Server runs. OTMQ allows user to configure at most two global naming services (primary and backup). To enable the backup naming service to take responsibility when the primary one is down, all the global naming services must be configured to use the same global name space. Therefore, when configured naming services run on different systems, they must use a shared file system.

After creating the name space, you must set the `DMQNS_DEVICE` environment variable to specify a device name for the name space because access to the name space for global naming is system dependent. Therefore, when a global naming service is configured, it must be told what device name to be used when it accesses this name space. This is done by setting the environment variable `DMQNS_DEVICE` as follows:

- For Windows NT, it should be set to a drive letter followed by a colon (for example, c:> o a full qualified share name (e.g. `\\machine\share`)
- For UNIX, it should be set to a file system specification (for example, / or `/usr` or `/mnt/dmqns`)

Note: this environment variable need only be set for the OTMQ Naming Server which provides the naming services. To use the global naming service, at least some portion of the global namespace file path must be specified. For example on UNIX, you can define it as "`/u/mydir`".

When an application refers to a queue by alias using the `tpqlocate(3c)` or the `tpqbind(3c)` functions, it can specify the alias as one of the following:

- unqualified name: The application uses only the queue alias such as "widgets" and does not specify the path. The naming service automatically prefixes the name with the value of the environment variable `DMQNS_DEVICE`. Furthermore, it prefix the value of the environment variable DMQNS_DEFAULTPATH begins with a "/". For example, if the `DMQNS_DEVICE` environment variable is set to "dev" and the DMQNS_DEFAULTPATH is set to "`/inventory`", the naming service would search for the name "widgets" in: `/dev/inventory/widgets`
- partially qualified name: The application specifies the queue alias and a portion of the path name. The naming service automatically prefixes the pathname and queue alias with the device specified as the `DMQNS_DEVICE` environment variable and the setting of the DMQNS_DEFAULTPATH environment variable. For example, if the `DMQNS_DEVICE`

environment variable is set to "/dev" and the DMQNS_DEFAULTPATH is set to "`/inventory`", the naming service would search for the name "`test/widgets`" in: `/dev/inventory/test/widgets`.

- fully qualified name: The application specifies that the alias is a fully qualified name using "/" as the first character of the name. When the first character of a name begins with "/", the naming service does not prefix any information to the name other than the device name specified by the `DMQNS_DEVICE` environment variable. This means that a fully qualified name includes the full path name and queue name. For example, if the `DMQNS_DEVICE` environment variable is set to "dev" and the DMQNS_DEFAULTPATH is set to "/inventory", the naming service will search for the name "/production/test/widgets" in: /dev/production/test/widgets. Listing 1-9 shows a global namespace file example.

Listing 1-9 Global Namespace File Example

```
PrimaryQ_1       0.0       L
myqueue1        0.0       G
MRQ13_1        1.13       L
SQ14_2          0.0       G
```

Attaching and Locating Queues

An application must attach to a queue using the `tpqattach(3c)` function before reading message from or sending message to a queue. It can identify the queue by its alias or its name. When sending a message, the target queue is always identified by its name. An application can directly use the queue name or it can use the `tpqlocate(3c)` function to derive the queue name from its alias.Listing 1-10 shows locating queue example.

Listing 1-10 Locating Queue Example

```
static char q_space[16] = "QSPACE";
static char q_name[128] = "myqueue1";
…
Q_NAME_CTL  name_ctl;
long scope = NAME_SCOPE_P;
```

```
/* join the application */
/* tpinit() */

/* attach to the QSpace */
/* tpqattach() */

/* locate queue named "Primary_queue" */
name_ctl.pName = "Primary_queue";
wait_mode = OTMQ_WF_RESP;  /* use synchronous way to get response */

if(tpqlocate(q_space, &name_ctl, 0, NULL, &scope, wait_mode, 0) == -1)
{
     /* deal with failed scenario */
     …
}
```

Static and Dynamic Binding of Queue Aliases

OTMQ offers two approaches to associating a queue alias with a queue address: static and dynamic.

Static binding refers to associating a queue name with a queue alias using the name space file. To enable such association, you can specify the name space file when creating the queue space; or you can stop the Naming Server, then update the queue space to specify a name space file, then restart the Naming Server again. For more information, see `tmqadmin` in the *Oracle Tuxedo Message Queue Reference Guide*.

Dynamic binding refers to the use of `tpqbind(3c)` to associate a queue alias to a queue name at application runtime. The queue alias will not be bound to a specific queue name until the `tpqbind(3c)` successfully return. To modify such association, the application must first unbind the queue alias from the specific queue name using `tpqbind(3c)`, and use the same API to associate another queue alias to the queue name again. When the application detach from the queue or exit the queue space, the Naming Server will unbind the association for this application automatically. shows a dynamic binding queue example.

Listing 1-11 Dynamic Binding Queue Example

```
static char q_space[16] = "QSPACE";
static char q_name[128] = "myqueue1";

…
Q_NAME_CTL  name_ctl;
long scope = NAME_SCOPE_G;

name_ctl.pName = "Primary_queue";
name_ctl.pGroup = q_space;
name_ctl.pQueue = q_name;

bind_time_out = 30;

if(tpqbind(q_space, &name_ctl, &scope, bind_time_out) == -1)
{
     /* deal with failed scenario */
     …
}
```

For more information, see `tpqlocate()` and `tpqbind()` in the *Oracle Tuxedo Message Queue Reference Guide*.

Using WS SAF

In WS mode, OTMQ messages that are sent using a recoverable delivery mode are written to the local store-and-forward (SAF) journal (`tmqsaf.jrn`) when the connection to the server system is not available.

When the feature is enabled, messages sent using the following reliable delivery modes are saved to the journal:

- `OTMQ_DIP_MEM & OTMQ_DEL_WF (using OTMQ_UMA_SAF)`
- `OTMQ_DIP_DQF & OTMQ_DEL_WF`
- `OTMQ_DIP_DQF & OTMQ_DEL_AK`
- `OTMQ_DIP_SAF & OTMQ_DEL_WF`

```
OTMQ_DIP_SAF & OTMQ_DEL_AK
```

OTMQ WS configuration options allow the SAF journal to be configured as follows:

A fixed-size file that does not reuse disk blocks

A fixed-size file that reuses (cycles) disk blocks

A dynamic file that grows indefinitely until no more disk blocks are available

These options allow you to determine how disk resources are used for message journals. Journal files that grow indefinitely periodically allocate an extent of disk blocks as needed to store messages. When all messages are sent from the SAF and the journal is empty, the disk blocks used by the journal are freed and the journal file is removed. Journal file size is in units of disk block size (4096 bytes).

Building Applications

As counterparts of Tuxedo buildclient(1) and buildserver(1) commands, OTMQ provides `buildqclient(1)` and `buildqserver(1)`.

`buildqclient(1)` is used to construct an OTMQ client module. The command combines the supplied files with the standard Oracle Tuxedo ATMI libraries and OTMQ libraries to form a load module.

`buildqserver(1)` is used to construct an OTMQ server load module. The command combines the supplied files with the standard server main routine, the standard Oracle Tuxedo ATMI libraries and OTMQ libraries to form a load module.

For more information, see `buildqclient` and `buildqserver` in the *Oracle Tuxedo Message Queue Reference Guide*.

CHAPTER 2

Oracle Tuxedo Message Queue PAMS Programming Guide

This chapter contains the following topics:

PAMS Application Programming Interface

Because the Oracle MessageQ application programming interface (API) is portable, the API is documented once for all supported platforms. This chapter describes all Oracle MessageQ callable services in alphabetical order using a standard description -format.

Note: Using APIs starting with "`tp`" instead of APIs starting with "`pams`" is *highly* recommended.

Oracle Tuxedo Message Queue supports PAMS APIs for Oracle Messageq (OMQ) compatibility. To migrate OMQ applications to OTMQ, please see Migrating from OMQ to OTMQ 12c.

Oracle MessageQ API Description Format

The beginning of each description contains the entry-point name and a brief description of the function performed. Table 2-1 Table 8-1 describes the sections in the description of each callable service.

Table 2-1 Callable Service Description Formats

In the section entitled . . .	You will find . . .
Syntax	The syntax for using the callable service with the entry-point name and argument list. Square brackets ([ ]) indicate optional arguments to the service. The syntax for using the callable
Arguments	The data type, passing mechanism, C language prototype, and access for each argument.
Argument Definition	More detailed information on how to use the callable service.
Description	More detailed information on how to use the callable service.
Return Values	The return codes with the platforms on which they are supported.
See Also	A list of related callable services. For more information about these services, see this "Oracle MessageQ API Data Types" section.
Example	A sample program illustrating the use of the callable service. These sample programs are available in the examples directory of the Oracle MessageQ media kit.

Oracle MessageQ API Data Types

Oracle MessageQ API arguments use data types defined by the C programming language and some data types defined by Oracle MessageQ software. Data types such as `short`, -`unsigned short`, and `char` are data types defined by the C programming language. Oracle MessageQ data types such as `q_address` and the PSB and `show_buffer` structures are defined in the `p_entry.h` include file.

Oracle MessageQ supports data type definitions for signed and -unsigned longwords. The `int32` data type defined by Oracle MessageQ is a 32-bit signed integer. The `int32` data type replaces the use of the integer data type long, the size of which is operating system dependent. The `int32` data type definition guarantees a consistent definition across all platforms and was added to accommodate next generation 64-bit architectures such as the Compact Alpha AXP

systems. The uint32 data type designates a 32-bit unsigned integer and replaces the use of unsigned long.

Note: The int32 and uint32 data type definitions are not available on Oracle MessageQ Version 2.0 platforms. Oracle MessageQ Version 2.0 software uses the standard signed longword and unsigned longword data types defined by the C programming language.

- pams_attach_q
- pams_bind_q
- pams_cancel_get
- pams_cancel_select
- pams_cancel_timer
- pams_close_jrn
- pams_confirm_msg
- pams_detach_q
- pams_exit
- pams_get_msg
- pams_get_msga
- pams_get_msgw
- pams_locate_q
- pams_open_jrn
- pams_put_msg
- pams_read_jrn
- pams_set_select
- pams_set_timer
- pams_status_text
- putil_show_pending

pams_attach_q

Connects an application program to the Oracle MessageQ message queuing bus by attaching it to a message queue. An application must successfully execute this function before it can send and receive messages. When an application uses this function to attach to a queue, it becomes the owner of the queue. Only one application program can attach to a primary queue and read messages from it. When an application attaches to a permanent primary queue defined with secondary queue attachments, the secondary queues are also attached by this function.

Syntax

```
int32 pams_attach_q ( attach_mode, q_attached, [q_type], [q_name],
[q_name_len], [name_space_list], [name_space_list_len], [timeout],
[nullarg_2], [nullarg_3] )
```

Arguments

Table 2-2 Arguments

Argument	Data Type	Mechanism	Prototype	Access
attach_mode	int32		int32 *	passed
q_attached	q_address		q_address *	returned

Table 2-2 Arguments

Argument	Data Type	Mechanism	Prototype	Access
[q_type]	int32	reference	int32 *	passed
[q_name]	char	reference	char *	returned
[q_name_len]	int32	reference	int32 *	passed
[name_space_list]	int32 array	reference	int32 array *	passed
[name_space_list_len]	int32	reference	int32 *	passed
[timeout]	int32	reference	int32 *	passed
[nullarg_2]	char	reference	char *	passed
[nullarg_3]	char	reference	char *	passed

Argument Definitions

attach_mode

Supplies the mode for attaching the application to a message queue. The three predefined constants for this argument are:

- PSYM_ATTACH_BY_NAME-Attach by name
- PSYM_ATTACH_BY_NUMBER-Attach by number
- PSYM_ATTACH_TEMPORARY-Attach as a temporary queue

When **attach_mode** is PSYM_ATTACH_BY_NAME, the application attaches to the queue identified by the specified queue or alias name. Oracle MessageQ finds the queue by implicitly performing a pams_locate_q call for the specified **q_name**. The procedure that Oracle MessageQ uses is determined by the **name_space_list** argument.

q_attached

Receives the queue address from Oracle MessageQ when an application has successfully attached to a message queue.

q_type

Supplies the queue type for the attachment. The two predefined constants for this argument are:

- PSYM_ATTACH_PQ-Primary queue (default)
- PSYM_ATTACH_SQ-Secondary queue

q_name

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

References to queue names are case sensitive and must match the queue name entered in the group initialization file. Some example queue names are: QUEUE_1, high-priority, and My$Queue.

The **q_name** argument has the following dependencies with the **attach_mode** argument:

- If the **attach_mode** argument is PSYM_ATTACH_BY_NAME, the **q_name** argument must contain a valid queue name as specified during Oracle MessageQ group configuration.
- If the **attach_mode** argument is PSYM_ATTACH_BY_NUMBER, the **q_name** argument is specified as an ASCII string of 1 to 3 numeric characters representing the queue number.
- If the **attach_mode** argument is PSYM_ATTACH_TEMPORARY, the **q_name** argument is not used and should be specified by passing a value of 0.

q_name_len

Supplies the number of characters in the q_name argument. The maximum string length on UNIX, Windows NT is 127 characters.

name_space_list

Supplies a list of name tables to search when the **attach_mode** argument PSYM_ATTACH_BY_NAME is specified.

If the **name_space_list** is specified, then the **name_space_list_len** argument must also be specified. If this argument is unspecified, then PSEL_TBL_GRP is the default.

Possible values in a **name_space_list** argument are as follows:

Table 2-3 name_space_list argument values

Location It Represents	Symbolic Value
Process Cache	PSEL_TBL_PROC PSEL_TBL_PROC
Group/group cache	PSEL_TBL_GRP PSEL_TBL_GRP
Global name space	PSEL_TBL_BUS (or PSEL_TBL_BUS_MEDIUM PSEL_TBL_BUS_LOW)

The **name_space_list** argument identifies the scope of the name as follows:

- To identify a local queue reference or a queue, an application must include PSEL_TBL_GRP in **name_space_list.** (Do not specify PSEL_TBL_BUS in the list because it would identify a global queue reference.)
- To identify a global queue reference, include PSEL_TBL_BUS (or PSEL_TBL_BUS_MEDIUM or PSEL_TBL_BUS_LOW) in the **name_space_list** argument and specify its pathname, either explicitly or implicitly. If the **q_name** argument contains any slashes (/), or periods (.), Oracle MessageQ treats it as a pathname. Otherwise, Oracle MessageQ treats **q_name** as a name and adds the DEFAULT_NAMESPACE_PATH to the name to create the pathname to lookup. (The DEFAULT_NAMESPACE_PATH is set in the %PROFILE section of the group initialization file.)

The **name_space_list** argument also controls the cache access as follows.

- To cause the lookup of a local queue reference or queue name to check the process cache before looking in the group cache, specify the **name_space_list** argument as PSEL_TBL_GRP and PSEL_TBL_PROC.
- To cause the lookup of a global queue reference to check the process cache and then the group cache before looking into the global name space, specify PSEL_TBL_BUS(or PSEL_TBL_BUS_LOW or PSEL_TBL_BUS_MEDIUM), PSEL_TBL_GRP and PSEL_TBL_PROC.
- To lookup all caches in the global name space before looking in the master database, specify PSEL_TBL_BUS_LOW

- instead of PSEL_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 PSEL_TBL_BUS_MEDIUM instead of PSEL_TBL_BUS.

For more information on dynamic binding of queue addresses, see the Using Naming topic.

name_space_list_len
Supplies the number of entries in the **name_space_list** argument. If the **name_space_list_len** argument is zero, Oracle MessageQ uses PSEL_TBL_GRP as the default in the **name_space_list** argument.

timeout
The number of PAMS time units (1/10 second intervals) to allow for the attach to complete. If a zero is specified, the group's ATTACH_TMO property is used. If the ATTACH_TMO property is also zero, 600 is used.

nullarg_2
Reserved for Oracle MessageQ internal use as a placeholder argument. This argument must be supplied as a null pointer.

nullarg_3
Reserved for Oracle MessageQ internal use as a placeholder argument. This argument must be supplied as a null pointer.

Description

Before an application can use the pams_attach_q function, the Oracle MessageQ **message queuing bus** must be configured. A Oracle MessageQ message queuing bus is a collection of one or more Oracle MessageQ **message queuing groups**. A message queuing group 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. A Oracle MessageQ 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 Oracle MessageQ environment.

To receive Oracle MessageQ messages, an application must attach to at least one message queue. The pams_attach_q function enables an application to attach in the following ways:

- An application can attach to a queue by specifying a **number**. To attach by number, the message queue must be configured in the group definition. Attaching by number 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 **name**. To attach by name, the message queue must be configured in the group definition. Attaching by name 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 eliminates the need to change code or recompile if the queue address changes. Therefore, attaching by name protects applications from changes in the Oracle MessageQ 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 number. Oracle MessageQ will assign a queue and return the number 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 group definition 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 multireader queue. A multireader queue can be read by many applications and is configured as part of the group definition.

Return Values

Table 2-4 Return Codes

Return Code	Platform	Description
PAMS__BADARGLIST	OpenVMS	Wrong number of call arguments has been passed to this function.
PAMS__BADDECLARE	All	Queue has already been attached to this application.
PAMS__BADNAME	All	Invalid name string was specified.
PAMS__BADPARAM	All	Invalid argument in the argument list.

Table 2-4 Return Codes

Return Code	Platform	Description
PAMS__BADPROCNUM	All	Queue number out of range.
PAMS__BADQTYPE	All	Invalid queue type.
PAMS__BADTMPPROC	OpenVMS	Invalid temporary queue number.
PAMS__DECLARED	All	The queue number is already attached to another application or process.
PAMS__DUPLQNAME	OpenVMS	Duplicate queue name.
PAMS__NETERROR	Clients	Network error resulted in a communications link abort.
PAMS__NOACCESS	All	No access to the resource. The address of the specified name is either 0 or it is in another group.
PAMS__NOACL	All	The queue access control file could not be found.
PAMS__NOOBJECT	All	No such queue name. For a global queue reference, this error can be caused by a bad default pathname in the group configuration file.
PAMS__NOQUOTA	OpenVMS	Insufficient receive message or byte quota to attach.
PAMS__NOTBOUND	All	The queue name is not bound to an address.
PAMS__NOTMRQ	OpenVMS	Attempting to attach to Multi-reader Queue and queue type is not an MRQ.
PAMS__NOTPRIMARYQ	All	Queue name or number is not a primary queue.

Table 2-4 Return Codes

Return Code	Platform	Description
PAMS__NOTSECONDARYQ	All	Queue name or number is not a secondary queue.
PAMS__PAMSDOWN	All	The specified Oracle MessageQ group is not running.
PAMS__PREVCALLBUSY	Clients	The previous call to CLS has not been completed.
PAMS__PNUMNOEXIST	OpenVMS	Target queue name or number does not exist.
PAMS__RESRCFAIL	All	Failed to allocate resources.
PAMS__SUCCESS	All	Successful completion of an action.
PAMS__TIMEOUT	All	The timeout period specified has expired.
PAMS__NOQUOTA	OpenVMS	Insufficient receive message or byte quota to attach.
PAMS__NOTBOUND	All	The queue name is not bound to an address.
PAMS__NOTMRQ	OpenVMS	Attempting to attach to Multi-reader Queue and queue type is not an MRQ.
PAMS__NOTPRIMARYQ	All	Queue name or number is not a primary queue.
PAMS__NOTSECONDARYQ	All	Queue name or number is not a secondary queue.
PAMS__PAMSDOWN	All	The specified Oracle MessageQ group is not running.
PAMS__PREVCALLBUSY	Clients	The previous call to CLS has not been completed.

Table 2-4 Return Codes

Return Code	Platform	Description
PAMS__PNUMNOEXIST	OpenVMS	Target queue name or number does not exist.
PAMS__RESRCFAIL	All	Failed to allocate resources.
PAMS__SUCCESS	All	Successful completion of an action.
PAMS__TIMEOUT	All	The timeout period specified has expired.

See Also

- pams_detach_q
- pams_exit
- pams_locate_q

Examples

Attach by Name

this example illustrates how to attach to a queue by name. The name "example_q_1" must be defined in your group configuration information as a primary queue or as a local queue alias or a primary queue. The complete code example called x_attnam.c is contained in the examples directory.

Attach by Number

this example illustrates how to attach to a queue by number. A queue numbered 1 must be defined in your group configuration information file as a primary queue. The complete code example called x_attnum.c is contained in the examples directory.

Attach as Temporary

this example illustrates how to attach as a temporary queue. The complete code example called x_atttmp.c is contained in the examples directory.

pams_bind_q

Dynamically associates a queue address 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.

Syntax

```
int32 pams_bind_q (q_addr, q_alias, q_alias_len, [name_space_list],
[name_space_list_len], [timeout], [nullarg_1]);
```

Arguments

Table 2-5 Arguments

Argument	Data Type	Mechanism	Prototype	Access
q_addr	q_address	reference	q_address *	passed
q_alias	char	reference	char *	passed
q_alias_len	int32	reference	int32 *	passed
[name_space_list]	int32 array	reference	int32 array *	passed
[name_space_list_len]	int32	reference	int32 *	passed
[timeout]	int32	reference	int32 *	passed
[nullarg_1]	char	reference	char *	passed

Argument Definitions

q_addr

The value specified to this argument controls whether the queue address is bound or unbound:

- If the queue address is specified, this function binds it to a **q_alias**.
- If 0 is specified, this function unbinds the **q_alias** from its queue address. The calling application must be bound to **q_alias** to set it back to zero.

q_alias

Identifies a global queue reference or a local queue reference. The procedure that Oracle MessageQ uses to find this alias is controlled by the **name_space_list** argument, which is described below.

q_alias_len

Specifies the number of characters in **q_alias**.

name_space_list

If specified, identifies a one-entry list containing either PSEL_TBL_BUS or PSEL_TBL_GRP.

To identify a local queue reference, an application must have a name space list of PSEL_TBL_GRP and pass its name in the **q_alias** argument. To identify a global queue reference, an application must have a name space list of PSEL_TBL_BUS and specify its pathname, either explicitly or implicitly:

- If the **q_alias** argument contains any slashes (/), or periods (.), Oracle MessageQ treats the **q_alias** as a pathname.
- Otherwise, Oracle MessageQ treats **q_alias** as a name and adds the group's DEFAULT_NAMESPACE_PATH to the name to create the pathname to lookup. (The DEFAULT_NAMESPACE_PATH is set in the %PROFILE section of the initialization file.)

For more information on dynamic binding of queue addresses, see the Using Naming topic.

name_space_list_len

Specifies the number of entries in **name_space_list** argument. The number of entries is either 0 or 1. If the number of entries is 0 (indicating that the **name_space_list** is omitted), PSEL_TBL_GRP is assumed.

timeout

Specifies the number of PAMS time units (1/10 second intervals) to allow for the bind to complete. If 0 is specified, the group's ATTACH_TMO property is used. If the ATTACH_TMO property is also 0, 600 is used.

nullarg_1

Reserved for Oracle MessageQ internal use as a placeholder argument. This argument must be supplied as a null pointer.

Description

Before an application can call pams_bind_q, it must be attached to the specified queue address. Listing 2-1 shows an attach before the bind call and is typical usage of the two functions together:

Listing 2-1 Example of Using pams_bind_q

```
int32 mode = PSYM_ATTACH_BY_NUMBER;
             int32 q_type = PSYM_ATTACH_PQ;
             int32 len=1;
             int32 status;
             q_address qid;

status = pams_attach_q(&mode,&qid,&q_type,"2",&len,0,0,0,0,0);
       if (status == pams__SUCCESS {
              int32 ns=PSEL_TBL_BUS;
              int32 ns_len=1;
              len = strlen("Q2");

status = pams_bind_q(&qid,"Q2",&len,&ns,&ns_len,0,0);
```

Return Values

Table 2-6 Return Code

Return Code	Platform	Description
PAMS__BADARGLIST	All	Invalid number of call arguments.

Table 2-6 Return Code

PAMS__BADNAME	All	Name contains bad characters.
PAMS__BADPARAM	All	The name space list is invalid.
PAMS__BOUND	All	Returned if a non-zero value for **q_addr** is passed and the specified **q_alias** is already assigned to a queue address.
PAMS__DUPLQNAME	All	Duplicate queue name.
PAMS__FAIL	All	Operation failed.
PAMS__NOACCESS	All	No access to the resource. The address of the specified name is either 0 or it is in another group.
PAMS__NOOBJECT	All	For a global reference, this error can be caused by a bad default pathname in the group configuration file.
PAMS__NOTBOUND	All	The queue name is not bound to an address.
PAMS__NOTDCL	All	Not attached to Oracle MessageQ.
PAMS__PAMSDOWN	All	The specified Oracle MessageQ group is not running.
PAMS__SUCCESS	All	Indicates successful completion.
PAMS__TIMEOUT	All	The timeout period specified has expired. In this situation, Oracle MessageQ internally unbinds the specified queue alias. Subsequent PAMS_bind_qcalls to the same name will return the PAMS__UNBINDINGerror until the internal unbind succeeds.
PAMS__NOACCESS	All	No access to the resource. The address of the specified name is either 0 or it is in another group.
PAMS__NOOBJECT	All	For a global reference, this error can be caused by a bad default pathname in the group configuration file.
PAMS__NOTBOUND	All	The queue name is not bound to an address.

Table 2-6 Return Code

PAMS__NOTDCL	All	Not attached to Oracle MessageQ.
PAMS__PAMSDOWN	All	The specified Oracle MessageQ group is not running.
PAMS__SUCCESS	All	Indicates successful completion.
PAMS__TIMEOUT	All	The timeout period specified has expired. In this situation, Oracle MessageQ internally unbinds the specified queue alias. Subsequent PAMS_bind_qcalls to the same name will return the PAMS__UNBINDINGerror until the internal unbind succeeds.
PAMS__UNBINDING	All	The bind cannot be done because Oracle MessageQ is still in the process of has unbinding the old binding.

See Also

- pams_attach_q
- pams_locate_q

Example

The pams_bind_q example illustrates how to bind a queue reference to a queue address at runtime. The complete code example called x_bind.c is contained in the examples directory.

pams_cancel_get

Cancels all pending pams_get_msga requests that match the value specified in the **sel_filter** argument. When a pending pams_get_msga request is canceled, the PAMS Status Block (PSB) delivery status is set to pams_CANCEL and the specified action routine is queued. The pams_cancel_get function waits until completion to allow for proper synchronization between the pams_cancel_get function and the request for pams_get_msga functions. Any outstanding pams_get_msga function requests are canceled by the pams_exit function or at image exit.

Syntax

```
int32 pams_cancel_get ( sel_filter )
```

Arguments

Table 2-7 Arguments

Argument	Data Type	Mechanism	Prototype	Access
sel_filter	int32	reference	int32 *	passed

Argument Definition

sel_filter

Supplies the criteria that enables the application to selectively cancel outstanding `pams_get_msga` requests. For a description of the **sel_filter** argument, see the `pams_get_msg` function. For a description of how to create a complex selection filter, see the `pams_set_select` function.

Return Values

Table 2-8 Return Codes

Return Code	Platform	Description
PAMS__BADARGLIST	OpenVMS	Argument list is invalid.
PAMS__SUCCESS	OpenVMS	Indicates successful completion.
SS$_EXQUOTA	OpenVMS	Process has exceeded its asynchronous system trap (AST) quota.

See Also

- `pams_cancel_select`
- `pams_get_msga`
- `pams_set_select`

pams_cancel_select

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

pams_set_select function. When the selection mask is used in the OpenVMS environment with asynchronous read requests, this function also cancels any pending pams_get_msga requests that use the referenced **index_handle**.

Syntax

```
int32 pams_cancel_select ( index_handle )
```

Arguments

Table 2-9 Arguments

Argument	Data Type	Mechanism	Prototype	Access
index_handle	int32	reference	int32 *	passed

Argument Definitions

index_handle

Supplies the index handle of the selection mask to cancel. The **index_handle** is returned by the pams_set_select function.

Return Values

Table 2-10 Return Codes

Return Code	Platform	Description
PAMS__BADARGLIST	OpenVMS	Invalid number of call arguments.
PAMS__BADPARAM	UNIX Windows NT	The value of the selection index is null.
PAMS__BADSELIDX	All	Invalid or undefined selective receive index.
PAMS__NETERROR	Clients	Network error resulted in a communications link abort.
PAMS__NOTDCL	All	Process has not been attached to Oracle MessageQ.

Table 2-10 Return Codes

PAMS__PAMSDOWN	UNIX Windows NT	The specified Oracle MessageQ group is not running.
PAMS__PREVCALLBUSY	Clients	Previous call to CLS has not been completed.
PAMS__SUCCESS	All	Indicates successful completion.

See Also

- pams_get_msga
- pams_set_select

pams_cancel_timer

Deletes the Oracle MessageQ timer identified by the **timer_id** argument that is passed to this function. All expired timers with the selected identification code that are waiting in the message queue are purged and are not delivered.

Syntax

int32 pams_cancel_timer (timer_id)

Arguments

Table 2-11 Arguments

Argument	Data Type	Mechanism	Prototype	Access
timer_id	int32	reference	int32 *	passed

Argument Definitions

timer_id

Supplies the timer ID of the timer to cancel. The timer_id is returned by the pams_set_timer function.

Return Values

Table 2-12 Return Codes

Return Code	Platform	Description
PAMS__BADARGLIST	OpenVMS	Invalid number of arguments.
PAMS__BADPARAM	All	The **timer_id** argument was specified as null.
PAMS__INVALIDNUM	All	The application has supplied an invalid value for the **timer_id**.
PAMS__NETERROR	Clients	Network error resulted in a communications link abort.
PAMS__NOTDCL	All	The application has not attached to a queue.
PAMS__PAMSDOWN	UNIX Windows NT	The specified Oracle MessageQ group is not running.
PAMS__PREVCALLBUSY	Clients	Previous call to CLS has not been completed.
PAMS__RESRCFAIL	All	Insufficient resources to complete the operation.
PAMS__SUCCESS	All	Indicates successful completion.

See Also

- pams_set_timer

pams_close_jrn

Closes the MRS journal file associated with the **jrn_handle** argument. The two types of journal files are dead letter journal (DLJ) and post confirmation journal (PCJ). See Using Recoverable Messaging for a description of how to use the Oracle MessageQ message recovery system.

Syntax

```
int32 pams_close_jrn ( jrn_handle )
```

Arguments

Table 2-13 Arguments

Argument	Data Type	Mechanism	Prototype	Access
jrn_handle	int32	reference	int32 *	passed

Argument Definitions

Jrn_handle

Supplies the journal handle of the message recovery journal file to close. The **jrn_handle** is returned by the pams_open_jrn function.

Return Values

Table 2-14 Return Codes

Return Code	Platform	Description
PAMS__BADARGLIST	OpenVMS	Invalid number of arguments.
PAMS__INVJH	OpenVMS	The application has supplied an invalid journal handle.
PAMS__SUCCESS	OpenVMS	Indicates successful completion.

See Also

- pams_confirm_msg
- pams_open_jrn
- pams_read_jrn

pams_confirm_msg

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 pams_confirm_msg 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 redelivered if the application detaches and then reattaches to the queue.

Oracle MessageQ 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 the installation and configuration guide for your system.

Successfully delivered recoverable messages can be recorded in the post confirmation journal (PCJ). The pams_confirm_msg 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 pams_confirm_msg function.

Syntax

```
int32 pams_confirm_msg ( msg_seq_num, confirmation_status, force_j )
```

Arguments

Table 2-15 Arguments

Argument	Data Type	Mechanism	Prototype	Access
msg_seq_num	uint32 array	reference	uint32 array *	passed
confirmation_status	int32	reference	int32 *	passed
force_j	char	reference	char *	passed

Argument Definitions

msg_seq_num

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

confirmation_status

Supplies the confirmation status value stored with the message in the post confirmation journal (PCJ) file. The value is set by the calling application. See the Using Recoverable Messaging topic for more information on using the PCJ file.

force_j

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

Table 2-16 force_j

Symbol	Description
PDEL_DEFAULT_JRN	Enables writing the message to the PCJ file if the journaling is configured in the group initialization file.
PDEL_FORCE_JRN	Enables writing to the PCJ only if journaling is configured. It is not possible to write messages to the PCJ on UNIX and Windows NT systems if a path was not defined for the PCJ in the group configuration information.
PDEL_NO_JRN	Disables journaling regardless of whether journaling is configured.

Description

The PSB status codes associated with recoverable message delivery are PAMS__CONFIRMREQ and PAMS__POSSDUPL. The PAMS__CONFIRMREQ PSB status code indicates that it is the first time the application received the recoverable message. The PAMS_POSSDUPL status code indicates that the message was retrieved from the recovery journal and may have been sent previously. This status code allows the application to take extra precautions to handle duplicate messages if necessary.

The PSB also contains a sequence number that uniquely identifies the message. The pams_confirm_msg function requires this sequence number. If one of these status codes is

present and the pams_confirm_msg function is not called, the message will continue to be stored by the message recovery system and will be delivered again if the application exits and then reattaches.

Return Values

Table 2-17 Return Code

Return Code	Platform	Description
PAMS__BADARGLIST	OpenVMS	Invalid number of arguments.
PAMS__BADPARAM	All	Bad argument value.
PAMS__BADSEQ	All	Journal sequence number is not known to the Message Recovery Services (MRS).
PAMS__DQF_DEVICE_FAIL	OpenVMS	I/O error writing to the destination queue file for the target queue.
PAMS__NETERROR	Clients	Network error resulted in a communications link abort.
PAMS__NOMRS	All	MRS is not available.
PAMS__NOTDCL	All	Process is not attached to Oracle MessageQ.
PAMS__NOTJRN	All	Message is not written to the PCJ file.
PAMS__NOTSUPPORTED	OpenVMS	Attached to the dead letter queue.
PAMS__PAMSDOWN	UNIX Windows NT	The specified Oracle MessageQ group is not running.
PAMS__PREVCALLBUSY	Clients	Previous call to CLS has not been completed.
PAMS__RESRCFAIL	OpenVMS	Oracle MessageQ resources exhausted.
PAMS__SUCCESS	All	Indicates successful completion.

See Also

- pams_get_msg
- pams_get_msga
- pams_get_msgw
- pams_put_msg

Example

Confirm Receipt of Recoverable Messages

This example demonstrates using recoverable messaging. It attaches to queue_1, puts some recoverable messages to queue_2, exits, attaches to queue_2, gets the messages, prints them out, then exits.

The queues named "queue_1" and "queue_2" are defined in your initialization file. On OpenVMS systems, you must set up a DQF for queue_2. The complete code example called x_recovr.c is contained in the examples directory.

pams_detach_q

Detaches a selected message queue or all of the application's message queues from the message queuing bus. 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 Oracle MessageQ message queuing bus.

Syntax

```
int32 pams_detach_q ( q, detach_opt_list, detach_opt_len,msgs_flushed )
```

Arguments

Table 2-18 Arguments

Argument	Data Type	Mechanism	Prototype	Access
q	q_address	reference	q_address	passed
detach_opt_list	int32 array	reference	int32 *	passed

Table 2-18 Arguments

detach_opt_len	int32	reference	int32 *	passed
msg_flushed	int32	reference	int32 *	returned

Argument Definitions

q

Supplies the queue address of the queue to be detached. This function can be used to detach primary, secondary, and multireader queues.

detach_opt_list

Supplies an array of int32 values used to control how the queue is detached. The predefined constants for this argument are:

- PSYM_NOFLUSH_Q-Detaches the queue without flushing the pending messages stored in memory. The default action is to flush pending messages in the queue before it is detached. Messages are never flushed from multireader queues.
- PSYM_DETACH_ALL-Detaches all of the application's message queues from the message queuing bus. Using this constant performs the same action as calling the pams_exit function.
- PSYM_CANCEL_SEL_MASK-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, Oracle MessageQ invalidates all selection masks that reference the queue or queues that you are detaching. You must cancel the invalidated selection masks using the pams_cancel_select function.

detach_opt_len

Supplies the number of int32 values in the **detach_opt_list** array. The maximum number of int32 longwords is 32,767.

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.

Description

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 pams_detach_q
- Detaching from the message queuing bus by calling pams_exit
- Exiting your application

If you do not ensure that the last message was confirmed before detaching or exiting, the message will be redelivered 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.

Return Values

Table 2-19 Return Codes

Return Code	Platform	Description
PAMS__BADARGLIST	OpenVMS	Invalid number of arguments.
PAMS__BADPARAM	All	Invalid **detach_opt_list**.
PAMS__DETACHED	All	Process has detached from Oracle MessageQ.
PAMS__NETERROR	Clients	Network error resulted in a communications link abort.
PAMS__NOTDCL	All	Not attached to Oracle MessageQ.
PAMS__PREVCALLBUSY	Clients	Previous call to CLS has not been completed.
PAMS__PNUMNOEXIST	All	Invalid queue address or queue not owned by process.
PAMS__SUCCESS	All	Queue successfully detached.

See Also

- pams_attach_q
- pams_exit

pams_exit

Terminates all attachments between the application and the Oracle MessageQ message queuing bus. All pending messages in temporary queues and permanent queues which are not permanently active multi-reader queues are discarded. Only the messages pending in permanently active multi-reader queues are retained. To retain messages in permanently active queues, call **pams_detach_q** with option `PSYM_NOFLUSH_Q` before calling `pams_exit`.

Syntax

```
int32 pams_exit (void)
```

Arguments

None.

Description

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 `pams_detach_q`
- Detaching from the message queuing bus by calling `pams_exit`
- Exiting your application

If you do not ensure that the last message was confirmed before detaching or exiting, the message will be redelivered 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.

Return Values

Table 2-20 Return Codes

Return Code	Platform	Description
`PAMS__NETERROR`	OpenVMS Client	Network error resulted in a communications link abort.
`PAMS__NOTDCL`	OpenVMS	Not attached to Oracle MessageQ.
`PAMS__PREVCALLBUSY`	OpenVMSClient	Previous call to CLS has not been completed.

Table 2-20 Return Codes

PAMS__PNUMNOEXIST	OpenVMS	Invalid queue address or queue not owned by process.
PAMS__SUCCESS	All	Indicates successful completion.

See Also

- pams_attach_q
- pams_detach_q

Example

Exit the Message Queuing Bus

This example shows how to use the pams_exit function. The complete code example called x_exit.c is contained in the examples directory.

pams_get_msg

Retrieves the next available message from a selected queue and moves it to the location specified in the **msg_area** 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 **msg_area** 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 pams__NOMOREMSG.

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 pams_confirm_msg 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 redelivered if the application detaches and then reattaches to the queue.

The receiver program determines whether each message is a FML32 buffer or large message by reading the **msg_area_len** argument. See the Sending and Receiving Oracle MessageQ Messages topic for more information on working with FML32 buffers and large messages.

Syntax

```
int32 pams_get_msg ( msg_area, priority, source, class, type, msg_area_len,
len_data, [sel_filter], [psb], [show_buffer], [show_buffer_len],
[large_area_len], [large_size], [nullarg_3] )
```

Arguments

Table 2-21 Arguments

Argument	Data Type	Mechanism	Prototype	Access
msg_area	char	reference	char *	returned
priority	char	reference	char *	passed
source	q_address	reference	q_address *	returned
class	short	reference	short *	returned
type	short	reference	short *	returned
msg_area_len	short	reference	short *	passed
len_data	short	reference	short*	returned
[sel_filter]	int32	reference	int32 *	passed
[psb]	struct psb	reference	struct psb *	
[show_buffer]	struct show_buffer	reference	struct show_buffer *	returned
[show_buffer_len]	int32	reference	int32 *	passed
[large_area_len]	int32	reference	int32 *	passed/

Table 2-21 Arguments

[large_size]	int32	reference	int32 *	returned
[nullarg_3]	char	reference	char*	passed

Argument Definitions

msg_area

For static buffer-style messaging, receives the address of a memory region where Oracle MessageQ writes the contents of the retrieved message. For FML-style messaging or when using double pointers, receives a pointer to the address of the message being retrieved.

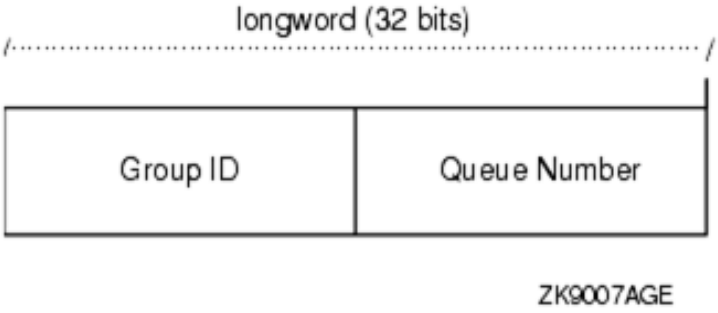
priority

Supplies the priority level for selective message reception. Priority ranges from 0 (lowest priority) to 99 (highest priority). If the priority is set to 0, the pams__get_msqw function gets messages of any priority. If the priority is set to any value from 1 to 99, the pams__get_msqw function gets only messages of that priority.

source

Receives a data structure containing the group ID and queue number of the sender program's primary queue in the following format:

Figure 2-1



class

Receives the class code of the retrieved message. The class is specified in the pams_put_msg function. Oracle MessageQ 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 p_typecl.h include file. On UNIX and Windows NT systems, the p_typecl.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 Oracle MessageQ are as follows:

Table 2-22 Class symbols

Reserved Class	Symbol Value
MSG_CLAS_MRS	28
MSG_CLAS_PAMS	29
MSG_CLAS_ETHERNET	100
MSG_CLAS_UCB	102
MSG_CLAS_TUXEDO	31001
MSG_CLAS_TUXEDO_TPSUCCESS	31002
MSG_CLAS_TUXEDO_TPFAIL	31003
MSG_CLAS_XXX	30000 through 32767 (except 31001-31003)

type

Receives the type code of the retrieved message. The type is specified in the pams_put_msg function. Oracle MessageQ supports the use of symbolic names for **type** argument values. Symbolic type names begin with MSG_TYPE_. For specific information on defining type symbols, see the p_typecl.h include file.

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

msg_area_len

- Supplies the size of the buffer (in bytes) for static message buffers of up to 32767 bytes. The **msg_area** buffer is used to store the retrieved message.

- For messages using double buffers, including FML32 buffers, this argument contains the symbol PSYM_MSG_BUFFER_PTR to indicate that the message is a pointer to the address of the message being retrieved. The **msg_area** buffer contains the message pointer. The size of the message is returned in the **large_size** argument. The **msg_area** buffer is used to store the retrieved message. The **large_area_len** argument is used to supply the size of the message buffer to receive the message. If the retrieved buffer is larger than the space allocated, space is dynamically reallocated and the new buffer size is stored in **large_area_len**.
- For large messages (buffer-style messages larger than 32767 bytes), this argument contains the symbol PSYM_MSG_LARGE to indicate that the message buffer is greater than 32K. The size of the message is returned in the **large_size** argument. The **msg_area** buffer is used to store the retrieved message. The **large_area_len** argument is used to supply the size of the message buffer to receive the large message.

len_data

For static buffer-style messaging with messages of up to 32767 bytes, this argument receives the number of bytes retrieved from the message queue and stored in the area specified by the msg_area argument. This field also receives the PSYM_MSG_BUFFER_PTR symbol for double buffer and FML-style messages and PSYM_MSG_LARGE for buffer-style messages larger than 32767 bytes.

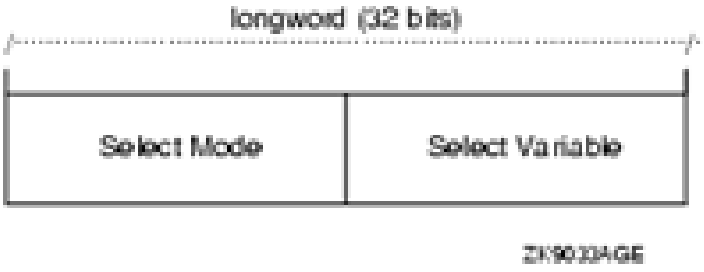
sel_filter

Supplies the criterion to enable the application to selectively retrieve messages. The argument contains one of the following selection criteria:

- Default selection
- Selection by message queue
- Message attributes
- Message source
- Compound select using the pams_set_select function .

The **sel_filter** argument is composed of two words as follows:

Figure 2-2



Default Selection

Enables applications to read messages from the queue based on the order in which they arrived. The default selection,

PSEL_DEFAULT, reads the next pending message from the message queue. Messages are stored by priority and then in FIFO order. To specify this explicitly, both words in the **sel_filter** argument should be set to 0.

Selection by Message Queue

Allows the application to retrieve messages based upon a queue type or combination of queue types. This selection criteria is used to retrieve the first pending message that matches the criteria on the first queue it encounters. FIFO ordering is maintained within each queue. The predefined constants for this argument are as follows:

Table 2-23 Selection by Message Queue

Select Mode	Select Variable	Mode Description
PSEL_PQ	0	Enables the application to read from the primary queue (PQ) only. The select variable must equal 0.
PSEL_AQ	Alternate queue number	Enables an application to read from an alternate queue (AQ) only. The queue type can be a secondary queue (SQ).

Table 2-23 Selection by Message Queue

PSEL_PQ_AQ	Alternate queue number	Attempts to selectively retrieve from a primary queue and then from an alternate queue.
PSEL_AQ_PQ	Alternate queue number	Attempts to selectively retrieve from an alternate queue and then from a primary queue.
PSEL_TQ_PQ	Alternate queue number	Attempts to selectively retrieve messages from a timer queue (TQ), and then from a primary queue.
PSEL_TQ_PQ_AQ	Alternate queue number	Attempts to selectively retrieve messages from a timer queue (TQ), then from a primary queue, and finally from an alternate queue.
PSEL_UCB	0	Retrieves messages only from the user callback queues (UCB).

Selection by Message Attribute

Enables the application to select messages based on the message type, class, or priority. The predefined constants for this argument are as follows:

Table 2-24 Selection by Message Attribute

Select Mode	Select Variable	Mode Description
PSEL_PQ_TYPE	Type	Selects the first pending message from the primary queue that matches the type value in the select variable word.

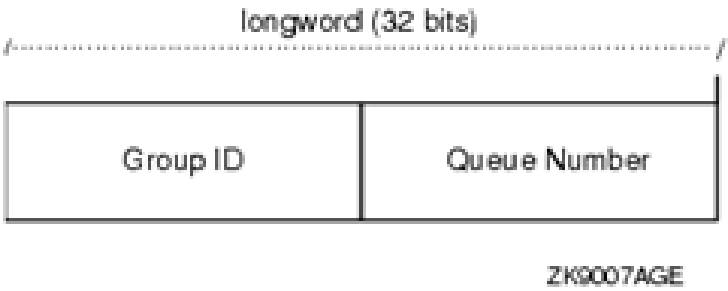
Table 2-24 Selection by Message Attribute

Select Mode	Select Variable	Mode Description
PSEL_PQ_CLASS	Class	Selects the first pending message from the primary queue that matches the class value in the select variable word.
PSEL_PQ_PRI	PSEL_PRI_ANY PSEL_PRI_P0 PSEL_PRI_P1 integer value between 0 and 99	Selects the first pending message with a priority equal to an integer between 0 and 99 inclusive (or equal to the select variable value) from within the primary queue. Specifying the direct integer value is the preferred method of selectedmessages by priority. Using PSEL_PRI_ANY enables the reading of any pending messages of all priorities. Setting PSEL_PRI_P0 enables the application to retrieve pending messages of priority 0 only. Setting PSEL_PRI_P1 enables the strict retrieval of pending messages with a priority of 1.

Selection by Message Source

Provides for the selection of pending messages from primary and secondary queues, by source group ID, queue number, or both. The format for selection by source -follows:

Figure 2-3



Some examples of possible **sel_filter** arguments and their actions are as follows:

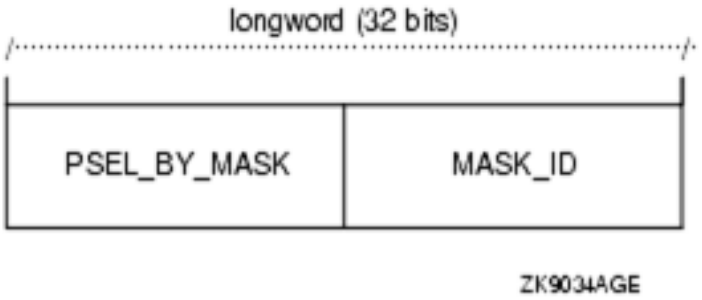
Table 2-25 sel_filter arguments

sel_filter Argument	Action
Zero or not specified	No filtering of any messages. All messages can be retrieved.
Source **q_address**	Only those messages that have a matching **q_address** are retrieved.
Selection mask created with `pams_set_select`	Only messages that exactly match the specified selection mask are retrieved.

Compound Selection

Allows the application to formulate complex rules for the order in which the message queues are searched. The `pams_set_select` function allows the application to create custom selection masks that can be used in the low-order word of the **sel_filter** argument. The format for compound selection follows:

Figure 2-4



psb

Receives a PAMS Status Block containing 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. The structure of the PSB is as follows:

Table 2-26 PSB Structure

Low Byte	High Byte	Contents	Description
0	1	Type	PSB type
2	3	Call Dependent	Currently not used.
4	7	PSB Delivery Status	The completion status of the function. For recoverable messages, this field contains PAMS__CONFIRMREQ or PAMS__POSSDUPL. For nonrecoverable messages, it may also contain a value of PAMS__SUCCESS.
8	15	Message Sequence Number	A unique number assigned to a message when it is sent and follows the message to the destination PSB. This number is input to pams_confirm_msgto release a recoverable message.
16	19	PSB UMA Status	This field is not used for the pams_get_msg function.
20	23	Function Return Status	This field is not used for the pams_get_msg function.
24	31	Not Used	Not used.

show_buffer

Receives additional information which Oracle MessageQ extracts from the message header. The structure of the **show_buffer** argument is as follows:

Table 2-27 show_buffer argument

Longword	Contents	Description
0	Version	The version of the show_buffer structure. Valid values are as follows: 10 = Version 1.0 20 = Version 2.0 50 = Version 5.0
1	Transfer Status	The status code associated with the transfer of **show_buffer** information into the application's buffer. Valid symbols are as follows: PAMS__SUCCESS All available information has been transferred. PAMS__BUFFEROVF Information was lost due to receiver buffer overflow. 0-No message returned. There is no information to transfer.
2	Transfer Size	The number of bytes transferred to the application buffer.

Table 2-27 show_buffer argument

3	Flags	A bit array showing the status of fields in the `show_buffe`r. A set bit indicates a valid field, while a cleared bit indicates indeterminable data or the end of the allocated `show_buffer` memory. The symbols for the flags field are as follows:
		`PSYM_SHOW_VERSION`
		`PSYM_SHOW_STATUS`
		`PSYM_SHOW_SIZE`
		`PSYM_SHOW_FLAGS`
		`PSYM_SHOW_TARGET`
		`PSYM_SHOW_ORIGINAL_TARGET`
		`PSYM_SHOW_SOURCE`
		`PSYM_SHOW_ORIGINAL_SOURCE`
		`PSYM_SHOW_DELIVERY` D
		`PSYM_SHOW_PRIORITY`
		`PSYM_SHOW_ENDIAN`
		`PSYM_SHOW_CORRELATION_I`

4	Not Used	Fills out the Control Section to its maximum 24 bytes.
5	Not Used	Fills out the Control Section to its maximum 24 bytes.
6	Not Used	Fills out the Control Section to its maximum 24 bytes.
7	Not Used	Fills out the Control Section to its maximum 24 bytes.
8	Not Used	Fills out the Control Section to its maximum 24 bytes.
9	Not Used	Fills out the Control Section to its maximum 24 bytes.
10	Target	The **q_address** of the latest message target.

Table 2-27 show_buffer argument

11	Original Target	The **q_address** of the original message target.
12	Source	The **q_address** of the latest message source.
13	Original Source	The **q_address** of the original message.
14	Delivery Mode	The delivery mode that was used to queue the message. This is not necessarily the delivery mode used to generate the message.
15	Priority	The priority used to queue the message.
16	Endian	The byte ordering or encoding schemes of 2- and 4-byte integers. The possible settings are as follows: PSYM_UNKNOWN PSYM_VAX_BYTE_ORDER or PSYM_LITTLE_ENDIAN PSYM_NETWORK_BYTE_ORDER or PSYM_BIG_ENDIAN PSYM_FML
17	Correlation I	The 32 byte correlation ID associated with the message.

show_buffer_len

Supplies the length in bytes of the buffer defined in the **show_buffer** argument. The minimum length is 40 bytes. If the buffer is too small to contain all of the information, then the return code PAMS__BUFFEROVF will be in the **show_buffer** transfer status.

large_area_len

Specifies the size of the message area to receive messages larger than 32K. Also specifies the length of the message buffer when using double buffers (as indicated by **PSYM_MSG_BUFFER_POINTER**). This argument also stores the length of double buffers and FML32 buffers after reallocation.

large_size

Returns the actual size of the large message, double buffer message, or FML32 message written to the message buffer.

nullarg_3

Reserved for Oracle MessageQ internal use as a placeholder argument. This argument must be supplied as a null pointer.

Return Values

Table 2-28 Return Code

Return Code	Platform	Description
PAMS__AREATOSMALL	All	Received message is larger than the user's message area.
PAMS__BADARGLIST	All	Wrong number of call arguments have been passed to this function.
PAMS__BADHANDLE	All	Invalid message handle.
PAMS__BADPARAM	All	Bad argument value.
PAMS__BADPRIORITY	All	Invalid priority value used for receive.
PAMS__BADSELIDX	All	Invalid or undefined selective receive index.
PAMS__BUFFEROVF	UNIX Windows NT	The size of the show_buffer specified is too small.
PAMS__EXHAUSTBLKS	OpenVMS	No more message blocks available.
PAMS__FMLERROR	All	Problem detected with internal format of FML message; this can be an error in processing or data corruption.
PAMS__INSQUEFAIL	All	Failed to properly queue a message buffer.
PAMS__MSGTOSMALL	All	The **msg_area_len** argument must be positive or zero.
PAMS__MSGUNDEL	All	Message returned is undeliverable.

Table 2-28 Return Code

PAMS__NEED_BUFFER_PTR	UNIX Windows NT	FML32 buffer received but **msg_area_len** argument not set to PSYM_MSG_BUFFER_PTR.
PAMS__NETERROR	Clients	Network error resulted in a communications link abort.
PAMS__NOACCESS	All	No access to resource.
PAMS__NOACL	All	Queue access control file could not be found.
PAMS__NOMEMORY	OpenVMS	Insuffucient memory resources to reallocate buffer pointer.
PAMS__NOMRQRESRC	All	Insufficient multireader queue resources to allow access.
PAMS__NOTDCL	All	Process has not been attached to Oracle MessageQ.
PAMS__NOTSUPPORTED	UNIX Windows NT	The supplied delivery mode is not -supported.
PAMS__PAMSDOWN	UNIX Windows NT	The specified Oracle MessageQ group is not running.
PAMS__PREVCALLBUSY	Clients	Previous call to CLS has not been completed.
PAMS__QUECORRUPT	OpenVMS	Message buffer queue corrupt.
PAMS__REMQUEFAIL	All	Failed to properly read from a message buffer.
PAMS__STALE	All	Resource is no longer valid and must be freed by the user.
PAMS__STOPPED	All	The requested queue has been stopped.
PAMS__SUCCESS	All	Indicates successful completion.

PBS Delivery Status

Table 2-29 PBS Delivery Status

PSB Delivery Status	Platform	Description
PAMS__CONFIRMREQ	All	Confirmation required for this message.
PAMS__PAMSDOWN	UNIX Windows NT	The specified Oracle MessageQ group is not running.
PAMS__POSSDUPL	All	Message is a possible duplicate.
PAMS__SUCCESS	All	Indicates successful completion.

See Also

- pams_get_msga
- pams_get_msgw
- pams_put_msg
- pams_set_select

Example

Read a Message

This example uses the pams_get_msg function to retrieve all the messages currently in the queue and sends them to a print function. The complete code example called x_get.c is contained in the examples directory.

pams_get_msga

The pams_get_msga function is only available on OpenVMS systems.

Requests asynchronous notification of a message arrival. The pams_get_msga function triggers an asynchronous system trap (AST) routine when a message arrives in that queue. Notification to the application occurs by triggering an AST, by setting an event flag, or both.

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 user-supplied **msg_area** 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, and the AST or event flag is triggered only when a matching message arrives.

If a queue has been sent a recoverable message, the receiver program can confirm receipt of the message using the pams_confirm_msg function. The pams_confirm_msg function enables the successfully delivered message to be deleted from the message recovery system. See the Using Recoverable Messaging topic for a description of the Oracle MessageQ recovery system.

See the Sending and Receiving Oracle MessageQ Messages topic for more information on working with FML32 buffers and large messages.

Syntax

```
int32 pams_get_msga ( msg_area, priority, source, class, type,
msg_area_len, len_data, [sel_filter], [psb], [show_buffer],
[show_buffer_len], [large_area_len], [large_size], [actrtn], [actparm],
[flag_id], [nullarg_3] )
```

Arguments

Table 2-30 Arguments

Argument	Data Type	Mechanism	Prototype	Access
msg_area	char	reference	char *	returned
priority	char	reference	char *	passed
source	q_address	reference	q_address *	returned
class	short	reference	short *	returned
type	short	reference	short *	returned

Table 2-30 Arguments

msg_area_len	short	reference	short *	passed
len_data	short	reference	short *	returned
[sel_filter]	int32	reference	int32 *	passed
[psb]	struct psb	reference	struct psb *	returned
[show_buffer]	struct show_buffer	reference	struct show_buffer *	returned
[show_buffer_len]	int32	reference	int32 *	passed
[large_area_len]	int32	reference	int32 *	passed/ returned
[large_size	int32	reference	int32 *	returned
[actrtn]	int32	value	int32 *	passed
[actparm]	int32	reference	int32 *	passed
[flag_id]	int32	reference	int32 *	passed
[nullarg_3]	char	reference	char *	passed

Argument Definitions

msg_area

For static buffer-style messaging, receives the address of a memory region where Oracle MessageQ writes the contents of the retrieved message. For FML-style messaging or when using double pointers, receives a pointer to the address of the message being retrieved. When using double buffer pointers with pams_get_msga, the new buffer size is returned in large_size. (This differs from pams_get_msg[w}, where the new buffer size is returned in large_area_len.)

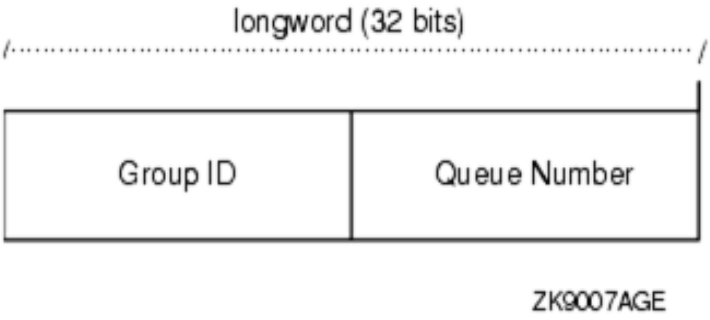
priority

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

source

Receives a data structure containing the group ID and queue number of the sender program's primary queue in the following format:

Figure 2-5 Group ID Data Structure



class

Receives the class code of the retrieved message. The class is specified in the `pams_put_msg` function. Oracle MessageQ 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 `p_typecl.h` include file.

Class symbols reserved by Oracle MessageQ are as follows:

Table 2-31 Class Symbols

Reserved Class	Symbol Value
MSG_CLAS_MRS	28
MSG_CLAS_PAMS	29
MSG_CLAS_ETHERNET	100
MSG_CLAS_UCB	102

Table 2-31 Class Symbols

MSG_CLAS_TUXEDO	31001
MSG_CLAS_TUXEDO_TPSUCCESS	31002
MSG_CLAS_TUXEDO_TPFAIL	31003
MSG_CLAS_XXX	30000 through 32767 (except 31001-31003)

type

Receives the type code of the retrieved message. The type is specified in the pams_put_msg function. Oracle MessageQ supports the use of symbolic names for **type** argument values. Symbolic type names begin with MSG_TYPE_. For specific information on defining type symbols, see the p_typecl.h include file.

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

msg_area_len

- Supplies the size of the buffer (in bytes) for buffer-style messages of up to 32767 bytes. The **msg_area** buffer is used to store the retrieved message.
- For messages using double buffers, including FML32 buffers, this argument contains the symbol PSYM_MSG_BUFFER_PTR to indicate that the message is a pointer to the address of the message being retrieved. The **msg_area** buffer contains the message pointer. The size of the message is returned in the **large_size** argument. The **msg_area** buffer is used to store the retrieved message. The **large_area_len** argument is used to supply the size of the message buffer to receive the message. If the retrieved buffer is larger than the space allocated, space is dynamically reallocated and the new buffer size is stored in **large_size**.
- For large messages (buffer-style messages larger than 32767 bytes), this argument contains the symbol PSYM_MSG_LARGE to indicate that the message buffer is greater than 32K. The size of the message is returned in the **large_size** argument The **msg_area** buffer is used to store the retrieved message. The **large_area_len** argument is used to supply the size of the message buffer to receive the large message.

len_data

For static buffer-style messaging with messages of up to 32767 bytes, this argument receives the number of bytes retrieved from the message queue and stored in the area specified by the **msg_area** argument. This field also receives the PSYM_MSG_BUFFER_PTR symbol for FML-style messages and PSYM_MSG_LARGE for buffer-style messages larger than 32767 bytes.

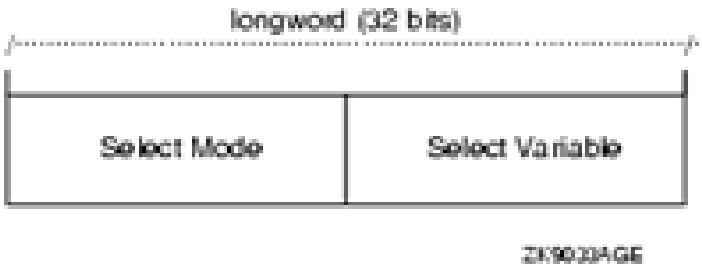
sel_filter

Supplies the criteria enabling the application to selectively retrieve messages. The argument contains one of the following selection criteria:

- Default selection
- Selection by message queue
- Message attributes
- Message source
- Compound selection using the pams_set_select function

The **sel_filter** argument is composed of two words as follows:

Figure 2-6 sel_filter argument



Default Selection

Enables applications to read messages from the queue based on the order in which they arrived. The default selection, PSEL_DEFAULT, reads the next pending message from the message queue. Messages are stored by priority and then in FIFO order. To specify this explicitly, both words in the **sel_filter** argument should be set to 0.

Selection by Message Queue

Allows the application to retrieve messages based upon a queue type or combination of queue types. This selection criteria is used to retrieve the first pending message that matches the criteria on the first queue it encounters. FIFO ordering is maintained within each queue.

The predefined constants for this argument are as follows:

Table 2-32 sel_filter argument

Select Mode	Select Variable	Mode Description
PSEL_PQ	0	Enables the application to read from the primary queue (PQ) only. The select variable must equal 0.
PSEL_AQ	Alternate queue number	Enables an application to read from an alternate queue (AQ) only. The queue type can be a secondary queue (SQ).
PSEL_PQ_AQ	Alternate queue number	Attempts to selectively retrieve from a primary queue and then from an alternate queue.
PSEL_AQ_PQ	Alternate queue number	Attempts to selectively retrieve from an alternate queue and then from a primary queue.
PSEL_TQ_PQ	Alternate queue number	Attempts to selectively retrieve messages from a timer queue (TQ), and then from a primary queue.
PSEL_TQ_PQ_AQ	Alternate queue number	Attempts to selectively retrieve messages from a timer queue (TQ), then from a primary queue, and finally from an alternate queue.
PSEL_UCB	0	Retrieves messages only from the user callback queues (UCB).

Selection by Message Attribute

Enables the application to select messages based on the message type, class, or priority. The predefined constants for this argument are as follows:

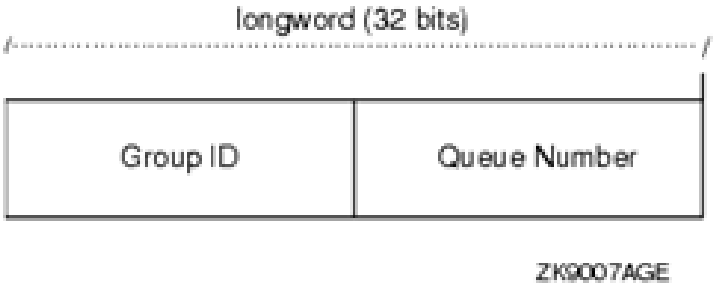
Table 2-33 Message Atribute Argument

Select Mode	Select Variable	Mode Description
PSEL_PQ_TYPE	Type	Selects the first pending message from the primary queue that matches the type value in the select variable word.
PSEL_PQ_CLASS	Class	Selects the first pending message from the primary queue that matches the class value in the select variable word.
PSEL_PQ_PRI	PSEL_PRI_ANY PSEL_PRI_P0 PSEL_PRI_P1 integer value between 0 and 99	Selects the first pending message with a priority equal to an integer between 0 and 99 inclusive (or equal to the select variable value) from within the primary queue. Specifying the direct integer value is the preferred method of selected messages by priority. Using PSEL_PRI_ANY enables the reading of any pending messages of all priorities. Setting PSEL_PRI_P0 enables the application to retrieve pending messages of priority 0 only. Setting PSEL_PRI_P1 enables the strict retrieval of pending

Selection by Message Source

Provides for the selection of pending messages from primary and secondary queues, by source group ID, queue number, or both. The format for selection by source follows:

Figure 2-7 Selection by Message Source



Some examples of possible **sel_filter** arguments and their actions are as follows:

Table 2-34 sel_filter Argument

sel_filter Argument	Action
Zero or not specified	No filtering of any messages. All messages can be retrieved.
Source **q_address**	Only those messages that have a matching **q_address** are retrieved.
Selection mask created with `pams_set_select`	Only messages that exactly match the specified selection mask are retrieved.

Compound Selection

Allows the application to formulate complex rules for the order in which the message queues are searched. The `pams_set_select` function allows the application to create custom selection masks that can be used in the low-order word of the **sel_filter** argument. The format for compound selection follows.

Figure 2-8 Compound Selection

psb

Receives a PAMS Status Block containing 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. The structure of the PSB is as follows:

Table 2-35 psb Structure

Low Byte	High Byte	Contents	Description
0	1	Type	PSB type
2	3	Call Dependent	Currently not used.
4	7	PSB Delivery Status	The completion status of the function. For recoverable messages, this field contains PAMS__CONFIRMREQor PAMS__POSSDUPL. For nonrecoverable messages, it may also contain a value of PAMS__SUCCESS.
8	15	Message Sequence Number	A unique number assigned to a message when it is sent and follows the message to the destination PSB. This number is input to pams_confirm_msgto release a recoverable message.

Table 2-35 psb Structure

16	19	PSB UMA Status	This field is not used with the pams_get_msga function.
20	23	Function Return Status	This field is not used with the pams_get_msga function.
24	31	Not Used	Not used.

Note: This function utilizes the AST services of OpenVMS; therefore, the application must check the status information returned in the PSB.

show_buffer

Receives additional information which Oracle MessageQ extracts from the message header. The structure of the **show_buffer** argument is as follows:

Table 2-36 show_buffer Argument

Longword	Contents	Description
0	Version	The version of the **show_buffer** structure. Valid values are as follows: 10 = Version 1.0 20 = Version 2.0 50 = Version 5.0
1	Transfer Status	The status code associated with the transfer of **show_buffer** information into the application's buffer. Valid symbols are as follows: PAMS__SUCCESS-All available information has been transferred. PAMS__BUFFEROVF-Information was lost due to receiver buffer overflow. 0-No message returned. There is no information to transfer.
2	Transfer Size	The number of bytes transferred to the application buffer.

Table 2-36 show_buffer Argument

3	Flags	A bit array showing the status of fields in the show_buffer. A set bit indicates a valid field, while a cleared bit indicates indeterminable data or the end of the allocated show_buffer memory. The symbols for the flags field are as follows: PSYM_SHOW_VERSION PSYM_SHOW_STATUS PSYM_SHOW_SIZE PSYM_SHOW_FLAGS PSYM_SHOW_TARGET PSYM_SHOW_ORIGINAL_TARGET PSYM_SHOW_SOURCE PSYM_SHOW_ORIGINAL_SOURCE PSYM_SHOW_DELIVERY PSYM_SHOW_PRIORITY PSYM_SHOW_ENDIAN PSYM_SHOW_CORRELATION_ID
4	Not Used	Fills out the Control Section to its maximum 40 bytes.
5	Not Used	Fills out the Control Section to its maximum 40 bytes.
6	Not Used	Fills out the Control Section to its maximum 40 bytes.
7	Not Used	Fills out the Control Section to its maximum 40 bytes.
8	Not Used	Fills out the Control Section to its maximum 40 bytes.
9	Not Used	Fills out the Control Section to its maximum 40 bytes.
10	Target	The **q_address** of the latest message target.
11	Original Target	The **q_address** of the original message target.
12	Source	The **q_address** of the latest message source.

Table 2-36 show_buffer Argument

13	Original Source	The **q_address** of the original message.
14	Delivery Mode	The delivery mode that was used to queue the message. This is not necessarily the delivery mode used to generate the message.
15	Priority	The priority used to queue the message.
16	Endian	The byte ordering or encoding schemes of 2- and 4-byte integers. The possible settings are as follows: PSYM_UNKNOWN PSYM_VAX_BYTE_ORDER or PSYM_LITTLE_ENDIAN PSYM_NETWORK_BYTE_ORDER or PSYM_BIG_ENDIAN PSYM_FML
17	Correlation ID	The 32 byte correlation ID associated with the message.

show_buff_len

Supplies the length in bytes of the buffer defined in the **show_buffer** argument. The minimum length is 40 bytes. If the buffer is too small to contain all of the information, then the return code PAMS__BUFFEROVF will be in the **show_buffer** transfer status.

large_area_len

Specifies the size of the message area to receive messages larger than 32K. Also specifies the length of the message buffer when using double buffers (as indicated by **PSYM_MSG_BUFFER_POINTER**).

large_size

Returns the actual size of the large message, double buffer message, or FML32 message written to the message buffer. When using double buffer pointers with pams_get_msga, the new buffer size is returned in large_size. (This differs from pams_get_msg[w}, where the new buffer size is returned in large_area_len.)

actrtn

Supplies the address of an int32 value that is the entry point to an action routine. This action routine is executed when the pams_get_msga function completes.

actparm

Supplies an int32 value that is passed to the action routine specified in the **actrtn** argument when it is invoked.

flag_id

Supplies the `int32` value for the flag number to be set when the `pams_get_msga` function completes. When the `pams_get_msga` function executes, it clears this flag. If this argument value is not supplied, no flag is used.

nullarg_3

Reserved for Oracle MessageQ internal use as a placeholder argument. This argument must be supplied as a null pointer.

Description

Because the `pams_get_msga` function executes asynchronously, it obtains several argument values only after the message arrives. These argument values are the message buffer, source, class, type of the message, and a PAMS Status Block (PSB) status code containing the delivery status, UMA status, and the sequence number of the message. These values are not set until the message arrival triggers the AST routine or sets the event flag.

The `pams_get_msga` function specifies an AST parameter which is passed by value to the AST routine when the parameter is called. This parameter is used to provide a context for the information contained in the message and can be used to identify the specific processing required for the message. Following are some suggestions and rules for programming with ASTs:

- Create a context area, separate from mainline, for each AST that is simultaneously posted. An address or index associated with the context area should be used as the AST parameter to ensure the appropriate context is associated with the data that is delivered by the `pams_get_msga` function.
- Ensure that the addresses of any fields that are filled in asynchronously are valid throughout the period that the AST is posted. A common error in using ASTs is to post an AST request that fills in fields on the stack and becomes invalid as soon as the caller returns.
- Data may be passed between AST routines and mainline by the following mechanisms:
  - Oracle MessageQ messages.
  - An event queue managed by interlocked queuing instructions.

  - Shared data fields between mainline and the AST routines such that access to the data is clear. The use of a context area for each AST request can accomplish this.

- Access to complex data structures shared between mainline and AST routines should be serialized by placing the access inside an AST safe critical section. One way to do this is with the `$SETAST` system service.

Return Values

Table 2-37 Return Code

Return Code	Platform	Description
PAMS__BADARGLIST	OpenVMS	Wrong number of call arguments have been passed to this function.
PAMS__BADPARAM	OpenVMS	Bad argument value.
PAMS__BADPRIORITY	OpenVMS	Invalid priority value used for receive.
PAMS__BADSELIDX	OpenVMS	Invalid or undefined selective receive index.
PAMS__BADHANDLE	OpenVMS	Invalid message handle.
PAMS__MSGTOSMALL	OpenVMS	The **msg_area_len** argument must be positive or zero.
PAMS__NOACCESS	OpenVMS	No access to the queue.
PAMS__NOACL	OpenVMS	No access to resource. The ACL check failed.
PAMS__NOMEMORY	OpenVMS	Insuffucient memory resources to reallocate buffer pointer.
PAMS__NOTDCL	OpenVMS	The application has not been attached to Oracle MessageQ.
PAMS__NOTSUPPORTED	OpenVMS	Feature not supported or available.
PAMS__RESRCFAIL	OpenVMS	Failed to allocate a resource.
PAMS__STALE	OpenVMS	Resource is no longer valid and must be freed by the user.

Table 2-37 Return Code

PAMS__STOPPED	OpenVMS	The requested queue has been stopped.
PAMS SUCCESS	OpenVMS	Indicates successful completion

PSB Delivery Status

Table 2-38 PSB Delivery Status

PSB Delivery Status	Platform	Description
PAMS__CONFIRMREQ	OpenVMS	Confirmation required for this message.
PAMS__POSSDUPL	OpenVMS	Message is a possible duplicate.
PAMS__SUCCESS	OpenVMS	Indicates successful completion.

See Also

- pams_cancel_get
- pams_get_msg
- pams_get_msgw
- pams_put_msg
- pams_set_select

pams_get_msgw

Retrieves the next available message from a specified queue and moves it to the location specified in the **msg_area**

argument. This function waits until a message arrives in the queue or a user-specified timeout period has elapsed.

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 user-supplied **msg_area** argument. Priority ranges from 0 (lowest priority) to 99 (highest priority). If the priority is set to 0, the pams_get_msqw function gets messages of any priority. If the priority is set to any

value from 1 to 99, the pams_get_msqw function gets only messages of that priority. 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 message arrives, or no available message meets the selection criteria before the **timeout** period expires, then the return status is PAMS_TIMEOUT.

If a queue has been sent a recoverable message, the receiver program can confirm receipt of the message using the pams_confirm_msg function. The pams_confirm_msg function enables the successfully delivered message to be deleted from the message recovery system. See the Using Recoverable Messaging topic for a description of the Oracle MessageQ recovery system.

See the Sending and Receiving Oracle MessageQ Messages topic for more information on working with FML32 buffers and large messages.

Syntax

```
int32 pams_get_msgw ( msg_area, priority, source, class, type,
msg_area_len, len_data, timeout, [sel_filter], [psb], [show_buffer],
[show_buffer_len], [large_area_len], [large_size],[nullarg_3] )
```

Argument

Table 2-39 Argument

Argument	Data Type	Mechanism	Prototype	Access
msg_area	char	reference	char *	returned
priority	char	reference	char *	passed
source	q_address	reference	q_address *	returned
class	short	reference	short *	returned
type	short	reference	short *	returned
msg_area_len	short	reference	short *	passed
len_data	short	reference	short *	returned

Table 2-39 Argument

timeout	int32	reference	int32 *	passed
[sel_filter]	int32	reference	int32 *	passed
[psb]	struct psb	reference	struct psb *	returned
[show_buffer]	struct show_buffer	reference	struct show_buffer *	returned
[show_buffer_len]	int32	reference	int32 *	passed
[large_area_len]	int32	reference	int32 *	passed/ returned
[large_size]	int32	reference	int32 *	returned
[nullarg_3]	char	reference	char *	passed

Argument Definitions

msg_area

For buffer-style messaging, receives the address of a memory region where Oracle MessageQ writes the contents of the retrieved message. For FML-style messaging or when using double ponters, receives a pointer to the address of the message being retrieved.

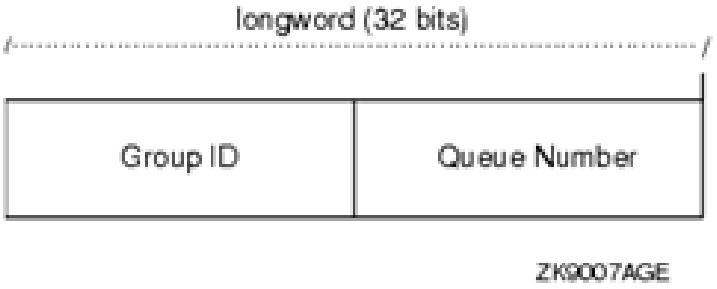
priority

Supplies the priority level for selective message reception. Priority ranges from 0 (lowest priority) to 99 (highest priority). If the priority is set to 0, the `pams__get_msqw` function gets messages of any priority. If the priority is set to any value from 1 to 99, the `pams__get_msqw` function gets only messages of that priority.

source

Receives a structure identifying the group ID and queue number of the sender program's primary queue in the following format:

Figure 2-9 group ID and queue number



class

Receives the class code of the retrieved message. The class is specified in the arguments of the `pams_put_msg` function. Oracle MessageQ 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 `p_typecl.h` include file. On UNIX and Windows NT systems, the `p_typecl.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 Oracle MessageQ are as follows:

Table 2-40 Class symbols

Reserved Class	Symbol Value
MSG_CLAS_MRS	28
MSG_CLAS_PAMS	29
MSG_CLAS_ETHERNET	100
MSG_CLAS_UCB	102
MSG_CLAS_TUXEDO	31001

Table 2-40 Class symbols

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

type

Receives the type code of the retrieved message. The type is specified in the arguments of the pams_put_msg function. Oracle MessageQ supports the use of symbolic names for **type** argument values. Symbolic type names begin with MSG_TYPE_. For specific information on defining type symbols, see the p_typecl.h include file.

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

msg_area_len

- Supplies the size of the buffer (in bytes) for buffer-style messages of up to 32767 bytes. The **msg_area** buffer is used to store the retrieved message.
- For messages using double buffers, including FML32 buffers, this argument contains the symbol PSYM_MSG_BUFFER_PTR to indicate that the message is a pointer to the address of the message being retrieved. The **msg_area** buffer contains the message pointer. The size of the message is returned in the **large_size** argument. The **msg_area** buffer is used to store the retrieved message. The **large_area_len** argument is used to supply the size of the message buffer to receive the message. If the retrieved buffer is larger than the space allocated, space is dynamically reallocated and the new buffer size is stored in **large_area_len**.
- For large messages (buffer-style messages larger than 32767 bytes), this argument contains the symbol PSYM_MSG_LARGE to indicate that the message buffer is greater than 32K. The size of the message is returned in the **large_size** argument. The **msg_area** buffer is used to store the retrieved message. The **large_area_len** argument is used to supply the size of the message buffer to receive the large message.

len_data

For static buffer-style messaging with messages of up to 32767 bytes, this argument receives the number of bytes retrieved from the message queue and stored in the area specified by the **msg_area** argument. This field also receives the

PSYM_MSG_BUFFER_PTR symbol for double buffer and FML-style messages and PSYM_MSG_LARGE for buffer-style messages larger than 32767 bytes.

timeout

Supplies the maximum amount of time the pams_get_msg function waits for a message to arrive before returning control to the application. The timeout value is entered in tenths (0.1) of a second. A value of 100 indicates a timeout of 10 seconds. If the timeout occurs before a message arrives, the status code of PAMS__TIMEOUT is -returned.

If an unlimited timeout period is required, set this argument to 0. On UNIX and Windows NT systems, a value of zero for this argument causes this function to block indefinitely or until it receives a message. On OpenVMS systems, this function waits for approximately 5 days or until it receives a message.

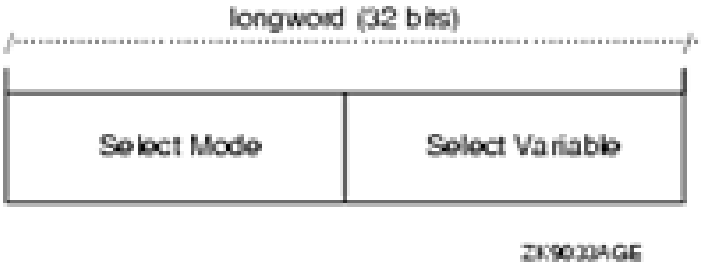
sel_filter

Supplies the criteria for the application to selectively retrieve messages. The argument contains one of the following selection criteria:

- Default selection
- Selection by message queue
- Message attributes
- Message source
- Compound selection using the pams_set_select function

The sel_filter argument is composed of two words as follows:

Figure 2-10 sel_filter argument



Default Selection

Enables applications to read messages from the queue based on the order in which they arrived. The default selection, PSEL_DEFAULT, reads the next pending message from the message queue. Messages are stored by priority and then in FIFO order. To specify this explicitly, both words in the **sel_filter** argument should be set to 0.

Selection by Message Queue

Allows the application to retrieve messages based upon a queue type or combination of queue types. This selection criteria is used to retrieve the first pending message that matches the criteria on the first queue it encounters. FIFO ordering is maintained within each queue. The predefined constants for this argument are as follows:

Table 2-41 Selection by Message Queue

Select Mode	Select Variable	Mode Description
PSEL_PQ	0	Enables the application to read from the primary queue (PQ) only. The select variable must equal 0.
PSEL_AQ	Alternate queue number	Enables an application to read from an alternate queue (AQ) only. The queue type can be a secondary queue (SQ).
PSEL_PQ_AQ	Alternate queue number	Attempts to selectively retrieve from a primary queue and then from an alternate queue.
PSEL_AQ_PQ	Alternate queue number	Attempts to selectively retrieve from an alternate queue and then from a primary queue.
PSEL_TQ_PQ	Alternate queue number	Attempts to selectively retrieve messages from a timer queue (TQ), and then from a primary queue.

Table 2-41 Selection by Message Queue

PSEL_TQ_PQ_AQ	Alternate queue number	Attempts to selectively retrieve messages from a timer queue (TQ), then from a primary queue, and finally from an alternate queue.
PSEL_UCB	0	Retrieves messages only from the user callback queues (UCB).

Selection by Message Attribute

Enables the application to select messages based on the message type, class, or priority. The predefined constants for this argument are as follows:

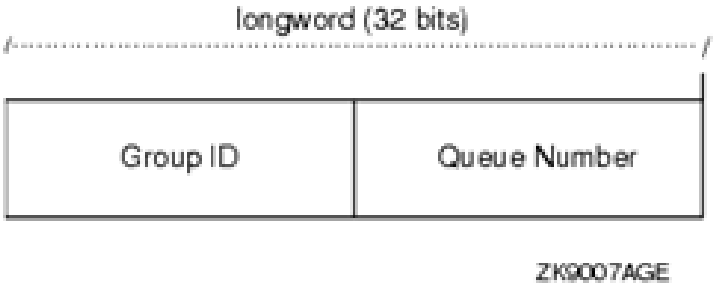
Table 2-42 Selection by Message Attribute

Select Mode	Select Variable	Mode Description
PSEL_PQ_TYPE	Type	Selects the first pending message from the primary queue that matches the type value in the select variable word.
PSEL_PQ_CLASS	Class	Selects the first pending message from the primary queue that matches the class value in the select variable word.
PSEL_PQ_PRI	PSEL_PRI_ANY PSEL_PRI_P0 PSEL_PRI_P1 integer value between 0 and 99	Selects the first pending message with a priority equal to an integer between 0 and 99 inclusive (or equal to the select variable value) from within the primary queue. Specifying the direct integer value is the preferred method of selected messages by priority. Using PSEL_PRI_ANY enables the reading of any pending messages of all priorities. Setting PSEL_PRI_P0 enables the application to retrieve pending messages of priority 0 only. Setting PSEL_PRI_P1 enables the strict retrieval of pending messages with a priority of 1.

Selection by Message Source

Provides for the selection of pending messages from primary and secondary queues, by source group ID, queue number, or both. The format for selection by source -follows:

Figure 2-11 Selection by Message Source



Some examples of possible **sel_filter** arguments and their actions are as follows:

Table 2-43 sel_filter arguments

sel_filter Argument	Action
Zero or not specified	No filtering of any messages. All messages can be -retrieved.
Source **q_address**	Only those messages that have a matching **q_address** are retrieved.
Selection mask created with `pams_set_select`	Only messages that exactly match the specified selection mask are retrieved.

Compound Selection

Allows the application to formulate complex rules for the order in which the message queues are searched. The `pams_set_select` function allows the application to create custom selection masks that can be used in the low-order word of the **sel_filter** argument. The format for compound selection follows:

Figure 2-12 Compound Selection

psb

Receives a PAMS Status Block containing 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. The structure of the PSB is as follows:

Table 2-44 PSB Structure

Low Byte	High Byte	Contents	Description
0	1	Type	PSB type
2	3	Call Dependent	Currently not used.
4	7	PSB Delivery Status	The completion status of the function. It contains the status from MRS. It can also contain a value of PAMS__SUCCESS when the message is not sent recoverably.
8	15	Message Sequence Number	A unique number assigned to a message when it is sent and follows the message to the destination PSB. This number is input to the pams_confirm_msg function to release a recoverable message.
16	19	PSB UMA Status	The completion status of the undeliverable message action (UMA). The PSB UMA status indicates if the UMA was not executed or applicable.

Table 2-44 PSB Structure

20	23	Function Return Status	After a Oracle MessageQ function completes execution, Oracle MessageQ software writes the return value to this field.
24	31	Not Used	Not used.

show_buffer

Receives additional information which Oracle MessageQ extracts from the message header. The structure of the **show_buffer** argument is as follows:

Table 2-45 show_buffer argument

Longword	Contents	Description
0	Version	The version of the show_buffer structure. Valid values are as follows: 10 = Version 1.0 20 = Version 2.0
1	Transfer Status	The status code associated with the transfer of show_buffer information into the application's buffer. Valid symbols are as follows: PAMS__SUCCESS-All available information has been transferred. PAMS__BUFFEROVF-Information was lost due to receiver buffer overflow. 0-No message returned. There is no information to transfer.
2	Transfer Size	The number of bytes transferred to the application buffer.

Table 2-45 show_buffer argument

Longword	Contents	Description

3	Flags	A bit array showing the status of fields in the show_buffer. A set bit indicates a valid field, while a cleared bit indicates indeterminable data or the end of the allocated show_buffer memory. The symbols for the flags field are as follows:
		`PSYM_SHOW_VERSION PSYM_SHOW_STATUS`
		`PSYM_SHOW_SIZE PSYM_SHOW_FLAGS`
		`PSYM_SHOW_TARGET`
		`PSYM_SHOW_ORIGINAL_TARGET`
		`PSYM_SHOW_SOURCE`
		`PSYM_SHOW_ORIGINAL_SOURCE`
		`PSYM_SHOW_DELIVERY PSYM_SHOW_PRIORITY`
		`PSYM_SHOW_ENDIAN`
		`PSYM_SHOW_CORRELATION_ID`

4	Not Used	Fills out the Control Section to its maximum 24 bytes.
5	Not Used	Fills out the Control Section to its maximum 24 bytes.
6	Not Used	Fills out the Control Section to its maximum 24 bytes.
7	Not Used	Fills out the Control Section to its maximum 24 bytes.
8	Not Used	Fills out the Control Section to its maximum 24 bytes.
9	Not Used	Fills out the Control Section to its maximum 24 bytes.
10	Target	The q_address of the latest message target.
11	Original Target	The q_address of the original message target.
12	Source	The q_address of the latest message source.
13	Original Source	The q_address of the original message.
14	Delivery Mode	The delivery mode that was used to queue the message.
15	Priority	The priority used to queue the message.

Table 2-45 show_buffer argument

Longword	Contents	Description
16	Endian	PSYM_VAX_BYTE_ORDER or PSYM_LITTLE_ENDIAN PSYM_NETWORK_BYTE_ORDER or PSYM_BIG_ENDIAN PSYM_FML
17	Correlation ID	The 32 byte correlation ID associated with the message.

show_buff_len

Supplies the length in bytes of the buffer defined in the **show_buffer** argument. The minimum length is 40 bytes. If the buffer is too small to contain all of the information, the return code PAMS__BUFFEROVF will be in the **show_buffer** transfer status.

large_area_len

Specifies the size of the message area to receive messages larger than 32K. Also specifies the length of the message buffer when using double buffers (as indicated by PSYM_MSG_BUFFER_POINTER). This argument also stores the length of double buffers and FML32 buffers after reallocation.

large_size

Returns the actual size of the large message, double buffer message, or FML message written to the message buffer.

nullarg_3

Reserved for Oracle MessageQ internal use as a placeholder argument. This argument must be supplied as a null pointer.

Return Codes

Table 2-46 Return Codes

Return Code	Platform	Description
PAMS__AREATOSMALL	All	Received message is larger than the application message area.
PAMS__BADARGLIST	All	Wrong number of call arguments have been passed to this function.
PAMS__BADHANDLE	All	Invalid message handle.

Table 2-46 Return Codes

PAMS__BADPARAM	All	Bad argument value.
PAMS__BADPRIORITY	All	Invalid priority value used for receive.
PAMS__BADSELIDX	All	Invalid or undefined selective receive index.
PAMS__BADTIME	OpenVMS	An invalid time was specified.
PAMS__BUFFEROVF	UNIX Windows NT	The size specified for the show_buffer argument is too small.
PAMS__EXHAUSTBLKS	OpenVMS	No more message blocks available.
PAMS__FMLERROR	All	Problem detected with internal format of FML message; this can be an error in processing or data corruption.
PAMS__INSQUEFAIL	All	Failed to properly queue a message buffer.
PAMS__MSGTOSMALL	All	The **msg_area_len** argument must be positive or zero.
PAMS__MSGUNDEL	All	Message returned is undeliverable.
PAMS__NEED_BUFFER_PTR	UNIX Windows NT	FML32 buffer received but **msg_area_len** argument not set to PSYM_MSG_BUFFER_PTR.
PAMS__NETERROR	Clients	Network error resulted in a communications link abort.
PAMS__NOACCESS	All	No access to resource. ACL check failed.
PAMS__NOACL	All	The queue access control file could not be found.
PAMS__NOMEMORY	OpenVMS	Insuffucient memory resources to reallocate buffer pointer.
PAMS__NOMRQRESRC	All	Insufficient multireader queue resources to allow access.

Table 2-46 Return Codes

PAMS__NOTDCL	All	Process has not been attached to Oracle MessageQ.
PAMS__NOTSUPPORTED	UNIX Windows NT	Specified delivery mode is not supported.
PAMS__PAMSDOWN	Windows NT	The specified Oracle MessageQ group is not running.
PAMS__PREVCALLBUSY	Clients	Previous call to CLS has not been completed.
PAMS__QUECORRUPT	OpenVMS	Message buffer queue corrupt.
PAMS__REMQUEFAIL	All	Failed to properly read a message buffer.
PAMS__STALE	All	Resource is no longer valid and must be freed by the user.
PAMS__STOPPED	All	The requested queue has been stopped.
PAMS__SUCCESS	All	Successful completion.
PAMS__TIMEOUT	All	Timeout period has expired.
PAMS__CONFIRMREQ	All	Confirmation required for this -message.
PAMS__PAMSDOWN	UNIX Windows NT	The specified Oracle MessageQ group is not running.
PAMS__POSSDUPL	All	Message is a possible duplicate.
PAMS__SUCCESS	All	Indicates successful completion.

See Also

- pams_get_msga
- pams_put_msg

- pams_set_select

Example

Block Until a Message Is Read

This example shows how to use the pams_get_msgw function. It sets an alarm to send messages to itself every 5 seconds; it uses pams_get_msgw to sit and wait for them. The queue named "queue_1" must be defined in your initialization file as a primary queue. The complete code example called x_getw.c is contained in the examples directory.

pams_locate_q

Locates the queue address for the specified queue name or queue alias. By default, this function waits for the queue address to be returned.

Syntax

```
int32 pams_locate_q ( q_name, q_name_len, q_address, [wait_mode],
[req_id], [resp_q], [name_space_list], [name_space_list_len], [timeout] )
```

Arguments

Table 2-47 Arguments

Argument	Data Type	Mechanism	Prototype	Access
q_name	char	reference	char *	passed
q_name_len	int32	reference	int32 *	passed
q_address	q_address	reference	q_address *	returned
[wait_mode]	int32	reference	int32 *	passed
[req_id]	int32	reference	int32 *	passed
[resp_q]	int32	reference	int32 *	passed
[name_space_list]	int32 array	reference	int32 array *	passed

Table 2-47 Arguments

[name_space_list_len]	int32	reference	int32 *	passed
[timeout]	int32	reference	int32 *	passed

Argument Definitions

q_name

Supplies the queue name or queue alias whose queue address is requested. The procedure that Oracle MessageQ uses to find this name is controlled by the **name_space_list** argument, described below.

q_name_len

Supplies the number of characters in the **q_name** argument. The maximum string length on UNIX, Windows NT, and OpenVMS systems is 255 characters. For all other Oracle MessageQ environments, the maximum string length is 31.

q_address

Receives the queue address assigned by Oracle MessageQ when an application has successfully located the queue name.

wait_mode

Supplies the search mode of the pams_locate_q 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:

PSYM_WF_RESP (default setting)-The application issues the pams_locate_q request and waits for the queue address to be returned.

PSYM_AK_RESP-The application issues the pams_locate_q address and continues processing. When the search is completed, the queue address is returned to the application's primary queue in a LOCATE_Q_REP message. The response message can be redirected to an alternate queue address using the **resp_q** argument.

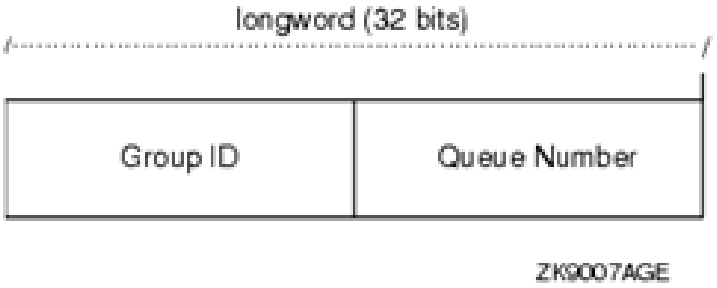
req_id

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

resp_q

Supplies an alternate queue to use for receiving the acknowledgment message of the **q_address**. If no response queue is specified, the acknowledgment message is sent to the sender program's primary queue. The **resp_q** argument has the following format:

Figure 2-13 resp_q argument



Note that the group ID field is always equal to zero because the sender program cannot specify a response queue outside its group.

name_space_list

If the **name_space_list** argument is specified, the **name_space_list_len** argument must also be specified. If this argument is unspecified, then `PSEL_TBL_GRP` is the default.

Possible values in a **name_space_list** argument are as follows:

Table 2-48 name_space_list argument

Location it represents	Symbolic value
Process cache	`PSEL_TBL_PROC`
Group/group cache	`PSEL_TBL_GRP`
Global name space	`PSEL_TBL_BUS (or PSEL_TBL_BUS_MEDIUM or PSEL_TBL_BUS_LOW)`

The **name_space_list** argument identifies the scope of the name as follows:

- To identify a local queue reference or a queue, an application must include PSEL_TBL_GRP in **name_space_list.** (Do not specify PSEL_TBL_BUS in the list because it would identify a global queue reference.)
- To identify a global queue reference, include PSEL_TBL_BUS (or PSEL_TBL_BUS_MEDIUM or PSEL_TBL_BUS_LOW) in the **name_space_list** argument and specify its pathname, either explicitly or implicitly. If the **q_name** argument contains any slashes (/), or periods (.), Oracle MessageQ treats it as a pathname. Otherwise, Oracle MessageQ treats **q_name** as a name and adds the DEFAULT_NAMESPACE_PATH to the name to create the pathname to lookup. (The DEFAULT_NAMESPACE_PATH is set in the %PROFILE section of the group initialization file.)

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

- To lookup a local queue reference or queue name, specify both PSEL_TBL_GRP and PSEL_TBL_PROC. This causes the process cache to be checked before looking into the group cache.
- To lookup a global queue reference, specify PSEL_TBL_BUS (or PSEL_TBL_BUS_LOW or PSEL_TBL_BUS_MEDIUM), PSEL_TBL_GRP and PSEL_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 PSEL_TBL_BUS_LOW instead of PSEL_TBL_BUS.

To lookup only the slower but more up-to-date caches in the global name space before looking in the masterdatabase, specify PSEL_TBL_BUS_MEDIUM instead of PSEL_TBL_BUS.

For more information on dynamic binding of queue addresses, see the Using Naming topic.

name_space_list_len

Supplies the number of entries in the **name_space_list** argument. If the **name_space_list_len** argument is zero, Oracle MessageQ uses PSEL_TBL_GRP as the default in the **name_space_list** argument.

timeout

Specifies the number of PAMS time units (1/10 second intervals) to allow for the locate to complete. If timeout is zero, the group's ATTACH_TMO property is used. If the ATTACH_TMO is also zero, 600 is used.

Return Values

Table 2-49 Return Codes

Return Code	Platform	Description
PAMS__BADARGLIST	OpenVMS	Wrong number of call arguments.
PAMS__BADNAME	UNIX Windows NT	The queue name contains illegal characters.
PAMS__BADPARAM	All	Invalid argument in the argument list.
PAMS__BADRESPQ	All	Invalid response queue specified.
PAMS__BOUND	All	Queue name in use.
PAMS__BUSNOTSET	UNIX Windows NT	DMQ_BUS_ID environment variable not set.
PAMS__GROUPNOTSET	UNIX Windows NT	DMQ_GROUP_ID environment variable not set.
PAMS__NETERROR	Clients	Network error resulted in a communications link abort.
PAMS__NOACCESS	All	The address of the specified name is either 0 or is in another group.
PAMS__NOOBJECT	All	Could not locate queue name.
PAMS__PAMSDOWN	All	The specified Oracle MessageQ group is not running.
PAMS__PREVCALLBUSY	Clients	Previous call to CLS has not been completed.
PAMS__RESRCFAIL	All	Failed to allocate resources.

Table 2-49 Return Codes

PAMS__SUCCESS	All	Successful completion of an action.
PAMS__TIMEOUT	All	The timeout period specified has expired.
PAMS__UNBINDING	All	Queue requested is in the process of unbinding from a PAMS_bind_q request.

See Also

- pams_attach_q
- pams_exit

Example

Locate a Queue Address

This example shows how to use the pams_locate_q function by attaching to queue_1 and locating queue_3 where a message is being sent. The queues named "queue_1" and "queue_3" must be defined in your initialization file; queue_1 must be a primary queue. The complete code example called x_locate.c is contained in the examples directory.

pams_open_jrn

Opens the selected message recovery journal. The Oracle MessageQ dead letter journal (DLJ) stores messages designated as recoverable that could not be delivered by the recovery system. The Oracle MessageQ postconfirmation journal (PCJ) stores recoverable messages that were successfully delivered. See the Using Recoverable Messaging topic for a description of Oracle MessageQ message recovery services.

Syntax

```
int32 pams_open_jrn ( jrn_filespec, jrn_filename_len, jrn_handle )
```

Arguments

Table 2-50 Arguments

Argument	Data Type	Mechanism	Prototype	Access
jrn_filespec	char	reference	char *	passed
jrn_filename_len	short	reference	short *	passed
jrn_handle	int32	reference	int32 *	returned

Argument Definitions

jrn_filespec

Supplies the file name of the message recovery journal from which the application will read stored messages.

Note: jrn_filespec should use the following format: <group name><queue name>.< SAF/DQF/ACK/DLQ/PCJ>. The group name length is four, and the queue name length is 8.

jrn_filename_len

Supplies the length of the file specification entered to the **jrn_filespec** argument specified (filename array) in number of bytes.

jrn_handle

Receives the journal handle for the selected message recovery file if this function completes successfully.

Note: jrn_filespec uses the following format: <group name><queue name>.< SAF/DQF/ACK/DLQ/PCJ>. The group name length is four, and the queue name length is 8.

A journal queue is locked by only one application that opens or reads the journal queue at one time. When the application attempts to open the journal queue, it tries to attach a temporary queue as a secondary queue at first. If there are not enough temporary queues left in the queue pool, the journal queue is not opened. The number of temporary queues can be defined when creating queue space; the default value is 200.

Return Values

Table 2-51 Return Code

Return Code	Platform	Description
PAMS__BADARGLIST	OpenVMS	Invalid number of call arguments.
PAMS__NOMEMORY	OpenVMS	Insufficient virtual memory.
PAMS__NOSUCHPCJ	OpenVMS	Error occured when attempting to open the specified journal queue.
PAMS__SUCCESS	OpenVMS	Indicates successful completion.

See Also

- pams_close_jrn
- pams_confirm_msg
- pams_put_msg
- pams_read_jrn

pams_put_msg

Sends a message to a target queue using a set of standard Oracle MessageQ delivery modes. Applications specify buffer-style or FML-style messaging using the **msg_size** argument. For buffer-style messaging using message buffers up to 32K, this argument supplies the length of the message in bytes in the user's **msg_area** buffer. In addition, you can use the **msg_size** argument to specify one of the following symbols:

- PSYM_MSG_FML-indicates FML-style messaging. The **msg_area** argument must contain a pointer to an FML32 buffer.
- PSYM_MSG_LARGE-indicates buffer-style message with messages up to 4MB in length. The pointer to the buffer is contained in the **msg_area** argument and the size of the large message buffer is contained in the **large_size** argument.

The **delivery** argument of the pams_put_msg function 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 if the message recovery cannot store the message. You must also supply the **psb** argument to receive the return status of the operation.

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 **resp_q** 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.

To use a pointer to an FML32 buffer when sending a message, the sender program specifies the symbol `PSYM_MSG_FML` as the **msg_size** argument to the `pams_put_msg` function.

Syntax

```
int32 pams_put_msg ( msg_area, priority, target, class, type, delivery,
msg_size, [timeout], [psb], [uma], [resp_q], [large_size],
[correlation_id],[nullarg_3] )
```

Arguments

Table 2-52 Arguments

Argument	Data Type	Mechanism	Prototype	Access
msg_area	char	reference	char *	passed
priority	char	reference	char *	passed
target	q_address	reference	q_address *	passed
class	short	reference	short *	passed
type	short	reference	short *	passed
delivery	char	reference	char *	passed
msg_size	short	reference	short *	passed
[timeout]	int32	reference	int32 *	passed

Table 2-52 Arguments

[psb]	struct psb	reference	struct psb *	returned
[uma]	char	reference	char *	passed
[resp_q]	q_address	reference	q_address *	passed
large_size	int32	reference	int32 *	passed
[correlation_id]	char	reference	char *	passed
[nullarg_3]	char	reference	char *	passed

Argument Definitions

msg_area

For buffer-style messaging, supplies the address of a memory region or a message pointer containing the message to be delivered to the target queue of the receiver program. For FML-style messaging, supplies the message pointer that points to an FML32 buffer containing the message.

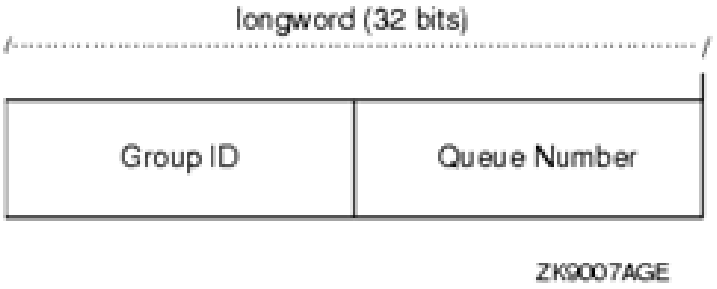
priority

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

target

Supplies the queue number and group ID of the receiver program's queue address in the following format:

Figure 2-14 target



class

Supplies the class code of message being sent. Oracle MessageQ 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 `p_typecl.h` include file. On UNIX and Windows NT systems, the `p_typecl.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 Oracle MessageQ are as follows:

Table 2-53 Class symbols

Reserved Class	Symbol Value
MSG_CLAS_MRS	28
MSG_CLAS_PAMS	29
MSG_CLAS_ETHERNET	100
MSG_CLAS_UCB	102
MSG_CLAS_TUXEDO	31001

Table 2-53 Class symbols

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

type

Supplies the type code for the message being sent. Oracle MessageQ supports the use of symbolic names for **type** argument values. Symbolic type names begin with MSG_TYPE_. For information on defining type symbols, see the p_typecl.h include file.

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

delivery

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

- PDEL_MODE_*sn_dip*-where *sn* is one of the following sender notification constants:
- WF-Wait for completion
- AK-Asynchronous acknowledgment
- NN-No notification

And *dip* is one of the following delivery interest point constants:

- ACK-Read from target queue and explicitly acknowledged using the pams_confirm_msg function. ACK can also be an implicit acknowledgement sent after the second pams_get_msg call by the receiving application.
- CONF-Delivered from the DQF and explicitly confirmed using the pams_confirm_msg 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.

msg_size

For buffer-style messaging using message buffers up to 32K, supplies the length of the message in bytes in the user's**msg_area** buffer. In addition, you can specify one of the following symbols:

- PSYM_MSG_FML--indicates FML-style messaging. The **msg_area** argument must contain a pointer to an FML32 buffer.
- PSYM_MSG_LARGE-indicates buffer-style messaging with messages up to 4MB in length. The pointer to the buffer is contained in the **msg_area** argument and the size of the large message buffer is contained in the **large_size** argument.

timeout

Supplies the maximum amount of time the pams_put_msg function waits for a message to arrive before returning control to the application. The timeout value is entered in tenths (0.1) of a second. A value of 100 indicates a timeout of 10 seconds. If the timeout occurs before a message arrives, the status code PAMS__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 PAMS 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.

The structure of the PSB is as follows:

Table 2-54 PSB Structure

Low Byte	High Byte	Contents	Description
1	0	Type	PSB type.
3	2	Call Dependent	Currently not used.

Table 2-54 PSB Structure

7	4	PSB Delivery Status	The completion status of the function. It contains the status from MRS. It can also contain a value of PAMS__SUCCESS when the message is not sent recoverably.
15	8	Message Sequence Number	A unique number assigned to the message when it is sent and follows the message to the destination PSB. This number is input to the pams_confirm_msg function to release a recoverable message.
19	16	PSB UMA Status	The completion status of the undeliverable message action (UMA). The PSB UMA status indicates if the UMA was not executed or applicable.
23	20	Function Return Status	After a Oracle MessageQ function completes execution, Oracle MessageQ software writes the return value to this field.
31	24	Not Used	Not used.

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 PDEL_UMA_*XXX* where *XXX* is one of the following symbols:

Table 2-55 UMA Symbols

Symbol	Description
DISC	Discard message
DISCL	Discard after logging message
DLJ	Dead letter journal
DLQ	Dead letter queue

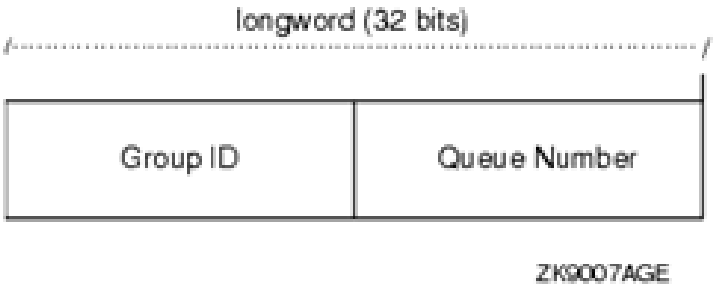
Table 2-55 UMA Symbols

RTS	Return to sender
SAF	Store and Forward

resp_q

Supplies a **q_address** 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 **resp_q** argument to receive the response messages. The **resp_q** argument has the following format:

Figure 2-15 resp_q argument



The group ID is always specified as zero because the sender program cannot assign a response queue outside its group.

large_size

Supplies the actual size of the large message written to the message buffer.

correlation_id

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

nullarg_3

Reserved for Oracle MessageQ internal use as a placeholder argument. This argument must be supplied as a null pointer.

Return Values

Table 2-56 Return Codes

Return Code	Platform	Description
PAMS__BADARGLIST	All	Wrong number of call arguments have been passed to this function.
PAMS__BADDELIVERY	All	Invalid delivery mode.
PAMS__BADHANDLE	All	Invalid message handle.
PAMS__BADPARAM	UNIX Windows NT OpenVMS	Attempt to use cross-group connection when cross-group communication is disabled. On OpenVMS systems, invalid NULL call argument.
PAMS__BADPRIORITY	All	Invalid priority value on send operation.
PAMS__BADPROCNUM	UNIX Windows NT	Invalid target queue address specified.
PAMS__BADRESPQ	All	Response queue not owned by process.
PAMS__BADTIME	OpenVMS	Invalid time specified.
PAMS__BADUMA	All	Undeliverable message action (UMA) is invalid.
PAMS__EXCEEDQUOTA	All	Target process quota exceeded; message was not sent.
PAMS__EXHAUSTBLKS	OpenVMS	No more message blocks available.
PAMS__FMLERROR	All	Problem detected with internal format of FML message; this can be an error in processing or data corruption.
PAMS__LINK_UP	OpenVMS	MRS has reestablished link.

Table 2-56 Return Codes

PAMS__MSGTOBIG	All	Message exceeded the size of the largest link list section (LLS).
PAMS__MSGTOSMALL	OpenVMS	Invalid (negative) msg_size specified in the argument list.
PAMS__NETERROR	Clients	Network error resulted in a communications link abort.
PAMS__NOMRS	OpenVMS	MRS is not available.
PAMS__NOTACTIVE	All	Target process is not currently active; message not sent.
PAMS__NOTDCL	All	Process has not been attached to Oracle MessageQ.
PAMS__NOTFLD	All	The buffer supplied is not an FML32 buffer.
PAMS__NOTSUPPORTED	All	The combination of delivery mode and uma selected is not supported.
PAMS__PNUMNOEXIST	OpenVMS	Target queue number does not exist.
PAMS__PREVCALLBUSY	Clients	Previous call to CLS has not been completed.
PAMS__REMQUEFAIL	All	Failed to properly dequeue a message buffer.
PAMS__STOPPED	All	The requested queue has been stopped.
PAMS__SUCCESS	All	Successful completion.
PAMS__TIMEOUT	All	Timeout period has expired.
PAMS__UNATTACHEDQ	All	Message successfully sent to unattached queue.
PAMS__WAKEFAIL	OpenVMS	Failed to wake up the target process.

Table 2-57 UMA Status

UMA Status	Platform	Description
PAMS__DISC_FAILED	All	Message not recoverable in destination queue file (DQF); undeliverable message action (UMA) was PDEL_UMA_DISC; message could not be discarded.
PAMS__DISC_SUCCESS	All	Message not recoverable in DQF; UMA was PDEL_UMA_DISC; message - discarded.
PAMS__DISCL_FAILED	All	Message not recoverable in DQF; UMA was PDEL_UMA_DISC; recoverability failure could not be logged or message could not be discarded.
PAMS__DISCL_SUCCESS	All	Message not recoverable in DQF; UMA was PDEL_UMA_DISC; message discarded after logging recoverability -failure.
PAMS__DLJ_FAILED	All	Message not recoverable in DQF; UMA was PDEL_UMA_DLJ; dead letter journal (DLJ) write operation failed.
PAMS__DLJ_SUCCESS	All	Message not recoverable in DQF; UMA was PDEL_UMA_DLJ; message written to the DLJ.
PAMS__DLQ_FAILED	All	Message not recoverable in DQF; UMA was PDEL_UMA_DLQ; message could not be queued to the DLQ.
PAMS__DLQ_SUCCESS	All	Message not recoverable in DQF; UMA was PDEL_UMA_DLQ; message queued to the DLQ.
PAMS__NO_UMA	All	Message is recoverable; UMA not -executed.
PAMS__RTS_FAILED	All	Message not recoverable in DQF; UMA was PDEL_UMA_RTS; message could not be returned to sender.
PAMS__RTS_SUCCESS	All	Message not recoverable in DQF; UMA was PDEL_UMA_RTS; message returned
PAMS__SAF_FAILED	All	Message not recoverable in DQF; UMA was PDEL_UMA_SAF; store and forward (SAF) write operation failed.

Table 2-57 UMA Status

PAMS__SAF_SUCCESS	All	Message not recoverable in DQF; UMA was PDEL_UMA_SAF; message recoverable from SAF file.
PAMS__UMA_NA	All	UMA not applicable.

See Also

- pams_get_msg
- pams_get_msga
- pams_get_msgw

Example

Send a Message

This example sends a number of messages to a queue. The complete code example called x_putslf.c is contained in the examples directory.

pams_read_jrn

Reads a message from a Oracle MessageQ journal file. Use the pams_open_jrn function to open the dead letter journal or postconfirmation journal for a message queuing group. Use the pams_close_jrn function to close the journal file after reading selected messages. Note that on UNIX and Windows NT systems, these functions are performed by running the Journal Replay utility.

The receiver program determines whether each message is a FML buffer or a large message by reading the **len_data** argument. See the Sending and Receiving Oracle MessageQ Messages topic for more information on working with message handles and large messages.

Syntax

```
int32 pams_read_jrn ( jrn_handle, msg_area, priority, source, class, type,
msg_area_len, len_data, target, write_time, conf_val, msg_seq_num,
mrs_status, [large_area_len], [large_size], [nullarg_3] )
```

Arguments

Table 2-58 Arguments

Argument	Data Type	Mechanism	Prototype	Access
jrn_handle	int32	reference	int32 *	passed
msg_area	char	reference	char *	returned
priority	char	reference	char *	returned
source	q_address	reference	q_address *	returned
class	short	reference	short *	returned
type	short	reference	short *	returned
msg_area_len	short	reference	short *	returned
len_data	short	reference	short *	returned
target	q_address	reference	q_address *	returned
write_time	unsigned int32	reference	unsigned int32 *	returned
conf_val	int32	reference	int32 *	returned
msg_seq_num	unsigned int32	reference	unsigned int32 *	returned
mrs_status	int32	reference	int32 *	returned
[large_area_len]	int32	reference	int32 *	returned
[large_size]	int32	reference	int32 *	returned
[nullarg_3]	char	reference	char *	returned

Argument Definitions

jrn_handle

Supplies the journal handle of the message recovery journal from which the application has selected to read journal entries. The journal handle is returned to the application by the `pams_open_jrn` function.

msg_area

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

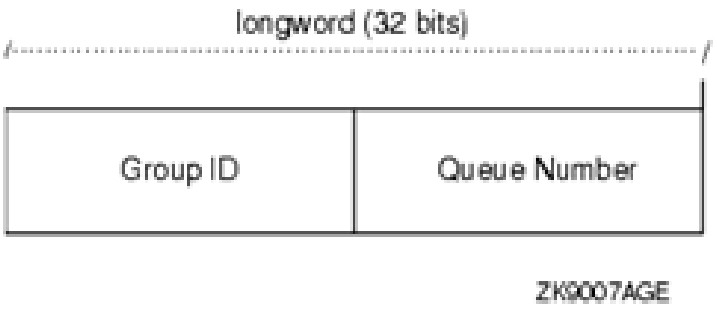
priority

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

source

Receives a structure containing the queue number and group ID of the sender program's primary queue in the following format:

Figure 2-16 queue number and group ID



class

Receives the class code of the retrieved message. The class is specified in the arguments of the `pams_put_msg` function. Oracle MessageQ 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 `p_typecl.h` include file. Class symbols reserved by Oracle MessageQ are as follows:

Table 2-59 Class symbols

Reserved Class	Symbol Value
MSG_CLAS_MRS	28
MSG_CLAS_PAMS	29
MSG_CLAS_ETHERNET	100
MSG_CLAS_UCB	102
MSG_CLAS_TUXEDO	31001
MSG_CLAS_TUXEDO_TPSUCCESS	31002
MSG_CLAS_TUXEDO_TPFAIL	31003
MSG_CLAS_XXX	30000 through 32767 (except 31001-31003)

type

Receives the type code of the journaled message. The type is specified in the arguments of the pams_put_msg function. Oracle MessageQ supports the use of symbolic names for **type** argument values. Symbolic type names begin with MSG_TYPE_. For information on defining type symbols, see the p_typecl.h include file. The OpenVMS symbol values range from -1 through -5000. Use of the **type** argument facilitates selective message reception. However, if the receiving application does not need a specific value for its processing, then use a value of 0.

msg_area_len

Supplies the size of the buffer (in bytes) for buffer-style messages of up to 32K bytes. The **msg_area** buffer is used to store the retrieved message.

len_data

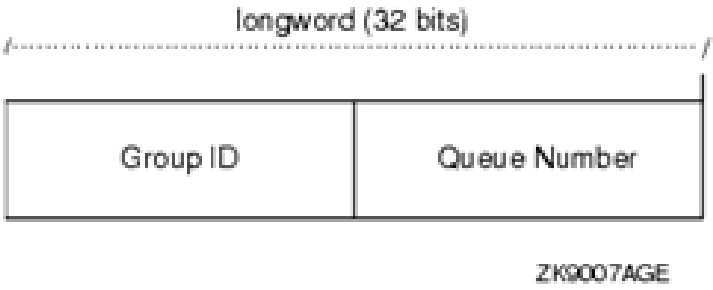
- For buffer-style messaging with messages of up to 32K, this argument receives the number of bytes retrieved from the message queue and stored in the area specified by the **msg_area** argument.

- For an FML-style message, this argument contains the symbol PSYM_MSG_BUFFER_PTR to indicate that the message is a pointer to an FML32 buffer.
- For large messages, this argument contains the symbol PSYM_MSG_LARGE to indicate that the message buffer is greater than 32K. The size of the message is returned in the **large_size** argument.

target

Receives the queue number and group ID of the receiver's queue address in the following format:

Figure 2-17 target



write_time

Receives the address of the quadword (an array of two int32 values) specifying the date and time that the recoverable message was confirmed. This parameter uses standard OpenVMS system time.

conf_val

Receives the message confirmation value.

msg_seq_num

Receives the message sequence number generated by Oracle MessageQ in the PSB of the received message. This argument should be set to the values in the PSB.

mrs_status

Receives the Message Recovery Services (MRS) status of the message.

large_area_len

Specifies the size of the message buffer to receive messages larger than 32K.

large_size
Returns the actual size of the large message written to the message buffer.

nullarg_3
Reserved for Oracle MessageQ internal use as a placeholder argument. This argument must be supplied as a null pointer.

Return Values

Table 2-60 Return Codes

Return Code	Platform	Description
PAMS__AREASTOSMALL	OpenVMS	Received message is larger than the user message area.
PAMS__BADARGLIST	OpenVMS	Invalid number of arguments supplied.
PAMS__BADHANDLE	OpenVMS	Invalid message handle.
PAMS__INVJH	OpenVMS	Invalid journal handle.
PAMS__MSGTOBIG	OpenVMS	Message in journal file is larger than GROUP_MAX_MESSAGE_SIZE.
PAMS__NOMEMORY	OpenVMS	Insufficient virtual memory.
PAMS__NOMOREMSG	OpenVMS	No more messages in journal.
PAMS__SUCCESS	OpenVMS	Indicates successful completion.
PAMS NOSUCHPCJ	ALL	All Error opening specified journal.
PAMS NOACCESS	ALL	All Invalid rmid parameter
PAMS NOTACTIVE	ALL	All Cannot access the qspace because it is not available.
PAMS TIMEOUT	ALL	This error code indicates that either a timeout has occurred.
PAMS STALE	ALL	An invalid or deleted queue name was specified.

See Also

- pams_close_jrn
- pams_open_jrn

pams_set_select

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 pams_set_select function creates an index handle that is used as the **sel_filter** argument of Oracle MessageQ functions for reading the message. When a selection index handle is passed to pams_get_msg, pams_get_msga or pams_get_msgw, 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.

Syntax

```
int32 pams_set_select ( selection_array, num_masks, index_handle )
```

Arguments

Table 2-61 Arguments

Argument	Data Type	Mechanism	Prototype	Access
selection_array	selection_array_	reference	selection_array_	passed
num_masks	short	reference	short *	passed
index_handle	int32	reference	int32 *	returned

Argument Definitions

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 `p_entry.h` as follows:

```
typedef struct _selection_array_component {
        int32 queue;
        int32 priority;
        int32 key_1_offset;
        int32 key_1_size;
        int32 key_1_value;
        int32 key_1_oper;
        int32 key_2_offset;
        int32 key 2 size;
        int32 key_2_value;
        int32 key_2_oper;
        int32 order_offset;
        int32 order_size;
        int32 order_order; union {
                pams__correlation_idcorrelation_id;
                pams__sequence_number sequence number
    } extended_key
} selection_array_component;
```

The `selection_array_component` data structure has the following components:

Table 2-62 selection_array_component data structure

Component	Description
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.

The following tables define the content of each of the components of the `selection_array_component` data structure.

Queue and Priority

The following table specifies the valid values that can be applied to the arguments in this part of the `Select_Queue` structure:

Table 2-63 Select_Queue structure

Field	Values	Description
Queue	Queue Number	Specifies the queue number to be searched. The queue number can be any message queue for which the application has read access. The queue number can be obtained from the **q_attached** argument of the pams_attach_q function or **q_address** of the pams_locate_q function. A value of 0 for this argument specifies the application's primary queue.
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.

Table 2-63 Select_Queue structure

	PSEL_PRI_ANY	Read priority 1 before reading priority 0 messages.
	PSEL_PRI_P0	Read priority 0 messages only.
	PSEL_PRI_P1	Read priority 1 messages only.

Comparison Keys

The following table specifies the arguments and valid values that can be applied to this part of the Selection_Array_Components structure:

Table 2-64 Selection_Array_Components structure

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).
	PSEL_SOURCE	Source address of message.
	PSEL_CLASS	Class of the message.
	PSEL_TYPE	Type of the message.
	PSEL_CORRELATION_ID	Correlation ID of the message. May be used for key_1_offset or key_2_offset but not both. If this symbol is specified, the Size field must be set to PSEL_CORRELATION_ID_SIZE (or 32 bytes).
	PSEL_SEQUENCE_NUMBER	Message sequence number acquired from the PAMS Status Buffer. If this symbol is specified, the Size field must be set to PSEL_SEQUENCE_NUMBER_SIZE (or 8 bytes).
Size		Specifies data type of the key to be compared.
	0	Disable use of key.

Table 2-64 Selection_Array_Components structure

	1	Byte (8 bits).
	2	Word (16 bits).
	4	int32 (32 bits).
	PSEL_SEQUENCE_NUMBER_SIZE	8 bytes
	PSEL_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.
	PSEL_OPER_EQ	Message field = value.
	PSEL_OPER_NEQ	Message field <> value.
	PSEL_OPER_GTR	Message field > value.
	PSEL_OPER_LT	Message field < value.
	PSEL_OPER_GTRE	Message field > or = value.
	PSEL_OPER_LTE	Message field < or = value.

Order Key

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

Table 2-65 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).
	PSEL_SOURCE	Source address of the message.
	PSEL_CLASS	Class of the message.
	PSEL_TYPE	Type of the message.
	PSEL_CORRELATION_ID	Correlation ID of the message. If this symbol is specified, the Size field must be set to PSEL_CORRELATION_ID_SIZE (or 32 bytes).
	PSEL_SEQUENCE_NUMBER	Message sequence number acquired from the PAMS Status Buffer. If this symbol is specified, the Size field must be set to PSEL_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).
	PSEL_SEQUENCE_NUMBER_SIZE	8 bytes
	PSEL_CORRELATION_ID_SIZE	32 bytes

Table 2-65 Order Key

Order		Order operator. The *order* variable specifies the sequence in which the select process is to be performed.
	PSEL_ORDER_FIFO	First pending.
	PSEL_ORDER_MIN	Minimum value of all pending.
	PSEL_ORDER_MAX	Maximum value of all pending.

Correlation ID

The correlation ID is a 32-byte user-defined identifier associated with a message. If PSEL_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, PSEL_CORRELATION_ID should only be specified for one of the comparison keys; specifying the correlation ID for both keys results in a PAMS__BADPARAM error.

If PSEL_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 PAMS Status Buffer (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 sequnce 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 PSEL_BY_MASK.
- The low-order word contains the index to the selection array.

The **index_handle** is passed as the **sel_filter** argument in pams_get_msg, pams_get_msga or pams_get_msgw, and pams_cancel_select functions. OpenVMS allows a maximum number of 500 index handles. Other Oracle MessageQ implementations offer 16K to 32K index handles.

Return Values

Table 2-66 Return Codes

Return Code	Platform	Description
PAMS__BADARGLIST	OpenVMS	Invalid number of arguments supplied.
PAMS__BADPARAM	All	Bad argument passed in the function call.
PAMS__IDXTBLFULL	All	Selective receive index table is full.
PAMS__NETERROR	Clients	Network error resulted in a communications link abort.
PAMS__NOTDCL	All	Process has not been attached to Oracle MessageQ.
PAMS__PAMSDOWN	UNIX Windows NT	The specified Oracle MessageQ group is not running.
PAMS__PREVCALLBUSY	Clients	Previous call to CLS has not been completed.
PAMS__SUCCESS	All	Indicates successful completion.

See Also

- pams_cancel_get
- pams_cancel_select
- pams_get_msg
- pams_get_msga
- pams_get_msgw

Example

Selecting Messages Using a Complex Selection Filter

This example shows the selective reception of messages using `pams_set_select` to build a complex message selection filter. The queue named "`queue_1`" must be defined in your initialization file as a primary queue. The complete code example called `x_select.c` is contained in the examples directory.

pams_set_timer

Creates a timer that sends a message to an application's primary queue when a time interval expires or a time of day arrives. The message is sent as a priority 1 message with a source of `PAMS__TIMER_QUEUE`, a class code of PAMS, and a type code of `TIMER_EXPIRED`. A **timer_id** is returned by this function as the first int32 value in the `TIMER_EXPIRED` message.

Note: Prior to Oracle MessageQ Version 5.0, the valid priority values were 0 and 1. In Version 5.0, the valid range is 0 to 99 (0 being the lowest priority and 99 the highest priority). Keep in mind that timer priorities are always 1 and take this into account when modifying existing programs to take advantage of the expanded priority range. Messages associated with timers have a priority of 1 and are not sent until all messages with priorities from 2 to 99 are read.

To act upon the timer message, the application uses the `pams_get_msgw` function to read its primary queue, block until the timer expiration message arrives, and then act upon it. To cancel a Oracle MessageQ timer, use the `pams_cancel_timer` function with the identification code of the timer you want to cancel.

Syntax

```
int32 pams_set_timer ( timer_id, timer_format, p_timeout,
s_timeout )
```

Arguments

Table 2-67 Arguments

Argument	Data Type	Mechanism	Prototype	Access
`timer_id`	`int32`	reference	`int32 *`	passed

Table 2-67 Arguments

timer_format	char	reference	char *	passed
p_timeout	int32	reference	int32 *	passed
s_timeout	unsigned quadword	reference	unsigned quadword *	passed

Argument Definitions

timer_id

Supplies a unique timer identification value created by the application. Must be greater than zero.

timer_format

Supplies the time format being used. Following are the two predefined constants for this argument:

P-selects the time interval in PAMS timer format supplied to the **p_timeout** argument. PAMS timer format expresses time in units of one tenth of a second. Using the PAMS timer format provides an operating system independent way to represent a time interval.

S-selects the system-dependent time format supplied to the **s_timeout** argument. Using a system-dependent time format limits the portability of applications to a specific operating system environment.

p_timeout

Supplies the amount of time to delay (delta) from the current time before returning a timer expiration message. If the **timer_format** argument is set to P, a value greater than 0 must be entered for this argument. This argument uses the PAMS timer format which expresses time in units of one tenth of one second.

s_timeout

On OpenVMS systems, use this argument to supply a pointer to an array of two int32 values used to set a 64-bit OpenVMS time format. The **s_timeout** argument can be specified as an absolute time or a delta time matching the OpenVMS time format rules. Note that if the caller exceeds the ASTLM or TQELM process quota, the process can enter the RWAST state.

On UNIX and Windows NT systems, use this argument to supply a two element array of int32 values. The values represent an absolute time (a UTC time in seconds and

microseconds) at which the timer will expire. To use the **s_timeout** argument, developers provide a pointer to a "`struct timeval`" as follows:

```
struct timeval theTime;

nStatus = pams_set_timer(&timer_id, "S", NULL, (int32 *) &theTime);
```

Return Values

Table 2-68 Return Codes

Return Code	Platform	Description
PAMS__BADARGLIST	OpenVMS	Invalid number of arguments supplied.
PAMS__BADPARAM	All	Bad argument value.
PAMS__INVALIDNUM	All	Invalid timer number passed to PAMS_set_timer.
PAMS__INVFORMAT	All	Invalid timer format specified in the call. Should be P or S.
PAMS__NETERROR	Clients	Network error resulted in a communications link abort.
PAMS__NOTDCL	All	Process has not been attached to Oracle MessageQ.
PAMS__NOTSUPPORTED	UNIX Windows NT	The S timer_format is not supported by Oracle MessageQ for UNIX and Windows NT systems.
PAMS__PAMSDOWN	UNIX Windows NT	The specified Oracle MessageQ group is not running.
PAMS__PREVCALLBUSY	Clients	Previous call to CLS has not been completed.
PAMS__RESRCFAIL	All	Insufficient resources to complete operation.
PAMS__SUCCESS	All	Indicates successful completion.

See Also

- pams_cancel_timer

Example

Set a Timer

This example shows how to use the Oracle MessageQ timer functions by setting a timer to go off every 5 seconds. When the timer expires, it sends messages to itself. While not handling the timer event, it sits and waits for other incoming messages. If it is interrupted, it cancels any outstanding timers. The queue named "queue_1" must be defined in your initialization file as a primary queue. The complete code example called x_timer.c is contained in the examples directory.

pams_status_text

Receives the severity level and text description of a user-supplied PAMS API return code and moves that information to a user-supplied storage area. If the error code is not known, an error is returned and the call parameters are not filled in.

Syntax

```
int32 pams_status_text ( code, severity, buffer, buflen, retlen)
```

Arguments

Table 2-69 Arguments

Argument	Data Type	Mechanism	Prototype	Access
code	int32	reference	int32 *	passed
severity	int32	reference	int32 *	returned
buffer	char	reference	char *	returned
buflen	int32	reference	int32 *	passed
retlen	int32	reference	int32 *	returned

Argument Definitions

code

Specifies the return value for which you would like the text description and severity level returned.

severity

Receives a code indicating the severity level of the message. Severity levels apply to both success and errormessages. They are designed to provide more information about the message being returned. The valid codes returned to this argument are as follows:

0 = warning

1 = success

2 = error

3 = informational

4 = fatal error

buffer

Receives the text description for the return status supplied.

buflen

Specifies the length of the buffer to store the text description returned. A buffer length of 256 bytes is adequate to store the text description for all return status codes. If the user buffer supplied is large enough, the string is zero terminated. The buffer length must be entered as a positive integer. Supplying a negative integer value to this argument causes the function to return a status of `PAMS__BADPARAM`. If you specify this argument as zero, no text is returned to the buffer and the function returns the status of `PAMS__TRUNCATED`.

retlen

Receives the size of the user-supplied buffer space that was filled by the text description returned.

Description

Application developers use the `pams_status_text` function to obtain a text description and severity level for each API return value. The text description contains both the symbolic name (as it is defined in the include files and described in the documentation) followed by a comma, a space, and then a description of the return value in the following format:

```
PAMS__SUCCESS, normal successful completion
```

In addition to the text description, this function returns a code indicating the severity level for both success and error messages.

For example, pams_detach_q has two possible success return codes; PAMS-- SUCCESS and PAMS__DETACHED. The PAMS SUCCESS return code is used to indicate that you successfully detached the specified queue(s).

PAMS__DETACHED is an informational return code indicating that the call was successful and that you have detached your last queue which effectively detaches your application from the message queuing bus in the same manner as the pams_exit function.

Return Values

Table 2-70 Return Codes

Return Code	Platform	Description
PAMS__BADARGLIST	OpenVMS	Invalid number of call parameters specified.
PAMS__BADPARAM	All	Invalid call parameter specified.
PAMS__FAILED	All	There is no translation for the specified return code.
PAMS__SUCCESS	All	Normal successful completion.
PAMS__TRUNCATED	All	The description was returned but was -truncated.

putil_show_pending

Requests the number of pending messages for a list of selected queues. To use the putil_show_pending function, specify the number of message queues for which you want to obtain a pending message count and the list of queue addresses for which you want to obtain a pending message count. The value returned by this function contains the total number of messages in each memory queue. On OpenVMS systems, this function also returns the number of pending messages in the local recovery journals targeted for delivery to the selected queue.

Syntax

```
int32 putil_show_pending ( count, in_q_list, out_pend_list )
```

Arguments

Table 2-71 Arguments

Argument	Data Type	Mechanism	Prototype	Access
count	int32	reference	int32 *	passed
in_q_list	short array	reference	short array*	passed
out_pend_list	int32 array	reference	int32 array *	returned

Argument Definitions

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 of `int32` values containing the queue numbers for which the pending message count is requested.

out_pend_list

Return Values

Table 2-72 Return Code

Return Code	Platform	Description
`PAMS__BADARGLIST`	UNIX Windows NT	Invalid argument supplied to this function.
`PAMS__BADPARAM`	OpenVMS	Invalid argument supplied to this function.
`PAMS__NETERROR`	Clients	Network error resulted in a communications link abort.
`PAMS__NOTDCL`	All	Process is not attached to Oracle MessageQ.

Table 2-72 Return Code

PAMS__RESRCFAIL	All	Insufficient resources to complete operation.
PAMS__PAMSDOWN	All	The specified Oracle MessageQ group is not running.
PAMS__PREVCALLBUSY	Clients	Previous call to CLS has not been completed.
PAMS__SUCCESS	All	Successful completion.

Example

Display Number of Pending Messages

This example shows how to use `putil_show_pending` to display the number of pending messages currently in the queue. A queue named "`queue_1`" must be defined during group configuration. The complete code example called `x_shopnd.c` is contained in the examples directory.

Using Message-Based Services

- Receiving a Response
- Obtaining the Status of a Queue
- Monitoring and Controlling Link Status
- Learning the Current Status of Queues
- Managing Message Recovery Files

Oracle MessageQ applications regularly perform standard tasks such as checking the state of a queue or the status of a cross-group connection before sending a message. To make these tasks easier, Oracle MessageQ offers message- based services, which are sets of predefined request, notification, and response messages exchanged between the application and Oracle MessageQ server processes.

Table 2-73 describes the functions performed by using message-based services and lists the servers they are available through.

Table 2-73 Overview of Message-Based Services

You can . . .
Obtain the status of a particular queue
Monitor and control link status
Obtain the current status of all queues
Register for broadcast messages
Manage message recovery files (OpenVMS systems only)
Transfer messages from one DQF file to another (OpenVMS systems only)

Oracle MessageQ uses message-based services to perform routine tasks such as obtaining queue status. There are two request-response paradigms used by message-based services. For some kinds of services, the sender program sends a request to a Oracle MessageQ server using a particular message. The Oracle MessageQ server returns the response in a message using a particular message type and format. If information was requested, it is returned in the message area of the response message.

In other cases, a sender program may register to receive ongoing updates of information. In this case, the sender program sends a registration request and receives a response if the registration request is successful. In addition, the sender program receives event-driven messages providing up-to-date information as requested. To stop receiving the event-driven messages, the sender program must send a deregistration request to the Oracle MessageQ server.

Service requests are directed to the primary queue of the Oracle MessageQ server designated to provide the selected service. Oracle MessageQ message-based service requests are delivered to Oracle MessageQ servers using the Oracle MessageQ application programming interface (API) or Oracle MessageQ scripts. Similarly, applications obtain response and notification messages by reading these messages from their primary or response queue.

Oracle MessageQ message-based services are sent between a user application program that functions as a requestor and a Oracle MessageQ server process that fulfills the request. For messages to be properly understood between systems, message data must be sent and returned in the endian format understood by both the requestor and the server.

Most Oracle MessageQ message-based services automatically perform this conversion if the endian format of the two systems is different. However, some message-based services do not

perform this conversion. Therefore, the user application must convert the message to the endian format of the server system to ensure that the message data is correctly interpreted.

See the description of each message for information on whether Oracle MessageQ performs the conversion or the application must check for differences in hardware data formats. See the Building and Testing Applications topic to learn how you can ensure that your application formats data properly and performs required conversions when sending standard messages between computer systems from different vendors.

You can send a service request message using the `pams_put_msg` function. Request messages use the **type** argument to identify the purpose of the message. Each request message has a predefined data structure.

To send a standard request message, supply the following:

A detailed description of each message in the Message Reference topic explains each field in the data structure and provides a sample C message structure.

Receiving a Response

Each Oracle MessageQ server returns response or notification messages to answer a service request. Most request messages have a response message. In addition, some service requests are answered by the Oracle MessageQ server with a notification message that supplies information to the sender program as it becomes available.

When an application requests information using the `pams_put_msg` function, it provides the Oracle MessageQ server with the group ID and queue number to which the response should be directed. The sender program then reads this queue using the `pams_get_msg`, `pams_get_msgw`, or pams_get_msga function to obtain the response information.

A Oracle MessageQ server response and notification message provides the following:

Table 2-74 A Oracle MessageQ server response

Source	The symbolic name of the Oracle MessageQ server fulfilling the request.
Class	The class code of the response is always PAMS, indicating that this is a Oracle MessageQ message- based service.

Table 2-74 A Oracle MessageQ server response

Type	The type code of the message received. For example, AVAIL_REG_REPLY.
Message data	The predefined data structure used to provide requested information in the response or notification message. The definition of all Oracle MessageQ message-based services messages is now provided in the p_msg.h include file.

A detailed description of each message in the Message Reference topic explains each field in the data structure and provides a sample C message structure.

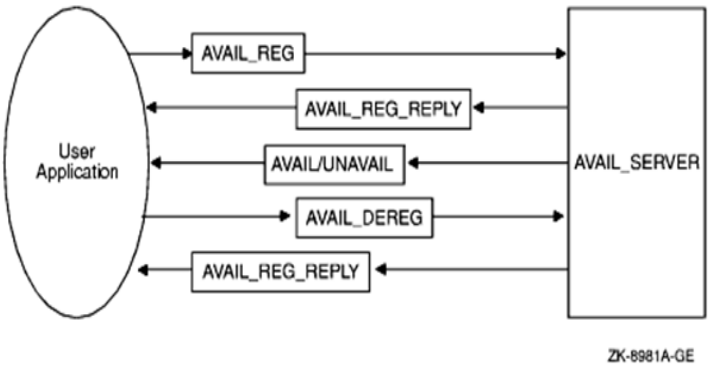
Obtaining the Status of a Queue

Oracle MessageQ message-based services enable applications to check whether a particular queue is available to receive messages. This set of messages returns information on the status of any active queue in a local or remote group.

To obtain information on the status of a particular queue, applications exchange the following messages with the Avail Server:

- AVAIL_REG-Request message to register to receive queue information.
- AVAIL_REG_REPLY-Response message to confirm registration or deregistration.
- AVAIL-Notification message to indicate that the queue is available.
- UNAVAIL-Notification message to indicate that the queue is unavailable.
- AVAIL_DEREG-Notification message to deregister from obtaining queue information.

Figure 2-18 Figure 5-1 Avail Server Message Flow



An application program registers to receive availability messages by sending a message of type AVAIL_REG to the local Avail Server process. The Avail Server responds with a message of type AVAIL_REG_REPLY, acknowledging the notification request.

After registration, the requestor immediately receives an AVAIL or UNAVAIL message indicating the current availability of the target queue. Queue availability messages provide ongoing notification when a specific queue becomes attached or detached and when a link is connected or lost. If the queue becomes active because a process becomes attached, the Avail Server sends a message of type AVAIL. If it becomes -inactive, the server sends a message of type UNAVAIL.

Applications must cancel availability notification by sending a message of type AVAIL_DEREG. The application receives a AVAIL_REG_REPLY message indicating the status of the operation. It is important to note that if the distribution queue for an AVAIL registration becomes unavailable, the registration will be automatically deleted by Oracle MessageQ. A subsequent attempt to deregister AVAIL services for this distribution queue will result in an error message indicating that the registration does not exist.

Monitoring and Controlling Link Status

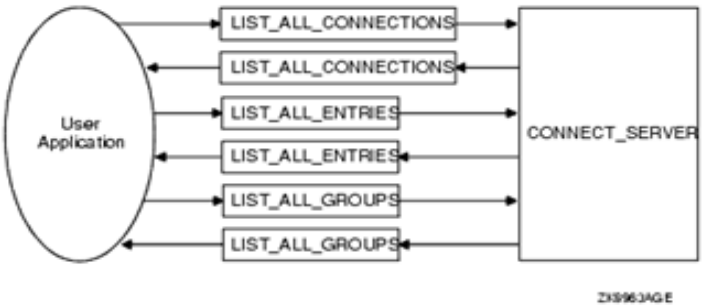
This section describes how applications can use Oracle MessageQ message-based services with the Connect Server process to obtain information on connections, queue entries, groups, cross-group connections, and link status.

Listing Cross-Group Connections, Entries, and Groups

An application can request a list of current cross-group connections or all configured cross-group entries from the Connect Server. This request allows the application to obtain the current Oracle MessageQ cross-group configuration and active cross-group connections. In addition, the Connect Server can provide a list of known queues in a group and a list of all groups defined on a message queuing bus.

- To obtain a list of all cross-group connections, configured groups, and queue entries, applications exchange the following messages with the Connect Server:
- `LIST_ALL_CONNECTIONS` (Request)-Request message to provide a list of all cross-group connections.
- `LIST_ALL_CONNECTIONS` (Response)-Response message to provide a list of all cross-group connections. Groups with no link connection are not listed.
- `LIST_ALL_ENTRIES` (Request)-Request message to provide a list of all queue entries for a group.
- `LIST_ALL_ENTRIES` (Response)-Response message to provide a list of all queue entries for a group.
- `LIST_ALL_GROUPS` (Request)-Request message to provide a list of groups on the message queuing bus.
- `LIST_ALL_GROUPS` (Response)-Response message to provide a list of all groups, connected and unconnected, on the message queuing bus.

Figure 2-19 Figure 5-2 Requesting Cross-Group Information



To obtain a list of all groups defined on the message queuing bus, send a LIST_ALL_GROUPS message to the Connect Server. To obtain a list of all cross-group connections for the message bus or a list of all cross-group entries, send a LIST_ALL_CONNECTIONS message to the Connect Server. To obtain a list of queues in a group, send a LIST_ALL_ENTRIES message.

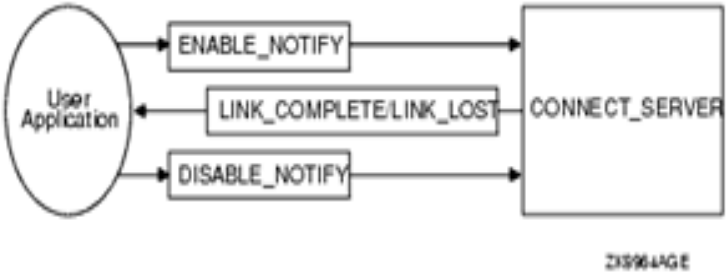
The reply to these requests is a variable-length message with the same type and class as the request. To read the information returned, the application uses the message size parameter returned by the pams_get_msg function and divides it by the byte size of the data object requested to determine the number of data entries returned. The byte size of these entries is described in the reference description of each message.

Obtain Notification of Cross-Group Links Established and Lost

An application can also use Connect Server messages to receive notification of cross-group links connected and disconnected in its own group. To obtain information on the status of cross-group links, use the following message- based services:

- ENABLE_NOTIFY-Request message to request notification of link changes.
- LINK_COMPLETE-Notification message to indicate that the cross-group link was created.
- LINK_LOST-Notification message to indicate that the cross-group link was lost.
- DISABLE_NOTIFY-Request message to request disabling of link change -notification.

Figure 2-20 Figure 5-3 Requesting Cross-Group Link Status



Applications send an ENABLE_NOTIFY message to the Connect Server to receive ongoing notification when new connections are made or lost. Registered applications receive a LINK_COMPLETE notification message when a new cross-group connection is created.

Applications receive a LINK_LOST message when a cross-group connection is lost. To deregister from -receiving further notification messages, the application sends a DISABLE_NOTIFY message to the Connect Server.

Note: **Note:** To receive ongoing notification of queue attachments, we recommend the use of the Queue Server messages,such as ENABLE_Q_NOTIFY_REQ. The ENABLE_NOTIFY message should no longer be used to obtain queue attachment information.

Controlling Cross-Group Links

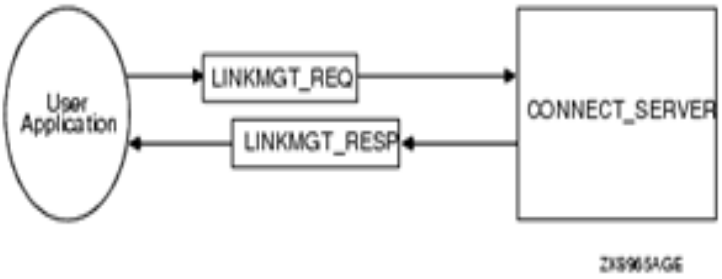
In addition to obtaining information on cross-group links, the Connect Server messages can be used to control cross- group connections through a feature called link management. Applications use link management messages to explicitly control the creation and deletion of cross-group links. Explicit control over remote links may be required by an application to restrict network communication with a particular node or to reduce network traffic.

The LINKMGT_REQ request message enables the following control functions:

- Inquire-Allows querying of a group's link state.
- Enable-Re-enables a link's address entries.
- Disable-Disables a link's address entries.
- Connect-Re-enables a link's address entries and connects to selected groups.
- Disconnect-Implicitly disables links and disconnects links to requested groups.

The LINKMGT_RESP response message notifies the requesting application if the request was successful and supplies information about the cross-group connection. Link management functions are also available through the System Manager utility on Oracle MessageQ for OpenVMS systems. Figure 5-4 is a graphical representation of the functional relationship facilitated by LINKMGT_REQ and LINKMGT_RESP:

Figure 2-21 Figure 5-4 Using Link Management



Link management can also be event driven. For example, an application event can trigger a link to another group, which enables message exchange.

Note: When using link management, automatic creation of cross-group connections must be disabled with the generate connect option `D` (disable) in the `%XGROUP` section of the Oracle MessageQ group initialization file to completely control all cross-group links. For more information, refer to the Enabling Network Connections in the Cross-Group Section topic in the Oracle MessageQ Installation and Configuration Guide for each platform.

Link Management Control Functions

The link management request message allows for the following control functions:

- Inquire-Allows querying of a group's link state.
- Enable-Re-enables a link's address entries.
- Disable-Disables a link's address entries.
- Connect-Re-enables a link's address entries and connects to selected groups.
- Disconnect-Implicitly disables links and disconnects links to requested groups.

Inquire Function

The Inquire function of the link management request message allows querying of a single group's link state. To use the Inquire function, specify the group number of the local or remote group for which you want to learn the link state. This function does not allow you to specify any selection parameters other than the group number. Because you can only inquire about the link state of one

group at a time, you cannot specify the PSYM_LINKMGT_ALL_GROUPS symbol in the group_number field.

The Inquire function performs endian translation when the request is sent to a Connect Server running on a system that uses a different byte order. Both the request and response messages are encoded in the endian of the request originator.

Request Message Format for the Inquire Function

Table 2-75 displays the Inquire function request message format:

Table 2-75 Inquire Function Request Message Format

Field	Required/ Optional	Setting
version	Required	10
user_tag	Required	User-specified code identifying the request.
function_code	Required	PSYM_LINKMGT_CMD_INQUIRY
group_number	Required	Group number to receive the action. Valid values are 1 to 32000.
connect_type	Optional	PSYM_LINKMGT_ALL_TRANSPORTS
reconnect_timer	Optional	PSYM_LINKMGT_USE_PREVIOUS
window_size	Optional	PSYM_LINKMGT_USE_PREVIOUS
window_delay	Optional	PSYM_LINKMGT_USE_PREVIOUS
transport_addr_len	Optional	0
node_name_len	Optional	0

Determining the Status of the Inquire Request

The status field of the LINKMGT_RESP message contains a return code indicating the outcome of the inquiry request. Refer to Table 5-3 for a description of each status return and the corresponding user action.

Table 2-76 Inquire function status returns and user actions

PSYM_LINKMGT Return Code	Description	Outcome	Description/User Action
MSGCONTENT	Invalid value in request message	Error	One of the field values in the inquiry request message is invalid. Check the syntax of the request message against the list of valid values and re-issue the corrected request message.
MSGFMT	Unknown request version or function code	Error	Correct the syntax of the request message. The version field of the must contain the number 10. The function code field must contain the symbol PSYM_LINKMGT_CMD_INQUIRY.
NOGROUP	The selected group does not have a cross group entry	Error	You requested the link state for a group that is not defined in the cross-group table. This group has no cross-group links.
OPERATIONFAIL	The command was unable to be successfully completed	Error	The inquire function failed due to a system resource problem. • Check the network connection to the target group to determine if the network link is up. • Check the Connect Server to determine if it is running out of virtual memory. • Check the log file to see if the cause of the error has been logged.
SUCCESS	The operation successfully completed	Success	Refer to the description of the link management response message below for a description of the data returned.

Response Message Format for Successful Inquire Requests

If the Inquire function is successful, the response message returns the status of both the incoming and outgoing cross-group links in the in_link_state and out_link_state fields. These fields specify the status of the link using the following symbols:

- PSYM_LINKMGT_CONNECTED-the incoming/outgoing cross-group link for the selected group is connected.
- PSYM_LINKMGT_NOCN-the incoming/outgoing cross-group link for the selected group is not connected.
- PSYM_LINKMGT_DISABLE-the incoming/outgoing cross-group link for the selected group is disabled.

If the link status for the group is PSYM_LINKMGT_CONNECTED, the response message contains the following information:

Table 2-77 Response Message Information

Field	Description
version	10
user_tag	User-specified code from the request message.
Status	PSYM_LINKMGT_SUCCESS
group_number	Group number that receives the action.
in_link_state	PSYM_LINKMGT_CONNECTED
out_link_state	PSYM_LINKMGT_CONNECTED
connect_type	Transport that message is connected over: PSYM_LINKMGT_TCPIP.
platform_id	Connected platform ID (PSYM_PLATFORM_*xxxx*).
reconnect_time	Reconnect timer value for this group.
window_size	Window size value negotiated for this group.
window_delay	Window delay value negotiated for this group.
transport_addr_len	Length of the transport_addr string.
transport_addr	ASCII representation of the TCP/IP port number.

Table 2-77 Response Message Information

Field	Description
node_name_len	Length of the node_name string.
node_name	Name of the node this link is connected to.

Enable Function

The Enable function of the link management request message re-enables a link's address entries if they have been disabled. All addresses in the cross-group connection table that match the selection criteria specified in the request message (for example, group number, connect type, node name, and transport address) will be enabled. All other address entries for the group or groups selected will be disabled. The Enable function will still complete if the link is already connected. The effects will not be visible until the existing link is lost.

The Enable function allows a link to occur only with the selected addresses for a group. If the group has a reconnection timer, the timer will be set to cause the connection to be attempted after the specified time and connections are not attempted immediately. Incoming connections are then allowed to occur.

The Enable function offers the following selection options:

- If the group_number field is set to PSYM_LINKMGT_ALL_GROUPS, then the node name and transport address cannot be specified.
- If a specific group number is specified and PSYM_LINKMGT_ALL_TRANSPORTS is specified, then the node name and transport address cannot be specified.
- On OpenVMS systems, if an entry that matches the selection criteria is not found, one will be created providing the group exists. On UNIX and Windows NT systems, the Enable function only enables existing address entries. It does not modify connection parameters or add new address entries.
- On OpenVMS systems, if the window or reconnect timer information is supplied, the specified values overwrite the existing information of the select entries. On UNIX and Windows NT systems, the Enable function does not modify connection parameters.

Note: The symbol PSYM_LINKMGT_ALL_TRANSPORTS is new to the LINK_MGT message API for Oracle MessageQ Version 4.0. On OpenVMS systems, the Enable function requires that the requesting process have either OPER or the DMQ$OPERATOR rights identifier.

Request Message Format for the Enable Function

Table 2-78 displays the Enable function message format:

Table 2-78 Enable function message format

Field	Required/ Optional	Setting
version	Required	10
user_tag	Required	User-specified code identifying the request.
function_code	Required	PSYM_LINKMGT_CMD_ENABLE
group_number	Required	Group number to receive the action. Valid values are 1 to 32000. Or, use the PSYM_LINKMGT_ALL_GROUPS symbol to enable all known links for groups with the connect_type requested.
connect_type	Required	Select the following transport type: PSYM_LINKMGT_TCPIP
reconnect_timer	Optional	Time it takes for the COM Server or Group Control Process (GCP) to reconnect to a communications link. Enter the number of seconds or the following values: PSYM_LINKMGT_NO_TIMER PSYM_LINKMGT_USE_PREVIOUS
window_size	Optional	Size of transmission window (cross-group protocol Version 3.0 or higher).
window_delay	Optional	Transmission window delay in seconds (cross-group protocol Version 3.0 or higher).
transport_addr	Optional	Transport address string 16 bytes in length; the TCP/IP port ID
transport_addr_len	Optional	Length of transport address. Valid values are 0 to 16 bytes. Zero specifies the use of the previous setting.

Table 2-78 Enable function message format

Field	Required/ Optional	Setting
node_name	Optional	ASCII text of node name. The length is determined by node_name_len up to 255 characters.
node_name_len	Optional	Length of the node name string. Zero specifies the use of the previous known value.

Determining the Status of the Enable Request

The status field of the LINKMGT_RESP message contains a return code indicating the outcome of the Enable request. See Table 2-79 for a description of each status return and the corresponding user action.

Table 2-79 Enable function status returns and user actions

PSYM_LINKMGT Return Code	Description	Outcome	Description/User Action
ALREADYUP	The link is already active	Warning	The Enable function completed although the link entries were already available.
MSGCONTENT	Invalid value in request message	Error	One of the field values in the enable request message is invalid. Check the syntax of the request message against the list of valid values and re-issue the corrected request message.
MSGFMT	Unknown request version or function code	Error	Correct the syntax of the request message. The version field of the must contain the number 10. The function code field must contain the symbol PSYM_LINKMGT_CMD_ENABLE.
NOGROUP	The selected group does not have a cross group entry	Error	No cross-group entries can be enabled because you requested the enable function for a group that is not defined in the cross-group table.

Table 2-79 Enable function status returns and user actions

PSYM_LINKMGT Return Code	Description	Outcome	Description/User Action
NOTRANSPORT	The selected group does not have any cross- group entries with specified transport	Error	No cross-group entries can be enabled because you requested the enable function for a group or groups that does not have a cross-group connection entry that uses the specified transport.
OPERATIONFAIL	The command was unable to be successfully completed	Error	The enable function failed due to a system resource problem. Check the Connect Server to determine if it is running out of virtual memory. Check the log file to see if the cause of the error has been logged.
SUCCESS	The operation successfully completed.	Success	Refer to the description of the link management response message below for a description of the data returned.

Response Message Format for Successful Enable Requests

If the Enable function is successful, the response message returns the information shown in the following table:

Table 2-80 Enable function

Field	Description
version	10
user_tag	User-specified code from the request message.
status	PAMS__SUCCESS
group_number	Group number or numbers to receive the action.
in_link_state	PSYM_LINKMGT_ENABLED
out_link_state	PSYM_LINKMGT_ENABLED
connect_type	Transport that message is connected over: PSYM_LINKMGT_TCPIP.

Table 2-80 Enable function

platform_id	Connected platform ID (PSYM_PLATFORM_*xxxx*).
reconnect_time	Reconnect timer value for this group.
window_size	Window size value negotiated for this group.
window_delay	Window delay value negotiated for this group.
transport_addr_len	Length of the transport_addr string.
transport_addr	ASCII representation of either the TCP/IP port number.
node_name_len	Length of the node_name string.
node_name	Name of the node this link is connected to.

Disable Function

The Disable function of the link management request message disables a link's address entries if they have been enabled. This prevents a link from occurring with the group's selected addresses. Connection attempts to and from the selected addresses are prevented.

All addresses in the group address table that match the selection criteria of the message (for example, group ID, connect type, node name, and transport address) will be disabled. All other address entries for the groups selected will not be affected. If no entry matches the group_number field, then PSYM_LINKMGT_NOGROUP is returned.

The Disable function takes matching cross-group entries out of the search list for connect processing.

Request Message Format for the Disable Function

Table 2-81 displays the Disable function message format:

Table 2-81 Disable Function Message Format

Field	Required/ Optional	Setting
version	Required	10
user_tag	Required	User-specified code identifying the request.

Table 2-81 Disable Function Message Format

Field	Required/ Optional	Setting
function_code	Required	PSYM_LINKMGT_CMD_DISABLE
group_number	Required	Group number to receive the action. Valid values are 1 to 32000. The PSYM_LINKMGT_ALL_GROUPS symbol indicates all known links for this group.
connect_type	Required	Select the following transport type: PSYM_LINKMGT_TCPIP
reconnect_timer	Optional	PSYM_LINKMGT_USE_PREVIOUS
window_size	Optional	PSYM_LINKMGT_USE_PREVIOUS
window_delay	Optional	PSYM_LINKMGT_USE_PREVIOUS
transport_addr	Optional	Transport address string 16 bytes in length; the TCP/IP port ID
transport_addr_len	Optional	Length of transport address. Valid values are 0 to 16 bytes. Zero indicates to use the previous setting.
node_name	Optional	ASCII text of node name. The length is determined by node_name_len up to 255 characters.
node_name_le	Optional	Length of the node name string. Zero indicates to use the previous known value.

Determining the Status of the Disable Request

The status field of the LINKMGT_RESP message contains a return code indicating the outcome of the Disable request. See Table 2-82 for a description of each status return and the corresponding user action.

Table 2-82 Disconnect function status returns and user actions

PSYM_LINKMGT Return Code	Description	Outcome	Description/User Action
MSGCONTENT	Invalid value in request message	Error	One of the field values in the disable request message is invalid. Check the syntax of the request message against the list of valid values and re-issue the corrected request message.
MSGFMT	Unknown request version or function code	Error	Correct the syntax of the request message. The version field of the must contain the number 10. The function code field must contain the symbol PSYM_LINKMGT_CMD_DISABLE.
NOGROUP	The selected group does not have a cross group entry	Error	No cross-group entries can be disabled because you requested the disable function for a group that is not defined in the cross-group table.
NOTRANSPORT	The selected group does not have any cross group entries with specified transport	Error	No cross-group entries can be disabled because you requested the disable function for a group or groups that does not have a cross-group connection entry that uses the specified transport.
OPERATIONFAIL	The command was unable to be successfully completed	Error	The disable function failed due to a system resource problem. • Check the Connect Server to determine if it is running out of virtual memory. • Check the log file to see if the cause of the error has been logged.
SUCCESS	The operation successfully completed.	Success	Refer to the description of the link management response message below for a description of the data returned.

Response Message Format for Successful Disable Requests

If the Disable function completes successfully, the response message contains the -following information:

Table 2-83 Response Message Format for Successful Disable Requests

Field	Description
version	10
user_tag	User-specified code from the request message.
status	PSYM_LINKMGT_SUCCESS
group_number	Group number that receives the action.
in_link_state	PSYM_LINKMGT_DISABLED
out_link_state	PSYM_LINKMGT_DISABLED
connect_type	Transport that message is connected over:PSYM_LINKMGT_TCPIP.
platform_id	Connected platform ID (PSYM_PLATFORM_*xxxx*).
reconnect_time	Reconnect timer value for this group.
window_size	Window size value negotiated for this group.
window_delay	Window delay value negotiated for this group.
transport_addr_len	Length of the transport_addr string.
transport_addr	ASCII representation of either the TCP/IP port number.
node_name_len	Length of the node_name string.
node_name	Name of the node this link is connected to.

Connect Function

The Connect function of the link management request message re-enables a link's address entries if they have been disabled, and causes an immediate connect attempt to occur with the selected groups if not already connected.

Incoming connections are then allowed to occur. This function will still be able to complete even if the link is already connected. The effects of the function will not be visible until the existing link is lost.

All addresses in the group address table that match the selection criteria of the message (for example, group ID, connect type, node name, and transport address) will be enabled, and all other address entries for the groups selected will be disabled. If a matching entry is not found, then one will be created, providing the group exists. If the window or reconnect timer information is supplied, then those values will overwrite the existing information of the selected entries.

If the group_number field is set to `PSYM_LINKMGT_ALL_GROUPS`, then node name and transport address cannot be specified. If a specific group number is specified, and `PSYM_LINKMGT_ALL_TRANSPORTS` is specified, then node name and transport address cannot be specified.

On OpenVMS systems, the Connect function requires that the requesting process have either `OPER` or the `DMQ$OPERATOR` rights identifier.

Request Message Format for the Connect Function

Table 2-84 displays the Connect request function message format:

Table 2-84 Connect Request Function Message Format

Field	Required/ Optional	Setting
`version`	Required	10
`user_tag`	Required	User-specified code identifying the request, if supplied.
`function_code`	Required	PSYM_LINKMGT_CMD_CONNECT
`group_number`	Required	Group number to receive the action. Valid values are 1 to 32000. The PSYM_LINKMGT_ALL_GROUPS symbol indicates all known links for this group.
`connect_type`	Required	Select the following transport type: PSYM_LINKMGT_TCPIP

Table 2-84 Connect Request Function Message Format

Field	Required/ Optional	Setting
reconnect_timer	Optional	Time it takes for the COM Server to reconnect to a communications link. Enter the number of seconds or the following values: PSYM_LINKMGT_NO_TIMER PSYM_LINKMGT_USE_PREVIOUS
window_size	Optional	Size of transmission window (cross-group protocol Version 3.0 or higher).
window_delay	Optional	Transmission window delay in seconds (cross-group protocol Version 3.0 or higher).
transport_addr	Optional	Transport address string 16 bytes in length' the TCP/IP port ID
transport_addr_ len	Optional	Length of transport address. Valid values are 0 to 16 bytes. Zero specifies the use of the previous setting.
node_name	Optional	ASCII text of node name. The length is determined by node_name_len up to 255 characters.
node_name_len	Optional	Length of the node name string. Zero specifies the use of the previous known value.

Determining the Status of the Connect Request

The status field of the LINKMGT_RESP message contains a return code indicating the outcome of the Connect request. See Table 2-85 for a description of each status return and the corresponding user action.

Table 2-85 Connect function status returns and user actions

PSYM_LINKMGT Return Code	Description	Outcome	Description/User Action
ALREADYUP	The link is already active	Warning	The Connect function completed although the link entries were already available.
MSGCONTENT	Invalid value in request message	Error	One of the field values in the connect request message is invalid. Check the syntax of the request message against the list of valid values and re-issue the corrected request message.
MSGFMT	Unknown request version or function code	Error	Correct the syntax of the request message. The version field of the must contain the number 10. The function code field must contain the symbol PSYM_LINKMGT_CMD_CONNECT.
NOGROUP	The selected group does not have a cross group entry	Error	No cross-group links can be connected because you requested the connect function for a group that is not defined in the cross-group table.
NOTRANSPORT	The selected group does not have any cross group entries with specified transport	Error	No cross-group links can be connected because you requested the connect function for a group or groups that does not have a cross-group connection entry using the specified transport.
OPERATIONFAIL	The command was unable to be successfully completed	Error	The connect function failed due to a system resource problem. Check the Connect Server to determine if it is running out of virtual memory. Check the log file to see if the cause of the error has been logged.
SUCCESS	The operation successfully completed.	Success	Refer to the description of the link management response message below for a description of the data returned.

Response Message Format for Successful Connect Requests

If the Connect request is successful, the response message contains the following -information:

Table 2-86 Response Message Format for Successful Disable Requests

Field	Description
`version`	10
`user_tag`	User-specified code from the request message.
`status`	PSYM_LINKMGT_SUCCESS
`group_number`	Group number that receives the action.
`in_link_state`	PSYM_LINKMGT_CONNECTED
`out_link_state`	PSYM_LINKMGT_CONNECTED
`connect_type`	Transport that message is connected over: PSYM_LINKMGT_TCPIP.
`platform_id`	Connected platform ID (PSYM_PLATFORM_*xxxx*).
`reconnect_time`	Reconnect timer value for this group.
`window_size`	Window size value negotiated for this group.
`window_delay`	Window delay value negotiated for this group.
`transport_addr_len`	Length of the transport_addr string.
`transport_addr`	ASCII representation of either the TCP/IP port number.
`node_name_len`	Length of the node_name string.
`node_name`	Name of the node this link is connected to.

Disconnect Function

The Disconnect function of the link management request message requests implicit disables of links and disconnects any links to the requested group. All addresses in the group address table that match the selection criteria of the message (for example, group ID, connect type, node name, and transport address) will be disconnected. All other address entries for the groups selected will not be affected. If no entry matches the group_number field, then `PSYM_LINKMGT_NOGROUP`

is returned. On OpenVMS systems, the Disconnect function requires that the requesting process have either OPER or the DMQ$OPERATOR rights identifier.

Request Message Format for the Disconnect Function

Table 2-87 displays the Disconnect function message format.

Table 2-87 Disconnect Function Message Format

Field	Required/ Optional	Setting
user_tag	Required	10
function_code	Required	User-specified code identifying the request.
group_number	Required	PSYM_LINKMGT_CMD_DISCONNECT
connect_type	Required	Group number to receive the action. Valid values are 1 to 32000. The PSYM_LINKMGT_ALL_GROUPS symbol means disconnect all known links for this group.
reconnect_timer	Required	Select the following transport type: PSYM_LINKMGT_TCPIP
window_size	Optional	PSYM_LINKMGT_USE_PREVIOUS
window_delay	Optional	PSYM_LINKMGT_USE_PREVIOUS
transport_addr	Optional	PSYM_LINKMGT_USE_PREVIOUS
transport_addr_ len	Optional	Transport address string 16 bytes in length; the TCP/IP port ID
node_name	Optional	Length of transport address. Valid values are 0 to 16 bytes. Zero specifies the use of the previous setting.
node_name_len	Optional	ASCII text of node name. The length is determined by node_name_len up to 255 characters.

Determining the Status of the Disconnect Request

The status field of the LINKMGT_RESP message contains a return code indicating the outcome of the Disconnect request. Refer to Table 2-88 for a description of each status return and the corresponding user action.

Table 2-88 Disconnect function status returns and user actions

PSYM_LINKMGT Return Code	Description	Outcome	Description/User Action
MSGCONTENT	Invalid value in request message	Error	One of the field values in the disconnect request message is invalid. Check the syntax of the request message against the list of valid values and re-issue the corrected request message.
MSGFMT	Unknown request version or function code	Error	Correct the syntax of the request message. The version field must contain the number 10. The function code field must contain the symbol PSYM_LINKMGT_CMD_DISCONNECT.
NOGROUP	The selected group does not have a cross-group entry	Error	No cross-group connections can be disconnected because you requested the disconnect function for a group that is not defined in the cross-group table.
NOGROUP	The selected group does not have a cross group entry	Error	No cross-group links can be connected because you requested the connect function for a group that is not defined in the cross-group table.
NOTRANSPORT	The selected group does not have any cross group entries with specified transport	Error	No cross-group links can be connected because you requested the connect function for a group or groups that does not have a cross-group connection entry using the specified transport.

Table 2-88 Disconnect function status returns and user actions

PSYM_LINKMGT Return Code	Description	Outcome	Description/User Action
OPERATIONFAIL	The command was unable to be successfully completed	Error	The connect function failed due to a system resource problem. • Check the Connect Server to determine if it is running out of virtual memory. • Check the log file to see if the cause of the error has been logged.
SUCCESS	The operation successfully completed.	Success	Refer to the description of the link management response message below for a description of the data returned.

Response Message Format for Successful Disconnect Functions

If the Disconnect function is successful, the response message returns the following -information:

Table 2-89 Response Message Format for Successful Disconnect Functions

Field	Description
version	10
user_tag	User-specified code from the request message.
status	PSYM_LINKMGT_SUCCESS
group_number	Group number that receives the action.
in_link_state	PSYM_LINKMGT_DISABLED
out_link_state	PSYM_LINKMGT_DISABLED
connect_type	Transport that message is connected over: PSYM_LINKMGT_TCPIP.
platform_id	Connected platform ID (PSYM_PLATFORM_*xxxx*).
reconnect_time	Reconnect timer value for this group.

Table 2-89 Response Message Format for Successful Disconnect Functions

Field	Description
window_size	Window size value negotiated for this group.
window_delay	Window delay value negotiated for this group.
transport_addr_len	Length of the transport_addr string.
transport_addr	ASCII representation of either the TCP/IP port number.
node_name_len	Length of the node_name string.
node_name	Name of the node this link is connected to.

Link Management Design Considerations

Table 2-90 lists important design considerations for applications using link management.

Table 2-90 Link Management Design Condsiderations

Feature	Description
Failover Node Table Disabled	When an application issues a LINKMGT_REQ request, the Connect Server disables the failover node table defined in the group initialization file. Disabling the failover node table ensures the application complete control over the attributes of the link request.
Additional Group Connections Disabled	When the application issues a LINKMGT_REQ request to disconnect a link, the Connect Server disables further connections to the group. Disabling connections ensures that no additional links to the group will occur until the application issues another LINKMGT_REQ request.
Connect Requests - Verified	When a connect request is made for a single group, the XGROUP_VERIFY table uses the information supplied in the message to determine whether to accept or reject the -request for a connection. Cross-group verification only works on incoming requests. The data structure for cross-group verification is overwritten by the information in the link management connect or disconnect message.

Table 2-90 Link Management Design Condsiderations

Feature	Description
Connect and Disconnect Requests Acknowledged	When the Connect Server receives a connect message after a link is already successfully connected, the Connect Server rejects the second connect message. When the Connect Server receives a disconnect message after a link is already successfully disconnected, the Connect Server acknowledges the second disconnect message with a successful -return message.
Restrictions on Local and Remote Requests	The Connect Server will only accept link control requests from a local application. However, the Connect Server will accept link status inquiries from remote as well as local -applications.
Privileges Required	Application link control requests on the OpenVMS system require that the application have VMS `OPER` privilege or be granted the `DMQ$OPERATOR` rights identifier.

Learning the Current Status of Queues

This section describes how applications can use Queue Server message-based services to obtain status information on all active queues in a particular group or to obtain notification of queue status changes. The list of active queues displays all attached permanent and temporary queues.

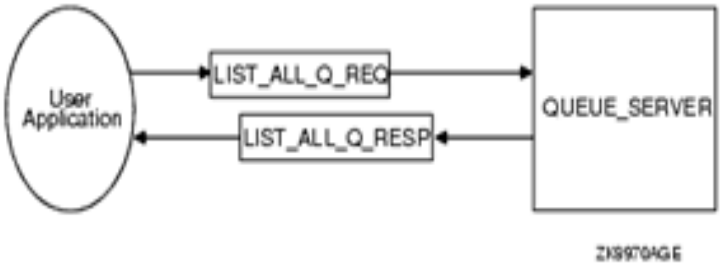
Listing Attached Queues in a Group

The Queue Server process can provide applications with a list of all attached queues for a selected group. This information is available for local and remote groups and includes a listing of both permanent and temporary queues. To request this list, the application program sends a message of type `LIST_ALL_Q_REQ` to the Queue Server process.

To learn the status of all queues in a selected group, an application exchanges the following messages with the Queue Server:

- `LIST_ALL_Q_REQ`-Request message to request the status of all queues.
- `LIST_ALL_Q_RESP`-Response message to provide a list of all queues and their status.

Figure 2-22 Figure 5-5 Listing All Queues



The application receives a response message from the Queue Server of type `LIST_ALL_Q_RESP` providing a list of all attached queues. Because a `LIST_ALL_Q_RESP` message may contain a long list of queue names, the application must allocate a sufficient buffer size to store the information returned.

Receiving Attachment Notifications

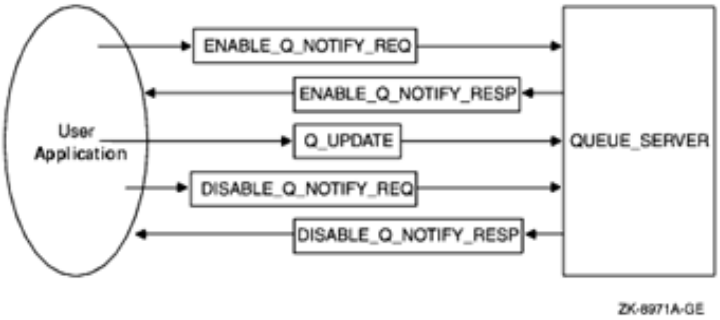
The Queue Server process can notify an application of all attached queues and subsequent queue attachments and detachments for its own group. An application registers for this service by sending a message of type `ENABLE_Q_NOTIFY_REQ` to the group's Queue Server process. The Queue Server responds with a message of type `ENABLE_Q_NOTIFY_RESP`, indicating the status of the registration request.

To learn the status of all queues and receive ongoing notification of new queue attachments and detachments, an application exchanges the following messages with the Queue Server:

- `ENABLE_Q_NOTIFY_REQ`-Request message to request the current status of all queues with notification of future queue status changes.
- `ENABLE_Q_NOTIFY_RESP`-Response message to provide the current status of all queues and confirmation that queue status changes will be reported.
- `Q_UPDATE`-Notification message to provide information on newly attached and detached queues in the selected group.
- `DISABLE_Q_NOTIFY_REQ`-Request message to request that notification of queue status changes be discontinued.

- DISABLE_Q_NOTIFY_RESP-Response message to indicate that notification of queue status changes has been successfully disabled.

Figure 2-23 Listing Available Queues



The registration request places the sender's response queue number in the list of applications to receive notification of new attachments and detachments. Notifications are sent using a message of type Q_UPDATE. The application can cancel the notification registration by sending a message of type DISABLE_Q_NOTIFY_REQ. The Queue Server responds with a reply of type DISABLE_Q_NOTIFY_RESP indicating the status of the registration cancellation request.

Managing Message Recovery Files

Oracle MessageQ message-based services are used with the MRS Server to maintain files for recoverable messaging and to turn MRS journaling capability on or off. Message-based services for performing these functions are available on OpenVMS systems only. The functions are also available through the Oracle MessageQ Manager Utility on OpenVMS systems. For complete information on how to use the Oracle MessageQ message recovery system, see the Sending Recoverable Messages topic.

Oracle MessageQ uses the following four Oracle MessageQ files for MRS message-based services:

Table 2-91 Oracle MessageQ files for MRS message-based services

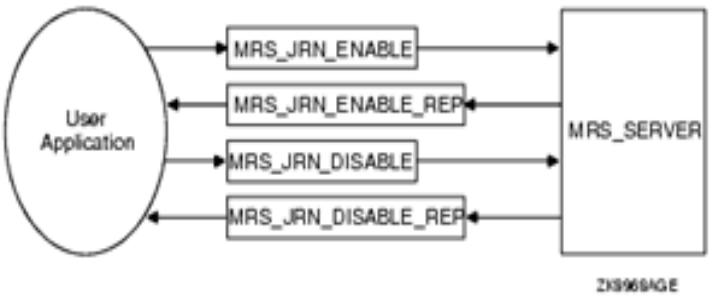
Store and forward file (SAF)	Messages designated for recovery on the sender system.
Destination queue file (DQF)	Messages designated for recovery on the receiver system.
Dead letter journal (DLJ)	Undelivered messages not designated for recovery by Oracle MessageQ. These messages can be delivered later from the DLJ by an application program.
Postconfirmation journal (PCJ)	Successfully delivered recoverable messages which form an audit trail of messaging events.

Controlling Journaling to the PCJ File

You can use the messages in Table 2-79 to disable journaling when replacing a PCJ file and then reenable journaling:

- MRS_JRN_DISABLE-Request message to disable journaling to the PCJ file.
- MRS_JRN_DISABLE_REP-Response message to indicate the status of the request.
- MRS_JRN_ENABLE-Request message to enable journaling to the PCJ file.
- MRS_JRN_ENABLE_REP-Response message to indicate the status of the request.

Figure 2-24 Disabling Journaling



Use the MRS_JRN_DISABLE message to disable journaling to the PCJ when you need to close the PCJ and open a new one. The MRS_JRN_DISABLE_REP message returns the status of the operation. Use the MRS_JRN_ENABLE message to enable journaling after you have opened a new PCJ file. The MRS_JRN_ENABLE_REP message returns the status of the operation.

Message Reference

This chapter contains detailed descriptions of all Oracle MessageQ message-based services alphabetized by message type. Each description lists the message type code name, the name of the Oracle MessageQ server performing the service, and a detailed definition of the message area and required arguments to send messages or read response and notification messages using the Oracle MessageQ API or scripts. The definition of all Oracle MessageQ message- based services messages is now provided in the p_msg.h include file.

Oracle MessageQ message-based services are sent between a user application program that functions as a requestor and a Oracle MessageQ server process that fulfills the request. For messages to be properly understood between systems, message data must be sent and returned in the endian format understood by both the requestor and the server. Most Oracle MessageQ message-based services automatically perform this conversion if the endian format of the two systems is different. However, some message-based services do not perform this conversion, therefore, the user application must convert the message to the endian format of the server system to ensure that the message data is correctly interpreted. Each message-based service description notes whether the data structure is RISC aligned and whether the server performs the endian conversion automatically. Please note that the environment variables VIEWFILES and VIEWDIR should be setted properly before using message-based services. For instance, export VIEWDIR=$TUXDIR/udataobj and VIEWFILES=otmq_mbs.V.

Table 2-92 Oracle MessageQ message-based services alphabetized by message type

- AVAIL
- AVAIL_DEREG
- AVAIL_REG
- AVAIL_REG_REPLY
- LINKMGT_REQ
- LINKMGT_RESP
- LINK_COMPLETE
- LINK_LOST
- LIST_ALL_Q_REQ
- LIST_ALL_Q_RESP
- LOCATE_Q_REP
- MRS_ACK
- MRS_JRN_ENABLE_REP
- Q_UPDATE
- SBS_DEREGISTER_REQ
- SBS_DEREGISTER_RESP
- UNAVAIL

- DISABLE_NOTIFY
- DISABLE_Q_NOTIFY_REQ
- DISABLE_Q_NOTIFY_RESP
- ENABLE_NOTIFY
- ENABLE_Q_NOTIFY_REQ
- ENABLE_Q_NOTIFY_RESP
- LIST_ALL_CONNECTIONS (Request)
- LIST_ALL_CONNECTIONS (Response)
- LIST_ALL_ENTRIES (Request)
- LIST_ALL_ENTRIES (Response)
- LIST_ALL_GROUPS (Request)
- LIST_ALL_GROUPS (Response)
- MRS_JRN_DISABLE
- MRS_JRN_DISABLE_REP
- MRS_JRN_ENABLE
- MRS_JRN_ENABLE
- MRS_JRN_ENABLE_REP
- SBS_REGISTER_REQ
- SBS_REGISTER_RESP
- SBS_SEQUENCE_GAP
- SBS_STATUS_REQ
- SBS_STATUS_RESP
- TIMER_EXPIRED

Note: In the section entitled "See Also" for every service, you will find a list of related Oracle MessageQ message-based services. For more information about these services, see "Message Reference" section.

AVAIL

Applications can register to receive notification when queues become active or inactive in local and remote groups by sending an AVAIL_REG message to the Avail Server. The AVAIL notification message is sent to registered applications when a queue in the selected group becomes active. See the Obtaining the Status of a Queue topic in the Using Message-Based Services section for an explanation of how to use this message.

Applications must cancel availability notification by sending a message of type AVAIL_DEREG. The application receives a AVAIL_REG_REPLY message indicating the status of the operation. It is important to note that if the distribution queue for an AVAIL registration becomes unavailable, the registration will be automatically deleted by Oracle

MessageQ. A subsequent attempt to deregister AVAIL services for this distribution queue will result in an error message indicating that the registration does not exist.

Note: The Avail Server performs endian conversion when this message is received between processes that run on systems that use different hardware data formats. This message is also RISC aligned.

C Message Structure

```
typedef struct _AVAIL
{ q_address target_q;
} AVAIL;
```

Message Data Fields

Table 2-93 Message Data Fields

Field	Data Type	Script Format	Description
target_q	q_address	DL	Address of queue that is now available.

Arguments

Table 2-94 Arguments

Argument	Data Type	Mechanism	Prototype	Access
Target	Supplied by AVAIL_REG	Supplied by AVAIL_REG	Target	Supplied by AVAIL_REG
Source	AVAIL_SERVER	PAMS__AVAIL_SERVER	Source	AVAIL_SERVER
Class	PAMS	MSG_CLAS_PAMS	Class	PAMS
Type	AVAIL	MSG_TYPE_AVAIL	Type	AVAIL

See Also

- AVAIL_DEREG
- AVAIL_REG

- AVAIL_REG_REPLY
- UNAVAIL

Example

The AVAIL services example illustrates avail services, avail register, avail deregister, and getting avail messages. The complete code example called x_avail.c is contained in the examples directory.

AVAIL_DEREG

Applications can register to receive notification when queues become active or inactive in local and remote groups by sending an AVAIL_REG message to the Avail Server. When notification messages are no longer needed, the application sends an AVAIL_DEREG message to the Avail Server to cancel registration. It is important to note that if the distribution queue for an AVAIL registration becomes unavailable, the registration will be automatically deleted by Oracle MessageQ. A subsequent attempt to deregister AVAIL services for this distribution queue will result in an error message indicating that the registration does not exist. See the Obtaining the Status of a Queue topic in the Using Message-Based Services section for an explanation of how to use this message.

Note: The Avail Server performs endian conversion when this message is sent between processes that run on systems that use different hardware data formats. This message is also RISC aligned.

C Message Structure

```
typedef struct _AVAIL_DEREG { int16 version;
int16 filler; q_address target_q;
q_address distribution_q; char req_ack;
} AVAIL_DEREG;
```

Message Data Fields

Table 2-95 Message Data Fields

Field	Data Type	Script Format	Description
version	word	DW	Format version number. Must be 20.

Table 2-95 Message Data Fields

filler	word	DW	Spacing for RISC alignment.
target_q	q_address	DL	Queue being monitored for its -availability.
distribution_q	q_address	DL	Queue notified of availability.
req_ack	Boolean	DB	If response required, 1; else 0.

Arguments

Table 2-96 Arguments

Argument	Script Format	pams_get_msg Format
Target	AVAIL_SERVER	PAMS__AVAIL_SERVER
Source	Supplied by Oracle MessageQ	Supplied by Oracle MessageQ
Class	PAMS	MSG_CLAS_PAMS
Type	AVAIL_DEREG	MSG_TYPE_AVAIL_DEREG

See Also

- AVAIL
- AVAIL_REG
- AVAIL_REG_REPLY
- UNAVAIL

Example

The AVAIL services example illustrates avail services, avail register, avail deregister, and getting avail messages. The complete code example called x_avail.c is contained in the examples directory.

AVAIL_REG

Applications can register to receive notification when queues become active or inactive in local and remote groups by sending an AVAIL_REG message to the Avail Server. See the Obtaining the Status of a Queue topic in the Using Message-Based Services section for an explanation of how to use this message. If the application detaches from the distribution queue, the AVAIL registration is automatically deleted. The application must cancel notification, regardless of queue type, by sending a message of type AVAIL_DEREG. The application receives a AVAIL_REG_REPLY message indicating the status of the operation.

Note: The Avail Server performs endian conversion when this message is sent between processes that run on systems that use different hardware data formats. This message is also RISC aligned.

C Message Structure

```
typedef struct _AVAIL_REG { int16 version;

int16 filler; q_address target_q;

q_address distribution_q; int32 timeout;

} AVAIL_REG;
```

Message Data Fields

Table 2-97 Message Data Fields

Field	Data Type	Script Format	Description
version	word	DW	Format version number. Must be 31.
filler	word	DW	Spacing for RISC alignment.
target_q	q_address	DL	Queue to be monitored for availability.
distribution_q	q_address	DL	Queue to receive availability messages.
timeout	int32	DL	Interval (specified in seconds) after which the function should timeout.

Arguments

Table 2-98 Arguments

Argument	Script Format	pams_get_msg Format
Target	AVAIL_SERVER	PAMS__AVAIL_SERVER
Source	Supplied by Oracle MessageQ	Supplied by Oracle MessageQ
Class	PAMS	MSG_CLAS_PAMS
Type	AVAIL_REG	MSG_TYPE_AVAIL_REG

See Also

- AVAIL_REG_REPLY
- AVAIL
- UNAVAIL
- AVAIL_DEREG

Example

The AVAIL services example illustrates avail services, avail register, avail deregister, and getting avail messages. The complete code example called x_avail.c is contained in the examples directory.

AVAIL_REG_REPLY

Applications register to receive notification when queues become active or inactive in local and remote groups by sending an AVAIL_REG message to the Avail Server. The AVAIL_REG_REPLY message indicates whether the application has successfully registered or deregistered from receiving notification messages. See the Obtaining the Status of a Queue topic in the Using Message-Based Services section for an explanation of how to use this message.

Note: The Avail Server performs endian conversion when this message is received between processes that run on systems that use different hardware data formats. This message is also RISC aligned.

C Message Structure

```
typedef struct _AVAIL_REG_REPLY { int16 status;
uint16 reg_id; int16 number_reg;
} AVAIL_REG_REPLY;
```

Message Data Fields

Table 2-99 Message Data Fields

Field	Data Type	Script Format	Description
status	word	DW	Status code: 1 = success; 0 = failure.
reg_id	unsigned word	DW	Returned subscription ID.
number_reg	word	DW	Number of registrants left on the Avail list.

Arguments

Table 2-100 Arguments

Argument	Script Format	pams_get_msg Format
Target	Sender of AVAIL_REG/DEREG	Sender of AVAIL_REG/DEREG
Source	AVAIL_SERVER	PAMS__AVAIL_SERVER
Class	PAMS	MSG_CLAS_PAMS
Type	AVAIL_REG_REPLY	MSG_TYPE_AVAIL_REG_REPLY

See Also

- AVAIL_REG
- AVAIL_DEREG

- AVAIL
- UNAVAIL

Example

The AVAIL services example illustrates avail services, avail register, avail deregister, and getting avail messages. The complete code example called x_avail.c is contained in the examples directory.

DISABLE_NOTIFY

Applications can register to receive notification when cross-group links are established and lost by sending an ENABLE_NOTIFY message to the Connect Server. When an application no longer needs to receive notification messages, it deregisters by sending a DISABLE_NOTIFY message to the Connect Server. The DISABLE_NOTIFY message can stop notification of cross-group link changes. See the Obtain Notification of Cross-Group Links Established and Lost topic in the Using Message-Based Services section for an explanation of how to use this message.

Note: The Connect Server performs endian conversion when this message is sent between processes that run on systems that use different hardware data formats. This message is also RISC aligned.

C Message Structure

```
typedef struct _ENABLE_NOTIFY { char reserved; char connection_flag;}
ENABLE_NOTIFY;
```

Message Data Fields

Table 2-101 Message Data Fields

Field	Data Type	Script Format	Description
reserved	unsigned char	DB	Reserved for use by Oracle MessageQ.
connection_flag	unsigned char	DB	Boolean flag to cancel cross-group connection notification, 1; else 0.

Arguments

Table 2-102 Arguments

Argument	Script Format	pams_get_msg Format
Target	CONNECT_SERVER	PAMS__CONNECT_SERVER
Source	Supplied by Oracle MessageQ	Supplied by Oracle MessageQ
Class	PAMS	MSG_CLAS_PAMS
Type	DISABLE_NOTIFY	MSG_TYPE_DISABLE_NOTIFY

See Also

- ENABLE_NOTIFY
- LINK_COMPLETE
- LINK_LOST

DISABLE_Q_NOTIFY_REQ

Applications can register to receive notification when queue states change in local or remote groups by sending an ENABLE_Q_NOTIFY_REQ message. The DISABLE_Q_NOTIFY_REQ is sent to the Queue Server when the application no longer needs to receive notification messages. See the Receiving Attachment Notifications topic in the Using Message-Based Services section for an explanation of how to use this message.

Note: The Queue Server performs endian conversion when this message is sent between processes that run on systems that use different hardware data formats. This message is also RISC aligned.

C Message Structure

```
typedef struct _Q_NOTIFY_REQ { int32 version;
int32 user_tag;
} Q_NOTIFY_REQ;
```

Message Data Fields

Table 2-103 Message Data Fields

Field	Data Type	Script Format	Description
version	int32	DL	Version of request.
user_tag	int32	DL	User-specified code to identify this request.

Arguments

Table 2-104 Arguments

Argument	Script Format	pams_get_msg Format
Target	QUEUE_SERVER	PAMS__QUEUE_SERVER
Source	Requesting program	Requesting program
Class	PAMS	MSG_CLAS_PAMS
Type	DISABLE_Q_NOTIFY_REQ	MSG_TYPE_DISABLE_Q_ NOTIFY_REQ

See Also

- DISABLE_Q_NOTIFY_RESP
- ENABLE_Q_NOTIFY_REQ
- ENABLE_Q_NOTIFY_RESP
- Q_UPDATE

DISABLE_Q_NOTIFY_RESP

Applications can register to receive notification when queue states change in local or remote groups by sending an ENABLE_Q_NOTIFY_REQ message. The DISABLE_Q_NOTIFY_REQ

message is sent to the Queue Server when the application no longer needs to receive notification messages. The DISABLE_Q_NOTIFY_RESP message indicates whether the application is successfully deregistered from receiving notification messages. See the Receiving Attachment Notifications topic in the Using Message-Based Services section for an explanation of how to use this message.

Note: The Queue Server performs endian conversion when this message is received between processes that run on systems that use different hardware data formats. This message is also RISC aligned.

C Message Structure

```
#define MAX_NUMBER_Q_RECS 50 typedef struct _Q_NOTIFY_RESP {int32 version;
int32 user_tag; int32 status_code; int32 last_block_flag; int32
number_q_recs; struct{

q_address q_num; q_address q_owner; int32q_type;int32q_active_flag; int32
q_attached_flag; int32q_owner_pid;} q_rec [50];} Q_NOTIFY_RESP;
```

Message Data Fields

Table 2-105 Message Data Fields

Field	Data Type	Script Format	Description
version	int32	DL	Version of response.
user_tag	int32	DL	User-specified code from request.
status_code	int32	DL	0=Error 1=Success -2=Refused
last_block_flag	int32	DL	Last block Boolean flag.
number_q_recs	int32	DL	Number of records in this message.
q_num	q_address	DL	Queue number.
q_owner	q_address	DL	Queue owner (only for secondary queues (SQs)).

Table 2-105 Message Data Fields

q_type	int32	DL	Queue type (numerically encoded P, S, M).
q_active_flag	int32	DL	Queue active Boolean flag.
q_attached_flag	int32	DL	Queue attached Boolean flag.
q_owner_pid	int32	DL	Queue owner process identification (PID).

Arguments

Table 2-106 Arguments

Argument	Script Format	pams_get_msg Format
Target	Requesting program	Requesting program
Source	QUEUE_SERVER	PAMS__QUEUE_SERVER
Class	PAMS	MSG_CLAS_PAMS
Type	DISABLE_Q_NOTIFY_RESP	MSG_TYPE_DISABLE_Q_ NOTIFY_RESP

See Also

- DISABLE_Q_NOTIFY_REQ
- ENABLE_Q_NOTIFY_REQ
- ENABLE_Q_NOTIFY_RESP
- Q_UPDATE

ENABLE_NOTIFY

Applications can register to receive notification when cross-group links are established and lost by sending an ENABLE_NOTIFY message to the Connect Server. See the Obtain

Notification of Cross-Group Links Established and Lost topic in the Using Message-Based Services section for an explanation of how to use this message.

Note: The Connect Server performs endian conversion when this message is sent between processes that run on systems that use different hardware data formats. This message is also RISC aligned.

C Message Structure

```
typedef struct _ENABLE_NOTIFY { char reserved;
char connection_flag;
} ENABLE_NOTIFY;
```

Message Data Fields

Table 2-107 Message Data Fields

Field	Data Type	Script Format	Description
reserved	unsigned char	DB	Reserved for use by Oracle MessageQ.
connection_flag	unsigned char	DB	Boolean flag for cross-group connection notification, 1; else 0.

Arguments

Table 2-108 Arguments

Argument	Script Format	pams_get_msg Format
Target	CONNECT_SERVER	PAMS__CONNECT_SERVER
Source	Supplied by Oracle MessageQ	Supplied by Oracle MessageQ
Class	PAMS	MSG_CLAS_PAMS
Type	ENABLE_NOTIFY	MSG_TYPE_ENABLE_NOTIFY

See Also

DISABLE_NOTIFY

LINK_COMPLETE

LINK_LOST

ENABLE_Q_NOTIFY_REQ

Applications can register to receive notification when queue states change in local or remote groups by sending an ENABLE_Q_NOTIFY_REQ message. This message requests a list of all active queues and then subsequent notification when queues become attached or detached and active or inactive. See the Receiving Attachment Notifications topic in the Using Message-Based Services section for an explanation of how to use this message.

Note: The Queue Server performs endian conversion when this message is sent between processes that run on systems that use different hardware data formats. This message is also RISC aligned.

C Message Structure

```
typedef struct _Q_NOTIFY_REQ { int32 version;
int32 user_tag;
} Q_NOTIFY_REQ;
```

Message Data Fields

Table 2-109 Message Data Fields

Field	Data Type	Script Format	Description
version	int32	DL	Version of request.
user_tag	int32	DL	User-specified code to identify this request.

Arguments

Table 2-110 Arguments

Argument	Script Format	pams_get_msg Format
Target	QUEUE_SERVER	PAMS__QUEUE_SERVER
Source	Requesting program	Requesting program
Class	PAMS	MSG_CLAS_PAMS
Type	ENABLE_Q_NOTIFY_REQ	MSG_TYPE_ENABLE_Q_NOTIFY_REQ

See Also

- DISABLE_Q_NOTIFY_REQ
- DISABLE_Q_NOTIFY_RESP
- ENABLE_Q_NOTIFY_RESP
- Q_UPDATE

ENABLE_Q_NOTIFY_RESP

Applications can register to receive notification when queue states change in local or remote groups by sending an ENABLE_Q_NOTIFY_REQ message. The ENABLE_Q_NOTIFY_RESP message delivers a list of all active queues and then subsequently notifies the application of attachments, detachments, and changes to active and inactive status using the Q_UPDATE message. See the Receiving Attachment Notifications topic in the Using Message-Based Services section for an explanation of how to use this message.

Note: The Queue Server performs endian conversion when this message is received between processes that run on systems that use different hardware data formats. This message is also RISC aligned.

C Message Structure

```
#define MAX_NUMBER_Q_RECS 50 typedef struct _Q_NOTIFY_RESP {int32 version;
int32 user_tag; int32 status_code;int32 last_block_flag; int32
number_q_recs; struct{
```

```
q_address q_num; q_address q_owner; int32q_type;int32q_active_flag; int32
q_attached_flag; int32q_owner_pid;} q_rec [50];} Q_NOTIFY_RESP;
```

Message Data Fields

Table 2-111 Message Data Fields

Field	Data Type	Script Format	Description
version	int32	DL	Version of response.
user_tag	int32	DL	User-specified code from request.
status_code	int32	DL	0=Error 1=Success -2=Refused
last_block_flag	int32	DL	Last block Boolean flag.
number_q_recs	int32	DL	Number of records in this message.
q_num	q_address	DL	Queue number.
q_owner	q_address	DL	Queue owner (only for secondary queues (SQs)).
q_type	int32	DL	Queue type (numerically encoded P, S, M).
q_active_flag	int32	DL	Queue active Boolean flag.
q_attached_flag	int32	DL	Queue attached Boolean flag.
q_owner_pid	int32	DL	Queue owner process identification (PID).

Arguments

Table 2-112 Arguments

Argument	Script Format	pams_get_msg Format
Target	Requesting program	Requesting program
Source	QUEUE_SERVER	PAMS__QUEUE_SERVER
Class	PAMS	MSG_CLAS_PAMS
Type	ENABLE_NOTIFY_RESP	MSG_TYPE_ENABLE_NOTIFY_RESP

See Also

- DISABLE_Q_NOTIFY_REQ
- DISABLE_Q_NOTIFY_RESP
- ENABLE_Q_NOTIFY_REQ
- Q_UPDATE

LINKMGT_REQ

Applications can use link management messages to explicitly control cross-group connections. Use the LINKMGT_REQ message to request a connection to a remote group, to disconnect from a remote group, or to obtain information about a remote Oracle MessageQ group. See the Controlling Cross-Group Links topic in the Using Message-Based Services section for an explanation of how to use this message.

Note: The Connect Server performs endian conversion when this message is sent between processes that run on systems that use different hardware data formats. This message is also RISC aligned.

C Message Structure

```
typedef struct _TADDRESS { int32 len;char str [16];} TADDRESS;

typedef struct _NODENAME { int32 len;char str [255];} NODENAME;
```

```
typedef struct _LINKMGT_REQ { int32 version;int32 user_tag; int32
function_code; int32 group_number; int32 connect_type;int32
reconnect_timer; int32 window_size; int32 window_delay;int32
reserved_space [10];TADDRESS transport_addr; NODENAME node_name;}
LINKMGT_REQ;
```

Message Data Fields

Table 2-113 Message Data Fields

Field	Data Type	Script Format	Description
version	int32	DL	Message version.
user_tag	int32	DL	User-specified code to identify this request.
function_code	int32	DL	Function of the message using PSYM_LINKMGT_CMD: _ENABLE _DISABLE _INQUIRY _CONNECT _DISCONNECT **Note:** _INQUIRY_CONNECT and _DISCONNECT : Not supported in this release for Oracle Tuxedo MP environment.
group_number	int32	DL	Group number to receive action; valid values are between 1 and 32,000; PSYM_LINKMGT_ALL_GROUPS indicates all known
connect_type	int32	DL	Type of transport to use, as follows: PSYM_LINKMGT_TCPIP

Table 2-113 Message Data Fields

reconnect_timer	int32	DL	Time it takes for the COM Server to reconnect to a communications link. Enter the number of seconds or the following values: PSYM_LINKMGT_NO_TIMER PSYM_LINKMGT_USE_PREVIOUS
window_size	int32	DL	Size of transmission window (cross-group protocol Version 3.0 and higher). Enter the number of messages or the following value: PSYM_LINKMGT_USE_PREVIOUS
window_delay	int32	DL	Transmission window delay in seconds (cross-group protocol Version 3.0 and higher). Enter the number of seconds or the following value: PSYM_LINKMGT_USE_PREVIOUS
reserved_space	10-int32 array	DL(10)	Reserved for Oracle MessageQ use.
transport_addr_len	int32	DL	Length of transport address. Values 0 to 16 bytes; 0 = use previous setting.
transport_addr	char char str *	A	Transport address string that is 16 bytes in length; the TCP/IP port ID.
node_name_len	int32	DL	Length of node name string; 0 = use previous known value.
node_name	char	A	ASCII text of node name; length determined by node_name_len up to 255 characters.

Arguments

Table 2-114 Arguments

Argument	Script Format	pams_get_msg Format
Target	CONNECT_SERVER	PAMS__CONNECT_SERVER
Source	Requesting program	Requesting program
Class	PAMS	MSG_CLAS_PAMS
Type	LINKMGT_REQ	MSG_TYPE_LINKMGT_REQ

See Also

- LINKMGT_RESP

LINKMGT_RESP

Applications can use link management messages to explicitly control cross-group connections. Use the LINKMGT_REQmessage to request a connection to a remote group, to disconnect from a remote group, or to obtain information about a remote Oracle MessageQ group. The LINKMGT_RESP message notifies the requesting application if the connection or disconnection request was successful and supplies information about the cross-group connection. See the Controlling Cross-Group Links topic in the Using Message-Based Services section for an explanation of how to use this message.

Note: The Connect Server performs endian conversion when this message is received between processes that run on systems that use different hardware data formats. This message is also RISC aligned.

C Message Structure

```
typedef struct _TADDRESS { int32 len;char str [16];
} TADDRESS;typedef struct _NODENAME { int32 len;
char str [255];} NODENAME;
```

```
typedef struct _LINKMGT_RESP { int32 version;int32 user_tag; int32 status;
int32 group_number; int32 in_link_state; int32 out_link_state; int32
connect_type; int32 platform_id; int32 reconnect_timer; int32 window_size;
int32 window_delay;int32 reserved_space [10]; TADDRESS transport_addr;
NODENAME node_name;} LINKMGT_RESP;
```

Message Data Fields

Table 2-115 Message Data Fields

Field	Data Type	Script Format	Description
version	int32	DL	Message version.
user_tag	int32	DL	User-specified code from request.
status	int32	DL	Completion status
group_number	int32	DL	Group number to receive action. Valid values are between 1 and 32,000; PSYM_LINKMGT_ALL_GROUPS indicates all
in_link_state	int32	DL	State of inbound link at time of request. Values are: PSYM_LINKMGT_UNKNOWN PSYM_LINKMGT_NOCNT PSYM_LINKMGT_CONNECTED PSYM_LINKMGT_DISABLED
out_link_state	int32	DL	State of outbound link at time of request; same values as in_link_state.
connect_type	int32	DL	Type of transport to use as follows:

Table 2-115 Message Data Fields

platform_id	int32	DL	Platform type preceded by the prefix PSYM_PLATFORM. Valid values are: VAX_VMS VAX_ULTRIX RISC_ULTRIX HP9000_HPUX MOTOROLA_VR32 SPARC_SUNOS IBM_RS6000_AIX OS2 MSDOS PDP11_RSX VAXELN MACINTOSH SCO_UNIX M68K VMS_AXP UNIX WINDOWSNT OSF1_AXP DYNIX_X86 UNKNOWN
reconnect_timer	int32	DL	Time it takes for the COM Server to reconnect to a communications link. Enter the number of seconds or the following values: PSYM_LINKMGT_NO_TIMER
window_size	int32	DL	Size of transmission window (cross-group protocol Version 3.0 and higher).
window_delay	int32	DL	Transmission window delay in seconds (cross-group protocol Version 3.0 and higher).

Table 2-115 Message Data Fields

reserved_space	10-int32 array	DL(10)	Reserved for Oracle MessageQ use.
transport_addr_ len	int32	DL	Length of transport address. Values 0 to 16 bytes; 0 = use previous setting.
transport_addr	char	A	Transport address string 16 bytes in length, the TCP/IP port ID.
node_name_len	int32	DL	Length of node name string. 0 = use previous known value.
node_name	char	A	ASCII text of node name; length determined by node_name_len up to 255 characters.

Status Code

Table 2-116 Status Code

Status Code	Description
PSYM_LINKMGT_ALREADYUP	Link already connected.
PSYM_LINKMGT_MSGCONTENT	Message incomplete or content inconsistent with dialog.
PSYM_LINKMGT_MSGFMT	Format error in dialog.
PSYM_LINKMGT_NOGROUP	Group is unknown.
PSYM_LINKMGT_NOPRIV	No privilege for attempted operation.
PSYM_LINKMGT_NOTRANSPORT	Requested transport is not available.
PSYM_LINKMGT_NOTSUPPORTED	Feature not supported.

Table 2-116 Status Code

PSYM_LINKMGT_OPERATIONFAIL	Requested operation failed.
PSYM_LINKMGT_SUCCESS	Normal successful return.

Arguments

Table 2-117 Arguments

Argument	Script Format	pams_get_msg Format
Target	Requesting program	Requesting program
Source	CONNECT_SERVER	PAMS__CONNECT_SERVER
Class	PAMS	MSG_CLAS_PAMS
Type	LINKMGT_RESP	MSG_TYPE_LINKMGT_RESP

See Also

LINKMGT_REQ

LINK_COMPLETE

Applications can register to receive notification when cross-group links are established and lost by sending an ENABLE_NOTIFY message to the Connect Server. Registered applications receive a LINK_COMPLETE message each time a cross-group connection occurs. See the Obtain Notification of Cross-Group Links Established and Lost topic in the Using Message-Based Services section for an explanation of how to use this message.

Note: The Connect Server **does not** perform endian conversion when this message is received between processes that run on systems that use different hardware data formats. The sender program must convert the message to the endian format of the target system to ensure that the message data is correctly interpreted. This message is RISC aligned.

C Message Structure

```
typedef struct _LINK_NOTIFICATION { int16 group_number;
int16 filler1; char os_type; char  filler2;
} LINK_NOTIFICATION;
```

Message Data Fields

Table 2-118 Message Data Fields

Field	Data Type	Script Format	Description
group_number	word	DW	Group address associated with link.
filler1	word	DW	Reserved for Oracle MessageQ.
os_type	byte	A(1)	Code indicating operating system of remote node.
filler2	byte	XB	Reserved for Oracle MessageQ.

Arguments

Table 2-119 Arguments

Argument	Script Format	pams_get_msg Format
Target	Requesting program	Requesting program
Source	CONNECT_SERVER	PAMS__CONNECT_SERVER
Class	PAMS	MSG_CLAS_PAMS
Type	LINK_COMPLETE	MSG_TYPE_LINK_COMPLETE

See Also

- DISABLE_NOTIFY

- ENABLE_NOTIFY
- LINK_LOST

LINK_LOST

Applications can register to receive notification when cross-group links are established and lost by sending an ENABLE_NOTIFY message to the Connect Server. Registered applications receive a LINK_LOST message each time a cross-group connection is lost. See the Obtain Notification of Cross-Group Links Established and Lost topic in the Using Message-Based Services section for an explanation of how to use this message.

Note: The Connect Server **does not** perform endian conversion when this message is received between processes that run on systems that use different hardware data formats. The sender program must convert the message to the endian format of the target system to ensure that the message data is correctly interpreted. This message is RISC aligned.

C Message Structure

```
typedef struct _LINK_NOTIFICATION { int16 group_number;int16 filler1; char
os_type; char  filler2;} LINK_NOTIFICATION;
```

Message Data Fields

Table 2-120 Message Data Fields

Field	Data Type	Script Format	Description
group_number	word	DW	Group address associated with link.
filler1	word	DW	Reserved for Oracle MessageQ.
os_type	byte	A(1)	Code indicating operating system of remote node.
filler2	byte	XB	Reserved for Oracle MessageQ.

Arguments

Table 2-121 Arguments

Argument	Script Format	pams_get_msg Format
Target	Requesting program	Requesting program
Source	CONNECT_SERVER	PAMS__CONNECT_SERVER
Class	PAMS	MSG_CLAS_PAMS
Type	LINK_LOST	MSG_TYPE_LINK_LOST

See Also

- DISABLE_NOTIFY
- ENABLE_NOTIFY
- LINK_COMPLETE

LIST_ALL_CONNECTIONS (Request)

An application can request a listing of all active and configured cross-group connections by sending a LIST_ALL_CONNECTIONS message to the Connect Server. The reply to this request is a variable-length message of the same type and class containing the cross-group connection information. See the Listing Cross-Group Connections, Entries, and Groups topic in the Using Message-Based Services section for an explanation of how to use this message.

Note: This message is RISC aligned.

C Message Structure

None.

Message Data Fields

None.

Arguments

Table 2-122 Arguments

Argument	Script Format	pams_get_msg Format
Target	CONNECT_SERVER	PAMS__CONNECT_SERVER
Source	Requesting program	Requesting program
Class	PAMS	MSG_CLAS_PAMS
Type	LIST_ALL_CONNECTIONS	MSG_TYPE_LIST_ALL_ -CONNECTIONS

See Also

- LIST_ALL_CONNECTIONS response message
- LIST_ALL_ENTRIES (Request)
- LIST_ALL_ENTRIES (Response)
- LIST_ALL_GROUPS (Request)
- LIST_ALL_GROUPS (Response)

LIST_ALL_CONNECTIONS (Response)

An application can request a listing of all active and configured cross-group connections by sending a LIST_ALL_CONNECTIONS message to the Connect Server. The reply to this request is a variable length-message of the same type and class containing the cross-group connection information. To read the information returned, the application must total the number of bytes in the reply and divide by the cross-group entry length, which is 20 bytes, to determine the number of records returned. See the Listing Cross-Group Connections, Entries, and Groups topic in the Using Message-Based Services section for an explanation of how to use this message.

This message does not return any information on groups with no link connection. The state field for LIST_ALL_CONNECTIONS should always be 3 (linked).

Note: The Connect Server **does not** perform endian conversion when this message is received between processes that run on systems that use different hardware data formats. The

sender program must convert the message to the endian format of the target system to ensure that the message data is correctly interpreted. This message is RISC aligned.

C Message Structure

```
typedef struct _GROUP_RECORD { int16 group_number;char group_name[4]; char
uic[3];char os_type; char node[6]; char state;char reserved[3];}
GROUP_RECORD;
```

Message Data Fields

Table 2-123 Message Data Fields

Field	Data Type	Script Format	Description
group_number	word	DW	Group address number.
group_name	4-char array	A(4)	Name truncated to 4 characters.
uic	3-char array	A(3)	Octal group user identification code
os_type	char	A(1)	Operating system type of group (OpenVMS only)‘.
node	6-char array	A(6)	Network node name.
state	char	A(1)	1=No link 2=Pending 3=Linked
reserved	3-char	ZB 3	Reserved for Oracle MessageQ.

Note: Treat variable "state" as int type.

Arguments

Table 2-124 Arguments

Argument	Script Format	pams_get_msg Format
Target	Supplied by Oracle MessageQ	Supplied by Oracle MessageQ

Table 2-124 Arguments

Source	CONNECT_SERVER	PAMS__CONNECT_SERVER
Class	PAMS	MSG_CLAS_PAMS
Type	LIST_ALL_CONNECTIONS	MSG_TYPE_LIST_ALL_ -CONNECTIONS

See Also

- LIST_ALL_CONNECTIONS request message
- LIST_ALL_ENTRIES (Request)
- LIST_ALL_ENTRIES (Response)
- LIST_ALL_GROUPS (Request)
- LIST_ALL_GROUPS (Response)

LIST_ALL_ENTRIES (Request)

An application can request a listing of all attached and configured queues in a group by sending a LIST_ALL_ENTRIES message to the Connect Server. The reply to this request is a variable-length message of the same type and class containing the queue information. See the Listing Cross-Group Connections, Entries, and Groups topic in the Using Message-Based Services section for an explanation of how to use this message.

Note: This message is RISC aligned.

C Message Structure

None.

Message Data Fields

None.

Arguments

Table 2-125 Arguments

Argument	Script Format	pams_get_msg Format
Target	CONNECT_SERVER	PAMS__CONNECT_SERVER
Source	Supplied by Oracle MessageQ	Supplied by Oracle MessageQ
Class	PAMS	MSG_CLAS_PAMS
Type	LIST_ALL_ENTRIES	MSG_TYPE_LIST_ALL_ ENTRIES

See Also

- LIST_ALL_ENTRIES response message
- LIST_ALL_CONNECTIONS (Request)
- LIST_ALL_CONNECTIONS (Response)
- LIST_ALL_GROUPS (Request)
- LIST_ALL_GROUPS (Response)

LIST_ALL_ENTRIES (Response)

An application can request a listing of all attached and configured queues in a group by sending a LIST_ALL_ENTRIES message to the Connect Server. The reply to this request is a variable length message of the same type and class containing the queue information. To read the information returned, the application must total the number of bytes in the reply and divide by the queue entry length, which is 24 bytes, to determine the number of records returned. See the Listing Cross-Group Connections, Entries, and Groups topic in the Using Message-Based Services section for an explanation of how to use this message.

Note: The Connect Server **does not** perform endian conversion when this message is received between processes that run on systems that use different hardware data formats. The

sender program must convert the message to the endian format of the target system to ensure that the message data is correctly interpreted. This message is RISC aligned.

C Message Structure

```
typedef struct _QLIST_RECORD { char q_name [20]; int16 q_number; char
attach_flag; char reserved;} QLIST_RECORD;
```

Message Data Fields

Table 2-126 Message Data Fields

Field	Data Type	Script Format	Description
q_name	20-char array	A(20)	Queue name, truncated to fit.
q_number	word	DW	Local queue address number.
attach_flag	Char	DB	1=Attached 0=Unattached
reserved	byte	ZB	Reserved for Oracle

Note: Treat variable " attach_flag" as int type.

Arguments

Table 2-127 Arguments

Argument	Script Format	pams_get_msg Format
Target	Requesting program	Requesting program
Source	CONNECT_SERVER	PAMS__CONNECT_SERVER
Class	PAMS	MSG_CLAS_PAMS
Type	LIST_ALL_ENTRIES	MSG_TYPE_LIST_ALL_ENTRIES

See Also

- LIST_ALL_ENTRIES request message
- LIST_ALL_GROUPS (Request)
- LIST_ALL_GROUPS (Response)
- LIST_ALL_CONNECTIONS (Request)
- LIST_ALL_CONNECTIONS (Response)

LIST_ALL_GROUPS (Request)

An application can request a listing of all groups on a message queuing bus by sending a LIST_ALL_GROUPS message to the Connect Server. The reply to this request is a variable-length message of the same type and class containing the group information. See the Listing Cross-Group Connections, Entries, and Groups topic in the Using Message- Based Services section for an explanation of how to use this message.

Note: This message is RISC aligned.

C Message Structure

None.

Message Data Fields

None.

Arguments

Table 2-128 Arguments

Argument	Script Format	pams_get_msg Format
Target	CONNECT_SERVER	PAMS__CONNECT_SERVER
Source	Supplied by Oracle MessageQ	Supplied by Oracle MessageQ
Class	PAMS	MSG_CLAS_PAMS
Type	LIST_ALL_GROUPS	MSG_TYPE_LIST_ALL_GROUPS

See Also

- LIST_ALL_GROUPS response message
- LIST_ALL_CONNECTIONS (Request)
- LIST_ALL_CONNECTIONS (Response)
- LIST_ALL_ENTRIES (Request)
- LIST_ALL_ENTRIES (Response)

LIST_ALL_GROUPS (Response)

An application can request a listing of all groups, connected and unconnected, on a message queuing bus by sending a LIST_ALL_GROUPS message to the Connect Server. The reply to this request is a variable-length message of the same type and class containing the group information. To read the information returned, the application must total the number of bytes in the reply and divide by the group entry length, which is 18 bytes, to determine the number of records returned. See the Listing Cross-Group Connections, Entries, and Groups topic in the Using Message-Based Services section for an explanation of how to use this message.

Note: The Connect Server **does not** perform endian conversion when this message is received between processes that run on systems that use different hardware data formats. The sender program must convert the message to the endian format of the target system to ensure that the message data is correctly interpreted. This message is RISC aligned.

C Message Structure

```
typedef struct _LIST_ALL_RESP { int16 group_number;

char   group_name [4];

char   uic_number [3]; charoperating_system; chardecnet_node [6]; char
connection_state; char reserved[3];

} LIST_ALL_RESP;
```

Message Data Fields

Table 2-129 Message Data Fields

Field	Data Type	Script Format	Description
group_number	word	DW	Group address number.
group_name	4-char array	A(4)	Name truncated to 4 characters.
uic_number	3-char array	A(3)	Octal group user identification code (UIC).
operating_system	char	A(1)	Operating system type of group.
decnet_node	6-char array	A(6)	Current DECnet node name. This can also be the TCP/IP node name. TCP/IP node names longer than 6 characters are truncated.
connection_state	char	A(1)	1=No link 2=Pending 3=Linked
reserved	3-char (VMS) 1-char (UNIX)	ZB	Reserved for Oracle MessageQ.

Arguments

Table 2-130 Arguments

Argument	Script Format	pams_get_msg Format
Target	Requesting program	Requesting program

Table 2-130 Arguments

Source	CONNECT_SERVER	PAMS__CONNECT_SERVER
Class	PAMS	MSG_CLAS_PAMS
Type	LIST_ALL_GROUPS	MSG_TYPE_LIST_ALL_GROUPS

See Also

- LIST_ALL_GROUPS request message
- LIST_ALL_CONNECTIONS (Request)
- LIST_ALL_CONNECTIONS (Response)
- LIST_ALL_ENTRIES (Request)
- LIST_ALL_ENTRIES (Response)

LIST_ALL_Q_REQ

The LIST_ALL_Q_REQ message is sent to the Queue Server to request a list of all attached permanent and temporary queues for a local or remote group. See the Listing Attached Queues in a Group topic in the Using Message-Based Services section for an explanation of how to use this message.

Note: **Note:** The Queue Server performs endian conversion when this message is sent between processes that run on systems that use different hardware data formats. This message is also RISC aligned.

C Message Structure

```
typedef struct _Q_NOTIFY_REQ { int32 version;
int32 user_tag;
} Q_NOTIFY_REQ;
```

Message Data Fields

Table 2-131 Message Data Fields

Field	Data Type	Script Format	Description
version	int32	DL	Version of request.
user_tag	int32	DL	User-specified code to identify this request.

Arguments

Table 2-132 Arguments

Argument	Script Format	pams_get_msg Format
Target	QUEUE_SERVER	PAMS__QUEUE_SERVER
Source	Requesting program	Requesting program
Class	PAMS	MSG_CLAS_PAMS
Type	LIST_ALL_Q_REQ	MSG_TYPE_LIST_ALL_Q_REQ

See Also

LIST_ALL_Q_RESP

LIST_ALL_Q_RESP

The LIST_ALL_Q_RESP message provides a list of all permanent queues and all attached temporary queues for a local or remote group. This information is requested by sending a LIST_ALL_Q_REQ message to the Queue Server. Because the response message may contain a long list of queue names, the application must allocate a sufficient buffer size to store the information returned. See Listing Attached Queues in a Group in Chapter 5, "Using Message-Based Services" for an explanation of how to use this message.

Note: The Queue Server performs endian conversion when this message is received between processes that run on systems that use different hardware data formats. This message is also RISC aligned.

C Message Structure

```
#define MAX_NUMBER_Q_RECS 50 typedef struct _Q_NOTIFY_RESP {

int32 version; int32 user_tag; int32 status_code;

int32 last_block_flag; int32 number_q_recs; struct{

q_address q_num; q_address q_owner; int32q_type;

int32  q_active_flag; int32q_attached_flag; int32q_owner_pid;

} q_rec [50];

} Q_NOTIFY_RESP;
```

Message Data Fields

Table 2-133 Message Data Fields

Field	Data Type	Script Format	Description
version	int32	DL	Version of response.
user_tag	int32	DL	User-specified code from request.
status_code	int32	DL	0=Error 1=Success -2=Refused
last_block_flag	int32	DL	Last block Boolean flag.
number_q_recs	int32	DL	Number of records in this message.
q_num	q_address	DL	Queue number.
q_owner	q_address	DL	Queue owner (only for secondary queues (SQs)).

Table 2-133 Message Data Fields

q_type	int32	DL	Queue type (numerically encoded P, S, M).
q_active_flag	int32	DL	Queue active Boolean flag.
q_attached_flag	int32	DL	Queue attached Boolean flag.
q_owner_pid	int32	DL	Queue owner process identification (PID). On Windows NT systems, thread identifier is returned.

Arguments

Table 2-134 Arguments

Argument	Script Format	pams_get_msg Format
Target	Requesting program	Requesting program
Source	QUEUE_SERVER	PAMS__QUEUE_SERVER
Class	PAMS	MSG_CLAS_PAMS
Type	LIST_ALL_Q_RESP	MSG_TYPE_LIST_ALL_Q_RESP

See Also

LIST_ALL_Q_REQ

LOCATE_Q_REP

The `pams_locate_q` function requests the queue address for a queue name. When this function is performed asynchronously, the results are returned in the LOCATE_Q_REP message. This message provides the location in the search list where the name is found, the status of the operation, a tag that can be set by the user, and the queue address associated with the name.

Note: This message is RISC aligned.

C Message Structure

```
typedef struct _LOCATE_Q_REP { int32 version;
int32 search_loc; q_address object_handle; int32 status;

int32 trans_id; char q_name [256];
} LOCATE_Q_REP;
```

Message Data Fields

Table 2-135 Message Data Fields

Field	Data Type	Script Format	Description
version	int32	DL	Format version number.
search_loc	int32	DL	Location in which name is found.
object_handle	q_address	DL	Queue address associated with name.
status	int32	DL	Return code from pams_locate_q.
trans_id	int32	DL	User-specified tag.
q_name	256-character array	A(256)	Name to locate.

Arguments

Table 2-136 Arguments

Argument	Script Format	pams_get_msg Format
Target	Supplied by Oracle MessageQ	Supplied by Oracle MessageQ

Table 2-136 Arguments

Source	Supplied by Oracle MessageQ	Supplied by Oracle MessageQ
Class	PAMS	MSG_CLAS_PAMS
Type	LOCATE_Q_REP	MSG_TYPE_LOCATE_Q_REP

MRS_ACK

The MRS_ACK message acknowledges the delivery of a recoverable message at the delivery interest point when a nonblocking request is issued. It responds to a pams_put_msg request when delivery modes of `PDEL_MODE_AK_DQF`, `PDEL_MODE_AK_SAF`, or `PDEL_MODE_AK_CONF` are specified. Status codes for the send operation are extracted from the PAMS Status Block (PSB), an argument value which is returned to the `pams_get_msg`, `pams_get_msga`, and `pams_get_msgw` function when the recoverable message is read. The status codes for the **`psb`** and **`uma`** arguments are listed in the Status Codes section of this description.

Note: This message is RISC aligned.

C Message Structure

None.

Message Data Fields

None.

Arguments

Table 2-137 Arguments

Argument	Script Format	pams_get_msg Format
Target	Sender program	Sender program
Source	MRS_SERVER	PAMS__MRS_SERVER

Table 2-137 Arguments

Class	MRS	MSG_CLAS_MRS
Type	MRS_ACK	MSG_TYPE_MRS_ACK

Status Code

Table 2-138 Status Code

Message	PSB Status
PAMS__DQF_DEVICE_FAIL	Message is not recoverable; destination queue file (DQF) I/O failed.
PAMS__ENQUEUED	Message is recoverable.
PAMS__MRS_RES_EXH	Message is not recoverable; MRS resource exhaustion.
PAMS__NO_DQF	Message is not recoverable; no DQF for target queue.
PAMS__NO_SAF	Message is not recoverable; no SAF file for target queue.
PAMS__SAF_DEVICE_FAIL	Message is not recoverable; SAF I/O failed.
PAMS__SAF_FORCED	Message is written to SAF file to maintain first-in/first-out (FIFO) order.
PAMS__SENDER_TMOEXPIRED	Send timeout expired prior to completion of MRS actions.
PAMS__STORED	Message is recoverable in store and forward (SAF) file. (Delivery mode was PDEL_MODE_AK_SAF.)

UMA Status

Table 2-139 UMA Status

Message	UMA Status
PAMS__DISC_SUCCESS	Message is not recoverable in DQF; UMA was PDEL_UMA_DISC; message discarded.
PAMS__DISC_FAILED	Message is not recoverable in DQF; UMA was PDEL_UMA_DISC; message could not be discarded.
PAMS__DISCL_SUCCESS	Message is not recoverable in DQF; UMA was PDEL_UMA_DISC; message discarded after logging recoverability failure.
PAMS__DISCL_FAILED	Message is not recoverable in DQF; UMA was PDEL_UMA_DISC; recoverability failure could not be logged or message could not be discarded.
PAMS__DLJ_SUCCESS	Message is not recoverable in DQF; UMA was PDEL_UMA_DLJ; message written to dead letter journal (DLJ).
PAMS__DLJ_FAILED	Message is not recoverable in DQF; UMA was PDEL_UMA_DLJ; dead letter journal write failed.
PAMS__DLQ_SUCCESS	Message is not recoverable in DQF; UMA was PDEL_UMA_DLQ; message queued to dead letter queue.
PAMS__DLQ_FAILED	Message is not recoverable in DQF; UMA was PDEL_UMA_DLQ; message could not be queued to dead letter queue.
PAMS__NO_UMA	Message is recoverable; undeliverable message action (UMA) not executed.
PAMS__RTS_SUCCESS	Message is not recoverable in DQF; UMA was PDEL_UMA_RTS; message returned to sender.
PAMS__RTS_FAILED	Message is not recoverable in DQF; UMA was PDEL_UMA_RTS; message could not be returned to sender.

Table 2-139 UMA Status

PAMS__SAF_SUCCESS	Message is not recoverable in DQF; UMA was PDEL_UMA_SAF; message recoverable from SAF file.
PAMS__SAF_FAILED	Message is not recoverable in DQF; UMA was PDEL_UMA_SAF; SAF write failed.

MRS_JRN_DISABLE

Disables journaling for a running message queuing group. This service is used to disable journaling before failing over auxiliary journals. See the Controlling Journaling to the PCJ File topic in the Using Message-Based Services section for an explanation of how to use this message. This service is available on OpenVMS systems only.

Note: The MRS Server **does not** perform endian conversion when this message is sent between processes that run on systems that use different hardware data formats. The sender program must convert the message to the endian format of the target system to ensure that the message data is correctly interpreted. This message is RISC aligned.

C Message Structure

```
/*****************************************/
/* STATUS VALUES FOR JRN_ENABLE message*/
/*****************************************/
#define JRN_SET_ERROR 0
#define JRN_SET_SUCCESS 1
#define JRN_SET_REFUSED 2
#define JRN_SET_ALREADY_DISABLED 3

#define JRN_SET_ALREADY_ENABLED 4
#define JRN_SET_SERVER_NOTUP 5

typedef struct _MRS_JRN_SET_ALL { int32 version;
```

```
int32 dqf_status; int32 saf_status; int32 pcj_status; int32 dlj_status;
} MRS_JRN_SET_ALL;
```

Message Data Fields

Table 2-140 Message Data Fields

Field	Data Type	Script Format	Description
version	int32	DL	Format version number. Must be 0.
dqf_status	int32	DL	0 = Error 1 = Success 2 = Refused 3 = Already Disabled
saf_status	int32	DL	0 = Error 1 = Success 2 = Refused 3 = Already Disabled
pcj_status	int32	DL	0 = Error 1 = Success 2 = Refused 3 = Already Disabled 5 = Server Not Available
dlj_status	int32	DL	0 = Error 1 = Success 2 = Refused 3 = Already Disabled 5 = Server Not Available

Arguments

Table 2-141 Arguments

Argument	Script Format	pams_get_msg Format
Target	MRS_SERVER	PAMS__MRS_SERVER
Source	Supplied by Oracle MessageQ	Supplied by Oracle MessageQ
Class	MRS	MSG_CLAS_MRS
Type	MRS_JRN_DISABLE	MSG_TYPE_MRS_JRN_DISABLE

See Also

- MRS_JRN_DISABLE_REP
- MRS_JRN_ENABLE
- MRS_JRN_ENABLE_REP

MRS_JRN_DISABLE_REP

Applications can request to disable journaling for a running message queuing group by sending an MRS_JRN_DISABLE message to the MRS Server. The MRS_JRN_DISABLE_REP message returns the status of the request. This service is used before failing over auxiliary journals. See the Controlling Journaling to the PCJ File topic in the Using Message- Based Services section for an explanation of how to use this message. This service is available on OpenVMS systems only.

Note: The MRS Server **does not** perform endian conversion when this message is received between processes that run on systems that use different hardware data formats. The sender program must convert the message to the endian format of the target system to ensure that the message data is correctly interpreted. This message is RISC aligned.

C Message Structure

```
/*****************************************/
/* STATUS VALUES FOR JRN_ENABLE message*/
/*****************************************/
```

```
#define JRN_SET_ERROR 0
#define JRN_SET_SUCCESS 1
#define JRN_SET_REFUSED 2
#define JRN_SET_ALREADY_DISABLED 3
#define JRN_SET_ALREADY_ENABLED 4
#define JRN_SET_SERVER_NOTUP 5

typedef struct _MRS_JRN_SET_ALL { int32 version;
int32 dqf_status; int32 saf_status; int32 pcj_status; int32 dlj_status;
} MRS_JRN_SET_ALL;
```

Message Data Fields

Table 2-142 Message Data Fields

Field	Data Type	Script Format	Description
version	int32	DL	Format version number. Must be 0.
dqf_status	int32	DL	0 = Error 1 = Success 2 = Refused 3 = Already Disabled
saf_status	int32	DL	0 = Error 1 = Success 2 = Refused 3 = Already Disabled
pcj_status	int32	DL	0 = Error 1 = Success 2 = Refused 3 = Already Disabled 5 = Server Not Available

Table 2-142 Message Data Fields

dlj_status	int32	DL	0 = Error 1 = Success 2 = Refused 3 = Already Disabled 5 = Server Not Available

Arguments

Table 2-143 Arguments

Argument	Script Format	pams_get_msg Format
Target	Requesting program	Requesting program
Source	MRS_SERVER	PAMS__MRS_SERVER
Class	MRS	MSG_CLAS_MRS
Type	MRS_JRN_DISABLE_REP	MSG_TYPE_MRS_JRN_ DISABLE_REP

See Also

- MRS_JRN_DISABLE
- MRS_JRN_ENABLE
- MRS_JRN_ENABLE_REP

MRS_JRN_ENABLE

Enables journaling for a running message queuing group after it has been disabled using the MRS_JRN_DISABLE message. This service is used before failing over auxiliary journals. See

the Controlling Journaling to the PCJ File topic in the Using Message-Based Services section for an explanation of how to use this message. This service is available on OpenVMS systems only.

Note: The MRS Server **does not** perform endian conversion when this message is sent between processes that run on systems that use different hardware data formats. The sender program must convert the message to the endian format of the target system to ensure that the message data is correctly interpreted. This message is RISC aligned.

C Message Structure

```
/****************************************/
/* STATUS VALUES FOR JRN_ENABLE message*/
/****************************************/
#define JRN_SET_ERROR 0
#define JRN_SET_SUCCESS 1
#define JRN_SET_REFUSED 2
#define JRN_SET_ALREADY_DISABLED 3
#define JRN_SET_ALREADY_ENABLED 4
#define JRN_SET_SERVER_NOTUP 5
typedef struct _MRS_JRN_SET_ALL { int32 version;
int32 dqf_status; int32 saf_status; int32 pcj_status; int32 dlj_status;
} MRS_JRN_SET_ALL;
```

Message Data Fields

Table 2-144 Message Data Fields

Field	Data Type	Script Format	Description
version	int32	DL	Format version number. Must be 0.

Table 2-144 Message Data Fields

dqf_status	int32	DL	0 = Error 1 = Success 2 = Refused 4 = Already Enabled
saf_status	int32	DL	0 = Error 1 = Success 2 = Refused 4 = Already Enabled
pcj_status	int32	DL	0 = Error 1 = Success 2 = Refused 4 = Already Enabled 5 = Server Not Available
dlj_status	int32	DL	0 = Error 1 = Success 2 = Refused 4 = Already Enabled 5 = Server Not Available

Arguments

Table 2-145 Arguments

Argument	Script Format	pams_get_msg Format
Target	MRS_SERVER	PAMS__MRS_SERVER
Source	Supplied by Oracle MessageQ	Supplied by Oracle MessageQ
Class	MRS	MSG_CLAS_MRS
Type	MRS_JRN_ENABLE	MSG_TYPE_MRS_JRN_ENABLE

See Also

- MRS_JRN_DISABLE
- MRS_JRN_DISABLE_REP
- MRS_JRN_ENABLE_REP

MRS_JRN_ENABLE_REP

Applications can request to reenable journaling for a running message queuing group after it has been disabled by sending an MRS_JRN_ENABLE message to the MRS Server. The MRS_JRN_ENABLE_REP message returns the status of the request. This service is used with MRS before failing over auxiliary journals. See the Controlling Journaling to the PCJ File topic in the Using Message-Based Services section for an explanation of how to use this message. This service is available on OpenVMS systems only.

Note: The MRS Server **does not** perform endian conversion when this message is received between processes that run on systems that use different hardware data formats. The sender program must convert the message to the endian format of the target system to ensure that the message data is correctly interpreted. This message is RISC aligned.

C Message Structure

```
/*****************************************/
/* STATUS VALUES FOR JRN_ENABLE message*/
/*****************************************/
#define JRN_SET_ERROR 0
#define JRN_SET_SUCCESS 1
#define JRN_SET_REFUSED 2
#define JRN_SET_ALREADY_DISABLED 3
#define JRN_SET_ALREADY_ENABLED 4
#define JRN_SET_SERVER_NOTUP 5

typedef struct _MRS_JRN_SET_ALL { int32 version;
int32 dqf_status; int32 saf_status; int32 pcj_status; int32 dlj_status;
};
```

```
typedef struct MRS_JRN_SET_ALL;
```

Message Data Fields

Table 2-146 Message Data Fields

Field	Data Type	Script Format	Description
version	int32	DL	Format version number. Must be 0.
dqf_status	int32	DL	0 = Error 1 = Success 2 = Refused 4 = Already Enabled
saf_status	int32	DL	0 = Error 1 = Success 2 = Refused 4 = Already Enabled
pcj_status	int32	DL	0 = Error 1 = Success 2 = Refused 4 = Already Enabled 5 = Server Not Available
dlj_status	int32	DL	0 = Error 1 = Success 2 = Refused 4 = Already Enabled 5 = Server Not Available

Arguments

Table 2-147 Arguments

Argument	Script Format	pams_get_msg Format
Target	Requesting program	Requesting program

Table 2-147 Arguments

Source	MRS_SERVER	PAMS__MRS_SERVER
Class	MRS	MSG_CLAS_MRS
Type	MRS_JRN_ENABLE_REP	MSG_TYPE_MRS_JRN_ENABLE_REP

See Also

- MRS_JRN_DISABLE
- MRS_JRN_DISABLE_REP
- MRS_JRN_ENABLE

Q_UPDATE

Applications can register to receive notification when queue states change in local or remote groups by sending an ENABLE_Q_NOTIFY_REQ message. The ENABLE_Q_NOTIFY_RESP message delivers a list of all active queues and then subsequently notifies the application of attachments, detachments, and changes to active and inactive status using the Q_UPDATE message. See the Receiving Attachment Notifications topic in the Using Message-Based Services section for an explanation of how to use this message.

Note: The Queue Server performs endian conversion when this message is received between processes that run on systems that use different hardware data formats. This message is also RISC aligned.

C Message Structure

```
#define MAX_NUMBER_Q_RECS 50 typedef struct _Q_NOTIFY_RESP {

int32 version; int32 user_tag; int32 status_code;

int32 last_block_flag; int32 number_q_recs; struct{

q_address q_num; q_address q_owner; int32q_type;

int32  q_active_flag; int32q_attached_flag; int32q_owner_pid;

} q_rec [50];

} Q_NOTIFY_RESP;
```

Message Data Fields

Table 2-148 Message Data Fields

Field	Data Type	Script Format	Description
version	int32	DL	Version of response.
user_tag	int32	DL	User-specified code from request.
status_code	int32	DL	0=Error 1=Success 2=Refused
last_block_flag	int32	DL	Last block Boolean flag.
number_q_recs	int32	DL	Number of records in this message.
q_num	q_address	DL	Queue number.
q_owner	q_address	DL	Queue owner (only for secondary queues (SQs)).
q_type	int32	DL	Queue type (numerically encoded P, S, M).
q_active_flag	int32	DL	Queue active Boolean flag.
q_attached_flag	int32	DL	Queue attached Boolean flag.
q_owner_pid	int32	DL	Queue owner process identification (PID).

Arguments

Table 2-149 Arguments

Argument	Script Format	pams_get_msg Format
Target	Requesting program	Requesting program

Table 2-149 Arguments

Source	QUEUE_SERVER	PAMS__QUEUE_SERVER
Class	PAMS	MSG_CLAS_PAMS
Type	Q_UPDATE	MSG_TYPE_Q_UPDATE

See Also

- ENABLE_Q_NOTIFY_REQ
- ENABLE_Q_NOTIFY_RESP
- DISABLE_Q_NOTIFY_REQ
- DISABLE_Q_NOTIFY_RESP

SBS_DEREGISTER_REQ

Requests SBS deregistration by exact match of MOT and distribution queue or by registration ID. This service replaces the SBS_DEREG service.

Note: The SBS Server performs endian conversion when this message is sent between processes that run on systems that use different hardware data formats. This message is also RISC aligned.

C Message Structure

```
typedef struct _SBS_DEREGISTER_REQ { int32 version;

int32 user_tag; int32 mot;

q_address distribution_q; int32 reg_id;

int32 req_ack;

} SBS_DEREGISTER_REQ;
```

Message Data Fields

Table 2-150 Message Data Fields

Field	Data Type	Script Format	Description
version	int32	DL	Message format version number. Must be 40.
user_tag	int32	DL	User-specified code to identify this request.
mot	int32	DL	The MOT broadcast stream from which the program wants to deregister. 0 if unused.
distribution_q	q_address	DW, DW	The Oracle MessageQ address of the distribution queue of the registration. A zero in the group number portion of the queue address automatically is replaced with the group
reg_id	int32	DL	The ID of the registration request to deregister. 0 if unused.
req_ack	int32	DL	1 if registration acknowledgment message is required; 0 otherwise.

Arguments

Table 2-151 Arguments

Argument	Script Format	pams_get_msg Format
Target	SBS_SERVER	PAMS__SBS_SERVER
Source	Source queue address of the requester.	Source queue address of the requester.
Class	PAMS	MSG_CLAS_PAMS
Type	SBS_DEREGISTER_REQ	MSG_TYPE_SBS_ DEREGISTER_REQ

See Also

- SBS_DEREGISTER_RESP
- SBS_REGISTER_REQ
- SBS_REGISTER_RESP

SBS_DEREGISTER_RESP

This response message acknowledges the SBS server deregistration of all entries matching the given MOT queue and distribution queue.

This service replaces the SBS_DEREG_ACK service.

Note: The SBS Server performs endian conversion when this message is received between processes that run on systems that use different hardware data formats. This message is also RISC aligned.

C Message Structure

```
typedef struct _SBS_DEREGISTER_RESP { int32 version;
int32 status; int32 user_tag; int32 number_reg;
} SBS_DEREGISTER_RESP;
```

Message Data Fields

Table 2-152 Message Data Fields

Field	Data Type	Script Format	Description
version	int32	DL	Message format version number. Must be 40.
status	int32	DL	Returned status code. Valid codes are as follows: PSYM_SBS_SUCCESS = Success PSYM_SBS_BADPARAM = Bad parameter PSYM_SBS_NOMATCH = No match
user_tag	int32	DL	User-specified code from the request message.
number_reg	int32	DL	The number of registrants left on this MOT or target.

Arguments

Table 2-153 Arguments

Argument	Script Format	pams_get_msg Format
Target	Requesting program	Requesting program
Source	SBS_SERVER	PAMS__SBS_SERVER
Class	PAMS	MSG_CLAS_PAMS
Type	SBS_DEREGISTER_RESP	MSG_TYPE_SBS_ DEREGISTER_RESP

See Also

- SBS_DEREGISTER_REQ
- SBS_REGISTER_REQ
- SBS_REGISTER_RESP

SBS_REGISTER_REQ

This request message requests registration for reception of broadcast messages. It can specify from 0 to 255 distribution rules, which must be satisfied for the message to be distributed to the distribution queue.

This service replaces the SBS_REG and SBS_REG_EZ services.

Note: The SBS Server performs endian conversion when this message is sent between processes that run on systems that use different hardware data formats. This message is also RISC aligned.

C Message Structure

```
typedef struct _SBS_REGISTER_HEAD { int32 version;

int32 user_tag; int32 mot;

q_address distribution_q; int32 req_ack;
```

```
int32 seq_gap_notify; int32 auto_dereg; int32 rule_count; int32
rule_conjunct;

} SBS_REGISTER_HEAD;

typedef struct _SBS_REGISTER_RULE { int32 offset;

int32 data_operator; int32 length;

int32 operand;

} SBS_REGISTER_RULE;

#define MAX_SEL_RULES 256

typedef struct _SBS_REGISTER_REQ { SBS_REGISTER_HEAD head;
SBS_REGISTER_RULE rule [256];

} SBS_REGISTER_REQ;
```

Message Data Fields

Table 2-154 Message Data Fields

Field	Data Type	Script Format	Description
version	int32	DL	Message format version number. Must be 40.
user_tag	int32	DL	User-specified code to identify this request.
mot	int32	DL	The MOT broadcast stream to which the program attempts to register.
distribution_q	q_address	DW, DW	The Oracle MessageQ address that receives any messages that are selected from the broadcast stream. A zero in the group number portion of the queue address is automatically replaced with
req_ack	int32	DL	1 if registration acknowledgment message is required; 0 otherwise.
seq_gap_notify	int32	DL	1 if broadcast stream sequence gap notification is required; 0 otherwise.

Table 2-154 Message Data Fields

auto_dereg	int32	DL	1 if registration request is to be purged on distribution queue detach; 0 otherwise.
rule_count	int32	DL	Number of distribution rules in the request (0, ... 255).
rule_conjunct	int32	DL	Valid values are: PSEL_ALL_RULES if all rules must be true for distribution to succeed;
* Following items are repeated "rule_count" times *			
data_offset	int32	DL	Valid values are: PSEL_TYPE PSEL_CLAS SDM tag ID Integer in the range 0, ..., MAX_MSG_SIZE, specifying an offset in the data
rule_count	int32	DL	Number of distribution rules in the request (0, ... 255).

Arguments

Table 2-155 Arguments

Argument	Script Format	pams_get_msg Format
Target	SBS_SERVER	PAMS__SBS_SERVER
Source	Requesting program	Requesting program
Class	PAMS	MSG_CLAS_PAMS
Type	SBS_REGISTER_REQ	MSG_TYPE_SBS_REGISTER_REQ

See Also

- SBS_DEREGISTER_REQ
- SBS_DEREGISTER_RESP
- SBS_REGISTER_RESP

SBS_REGISTER_RESP

This message provides a response to an SBS_REGISTER_REQ subscriber registration. The response contains a status field, which is 1 on success. The message also contains the user tag, specified in the request message, the registration ID and the number of registered entries for the MOT address.

This service replaces the SBS_REG_REPLY and SBS_REG_EZ_REPLY services.

Note: The SBS Server performs endian conversion when this message is received between processes that run on systems that use different hardware data formats. This message is also RISC aligned.

C Message Structure

```
typedef struct _SBS_REGISTER_RESP { int32 version;
int32 user_tag; int32 status; int32 reg_id; int32 number_reg;
} SBS_REGISTER_RESP;
```

Message Data Fields

Table 2-156 Message Data Fields

Field	Data Type	Script Format	Description
version	int32	DL	Message format version number. Must be 40.
user_tag	int32	DL	User-specified code from the request message.
status	int32	DL	Returned status code. Valid codes are as follows: PSYM_SBS_SUCCESS = Success PSYM_SBS_BADPARAM = Bad parameter PSYM_SBS_RESRCFAIL = Failed to allocate resource

Table 2-156 Message Data Fields

reg_id	int32	DL	Returned registration ID.
number_reg	int32	DL	Number of entries currently registered for this MOT or target.

Arguments

Table 2-157 Arguments

Argument	Script Format	pams_get_msg Format
Target	Source of registrant	Source of registrant
Source	SBS_SERVER	PAMS__SBS_SERVER
Class	PAMS	MSG_CLAS_PAMS
Type	SBS_SEQUENCE_RESP	MSG_TYPE_SBS_SEQUENCE_ RESP

See Also

- SBS_DEREGISTER_REQ
- SBS_DEREGISTER_RESP
- SBS_REGISTER_REQ

SBS_STATUS_REQ

The SBS server supports a message-based status request. This request details the current condition of each MOT being used by the server and its activity with other Oracle MessageQ groups, which are also running the SBS server.

The request message is targeted to the SBS_SERVER with message class PAMS and message type SBS_STATUS_REQ. Upon receipt of the message, the SBS server validates the request. If the request is incorrect, the response message contains an error status. The SBS server responds with the reply message of type SBS_STATUS_RESP.

Note: The SBS Server performs endian conversion when this message is sent between processes that run on systems that use different hardware data formats. This message is also RISC aligned.

C Message Structure

```
typedef struct _SBS_STATUS_REQ { int32 version;
int32 user_tag; int32 start_mot; int32 end_mot; int32 reset;
} SBS_STATUS_REQ;
```

Message Data Fields

Table 2-158 Message Data Fields

Field	Data Type	Script Format	Description
version	int32	DL	Message format version number. Must be 40.
user_tag	int32	DL	User-specified code to identify this request.
start_mot	int32	DL	Lowest MOT for which statistics are desired.
end_mot	int32	DL	Highest MOT for which statistics are desired.
reset	int32	DL	0: Do not reset counters for the remote server data after constructing the reply message. 1: Reset counters for the remote server data after constructing the reply message.

Arguments

Table 2-159 Arguments

Argument	Script Format	pams_get_msg Format
Target	SBS_SERVER	PAMS__SBS_SERVER

Table 2-159 Arguments

Source	Requesting program's primary or reply queue	Requesting program's primary or reply queue
Class	PAMS	MSG_CLAS_PAMS
Type	SBS_STATUS_REQ	MSG_TYPE_SBS_STATUS_REQ

See Also

- SBS_STATUS_RESP

SBS_STATUS_RESP

This message is returned following the successful processing of the SBS_STATUS_REQ request message. It is a variable format message and is made up of a variable number of fixed length parts. To parse the message, each variable length section has a count.

Note: The SBS Server performs endian conversion when this message is received between processes that run on systems that use different hardware data formats. This message is also RISC aligned.

C Message Structure

```
typedef struct _SBS_STATUS_RESP { int32 version;

int32 user_tag; int32 status; int32 num_rec; int32 last_block; char data
[31980];

} SBS_STATUS_RESP;

typedef struct _SBS_STATUS_RESP_MOT { int32 mot;

union  { struct{

union  { struct{

char s_b1; char s_b2; char s_b3; char s_b4;

} S_un_b; struct{

uint16 s_w1; uint16 s_w2;

} S_un_w; uint32 S_addr;

} inet_addr; uint16 inet_port;
```

```
} udp; struct{
char mca_addr [12];
char protocol [4];
} eth; struct {
char unused [20];
} dmq;
int32 filler [5];
} transport;
int32 heartbeat_timer; int32 xmit_silo;
int32 rcv_silo; int32 rcv_silo_max; int32 num_reg;
int32 complete_rcvd; int32 complete_bytes; int32 seq_gaps;
int32 whole_msg_gaps; int32 whole_silo_gap; struct  {
char device_name [16]; struct{
uint32 tv_sec; uint32 tv_usec;
} fail_tod; int32 msgs_sent; int32 bytes_sent; int32 pkts_sent; int32
pkts_rcvd;
int32 dupl_pkts_disc;
} rail [2];
}  SBS_STATUS_RESP_MOT;
typedef struct _SBS_STATUS_REP_REG_Q { q_address reg_q;
} SBS STATUS REP REG Q;
typedef struct _SBS_STATUS_REP_NUM_GROUPS { int32 num_groups;
}  SBS_STATUS_REP_NUM_GROUPS;
typedef struct _SBS_STATUS_RESP_GROUP { int32 group;
int32  rexmit_reqs_to_remote; int32  rexmit_sat_by_remote; int32
late_rexmit;
int32  rexmit_reqs_from_remote; int32  rexmit_sat_by_local;
}  SBS_STATUS_RESP_GROUP;
```

Message Data Fields

Table 2-160 Message Data Fields

Field	Data Type	Script Format	Description
version	int32	DL	Message format version number. Must be 40.
user_tag	int32	DL	User-specified code to identify this request.
Status	int32	DL	Returned status code. Valid codes are as follows: PSYM_SBS_SUCCESS = Success PSYM_SBS_BADPARAM = Bad parameter PSYM_SBS_NOMATCH = No match
num_rec	int32	DL	Number of MOTs reported in this message.
last_block	int32	DL	1 if this is the last message; 0 otherwise.
* Remainder of message repeated "num_rec" times up to a maximum of 50 records per Local SBS Server data *			
mot	int32	DL	MOT for which statistics are being reported.
transport		A(20)	Transport specific address information associated with the MOT. The format is dependant on the type of transport referred to.
heartbeat_timer	int32	DL	Heartbeat timer setting.
xmit_silo	int32	DL	Transmit silo size (MABs).
rcv_silo	int32	DL	Receiver silo size (MABs).
rcv_silo_max	int32	DL	Maximum occupancy of receive silo (MABs).
num_reg	int32	DL	Number of registrants for this MOT.

Table 2-160 Message Data Fields

complete_rcvd	int32	DL	Number of complete messages received.
complete_bytes	int32	DL	Number bytes contained in "complete_rcvd" messages.
seq_gaps	int32	DL	Total sequence gaps reported on this MOT.
whole_msg_gaps	int32	DL	Number complete messages detected missed initially.
whole_silo_gap	int32	DL	Number times sequence gap caused entire silo flush.
* Transport rail information repeated two times *			
device_name	char	A(16)	Optimized device address.
fail_tod		DL(2)	Shutdown timestamp in seconds.
msgs_sent	int32	DL	Number of messages sent on this rail.
bytes_sent	int32	DL	Number of bytes sent on this rail.
pkts_sent	int32	DL	Number of packets sent on this rail.
pkts_rcvd	int32	DL	Number of packets received on rail.
dupl_pkts_disc	int32	DL	Number of duplicate packets discarded from this rail.
* Registrant data: repeated "num_reg" times *			
reg_q	q_address	DW, DW	Queue address of registrant.
* End of registrant data *			
num_groups	int32	DL	Number of remote SBS servers communicating with the local SBS server.

Table 2-160 Message Data Fields

* Remote SBS server data: Following fields repeated "num_groups" times *			
group	int32	DL	Group number of remote SBS server.
rexmit_reqs_to_ remote	int32	DL	Number of retransmission requests from the local SBS server to the remote SBS server.
rexmit_sat_by_ remote	int32	DL	Number of retransmission requests satisfied by the remote SBS server.
late_rexmit	int32	DL	Number of retransmission requests that were received too late to prevent a sequence gap.
rexmit_reqs_ from_remote	int32	DL	Number of retransmission requests from the remote SBS server.
rexmit_sat_by _local	int32	DL	Number of retransmission requests satisfied by the local SBS server for the remote server.
* End of remote server data *			

Arguments

Table 2-161 Arguments

Argumentt	Script Format	pams_get_msg Format
Target	Requesting program's primary or reply queue	Requesting program's primary or reply queue
Source	SBS_SERVER	PAMS__SBS_SERVER
Type	SBS_STATUS_RESP	MSG_TYPE_SBS_STATUS_RESP

See Also

- SBS_STATUS_REQ

TIMER_EXPIRED

TIMER_EXPIRED is a response message to the pams_set_timer function. This message is sent to the timer queue associated with sender program's primary queue. Each call to the pams_set_timer function generates one message of type TIMER_EXPIRED when the timer expires.

Note: This message is RISC aligned.

C Message Structure

```
typedef struct _TIMER_EXPIRED { int32 timer_id;
char reserved [20];
} TIMER_EXPIRED;
```

Message Data Fields

Table 2-162 Message Data Fields

Field	Data Type	Script Format	Description
timer_id	int32	DL	Timer ID specified in the pams_set_timer call.
reserved	20-byte array	A(20)	Reserved for Oracle MessageQ.

Arguments

Table 2-163 Arguments

Argument	Script Format	pams_get_msg Format
Target	primary queue	primary queue
Source	TIMER_QUEUE	PAMS__TIMER_QUEUE

Table 2-163 Arguments

Class	PAMS	MSG_CLAS_PAMS
Type	TIMER_EXPIRED	MSG_TYPE_TIMER_EXPIRED

UNAVAIL

Applications register to receive notification when queues become active or inactive in local and remote groups by sending an AVAIL_REG message to the Avail Server. The UNAVAIL notification message is sent to the registered application when a queue in the selected group becomes inactive. See the Obtaining the Status of a Queue topic in the Using Message-Based Services section for an explanation of how to use this message.

Note: The Avail Server performs endian conversion when this message is received between processes that run on systems that use different hardware data formats. This message is also RISC aligned.

C Message Structure

```
typedef struct _UNAVAIL { q_address target_q;
} UNAVAIL;
```

Message Data Fields

Table 2-164 Message Data Fields

Field	Data Type	Script Format	Description
target_q	q_address	DL	Address of unavailable queue.

Arguments

Table 2-165 Arguments

Argument	Script Format	pams_get_msg Format
Target	Supplied by AVAIL_REG	Supplied by AVAIL_REG

Table 2-165 Arguments

Source	AVAIL_SERVER	AVAIL_SERVER
Class	PAMS	MSG_CLAS_PAMS
Type	UNAVAIL	MSG_TYPE_UNAVAIL

See Also

- AVAIL_REG
- AVAIL_REG_REPLY
- AVAIL
- AVAIL_DEREG