The Open Group
Request For Comments: 99.0
March 1997

J. Bowe (TOG)
J. Salamone (TOG)

# MESSAGE QUEUING FOR DCE

## 1. INTRODUCTION

Providing a means for applications to communicate is a fundamental service of any distributed computing infrastructure. One method, which is analogous to inter-process communication mechanisms typically available within a single system, is message exchange. A message is a unit of data that one application process may send to other application processes. The size, content and significance of a message is determined by the communicating application processes.

DCE has always offered Remote Procedure Call (RPC) based communication. Some DCE-detractors claim that message passing and/or queuing is superior. We feel this sometimes turns into a religious battle. Our view is that it depends on the application. Providing some message-oriented services will make DCE more attractive for some application writers.

RFC 95, “DCE/NEXT: REQUIREMENTS SUMMARY”, asks for "support for message queuing in DCE" and "support for messaging in DCE, including asynchronous RPC".

A prototype of this Message-Oriented Services package was shipped with DCE 1.2.2 in the unsupported area, ./src/nosupport/messaging in January 1997. Functionality includes most of the message queuing features outlined here.

### 1.1. Some Background on Messaging

Viewed from the highest level, there are three components of interest in the messaging environment: those that use the messaging service; those that the messaging service uses; and the messaging service itself. Each of these will be described in more detail in later sections.

There are primarily two types of message service users: senders and recipients of messages; and administrators. A sender is a process which calls upon the messaging service to communicate a message to other processes. A recipient is a process which calls upon the messaging service to retrieve messages that were sent by other processes. A process may be both a sender and a receiver. An administrator controls the configuration and operation of the messaging service. The services provided by the messaging system to support these usage types are each described separately.

The messaging system itself provides services through two interfaces, one for message communication and related functions, and one for administration. Internally, the system is logically partitioned into the components which support the interfaces local to an individual system, queue managers, and queues. There may be multiple instances of each of these components as described later.

The messaging system makes use of data transmission, marshaling, naming, and security services provided by the underlying computing environment, in this case, DCE.

The messaging service allows senders and receivers to execute concurrently and to send and receive messages asynchronously. Two forms of asynchronous message communication to consider are message queuing and message passing. The focus of this work is to build a message queuing system, therefore, message passing will only be mentioned to illustrate some of the differences between queuing versus passing. To adequately describe these forms of message communication, some of the components of the messaging system must first be described in more detail.

### 1.2. Document Structure

The remainder of this document is divided into four sections. Section 2 describes some terminology. Section 3 describes the major goals of the project. Section 4 describes the network data structures Section 5 describes the RPC interfaces. Section 6 describes the API. Section 7 describes the administrative interface.

## 2. TERMINOLOGY

A "queue" is a container for 0 or more messages.

"Message queuing" is the exchange of messages between a sender and recipient through a third party. Sender and recipient processes need not be active at the same time.

"Message passing" is the exchange of messages between a sender and recipient directly. Sender and recipient processes must not be active at the same time.

A "queue manager" is a DCE server that manages a collection of queues. Message queuing requires a queue manager.

"Persistence" is how a message is stored within a queue manager. "Volatile" means it is stored in memory; this means it may be lost when the qeueue manager process exits. "Persistent" means it is stored in a stable place, typically on disk; this means it is not lost when the qeueue manager process exits and is restarted. We will use the DCE backing store library for persistent storage.

## 3. GOALS AND NON-GOALS

The following are the goals of this project:

(a) Provide an infrastructure for secure message queuing between the queue manager and senders and recipients.

(b) Implement on top of commercial DCE releases (OSF release 1.1). This means the DCE source is not a requirement.

(c) Supply a set of APIs that let a user send an arbitrarily complex data structure to be to a destination.

The following are **not** goals:

(a) Provide message passing.

(b) Transaction capability.

### 3.1. System Features

The following is a list of queuing system features we will provide.

(a) Adheres to the DCE security model. In addition, the API provides an interface for an end-to-end protection level and a means to identify initial senders to recipients.

(b) An easy way to encode and decode (marshal) structured data as the message. This will be done with the existing RPC encoding services.

(c) Uses the DCE namespace to identify queue managers and queues.

(d) Automatic acknowledgement of message receipt (or dequeue)

(e) Support the following attributes on queues

  (i) queue name

  (ii) queue name aliases

  (iii) annotation

  (iv) creation timestamp

(v) maximum message size

(vi) maximum length of queue

(vii) enable and disable of enqueue and dequeue operations

(viii) current length

(ix) last activity timestamp

(x) idle timeout for empty queues

(xi) persistence

(f) Support the following attributes on messages

(i) unique message identifier (UUID)

(ii) enqueue timestamp

(iii) message type

(iv) data type (UUID)

(v) priority

(vi) message expiration

(vii) persistence

(viii) message sender (principal identity)

(ix) message delivery notice options

(x) end-to-end protection level (none, authentication, integrity, packet-level privacy)

(g) Dequeue filtering based on some of the above message attributes

## 4. DATA STRUCTURES

Included here are the "major" data structures, as defined in various idl files. Minor, supporting types and enumerations are not shown.

This is the attribute structure of an individual message.

```
typedef struct mos_mattr_s_t {
    uuid_t          id;             /* ID of this item in the queue */
    uuid_t          datatype;       /* manager that encoded it */
    mos_msg_attr_msgtype_t
    msgtype;        /* message type */
    sec_id_pa_t     sender;         /* sender identity */
    unsigned32      priority;       /* priority */
    unsigned32      flags;          /* flags */
    mos_msg_attr_persistence_t
    persistence;
    /* notice_dest, notice_sec may be null */
    unsigned32      notice_opts;    /* notice options */
    [ptr,string]
      char          *notice_dest;   /* name of ack/reply queue */
    [ptr,string]
      char          *notice_sec_name;
    /* sec group other qmgr must be in */
    unsigned32      protectlvl;     /* protection level */
    /* want a counter instead of time enqueued? */
    utc_t           time_enqueued;  /* when msg was enqueued by q-mgr */
```

```
        utc_t           expire_time;    /* when this msg will expire */
        utc_t           valid_time;     /* when this msg will become valid */
        uuid_t          qid;            /* ID of queue where this lives */
    } mos_mattr_t;
```

This is an array of bytes, which, as far as the messaging system is concerned, is the message payload or body.

```
    typedef struct {
       unsigned32           size;
       [ptr,size_is(size)]
       byte *               data;
    } mos_datatype_bytearray_t;
```

This is an item in the queue, composed of the message attributes and the body.

```
    typedef struct mos_qitem_s_t {
        mos_mattr_t                     mattr;
        mos_datatype_bytearray_t        body;
    } mos_qitem_t;
```

This is the attribute structure of a queue.

```
    typedef struct mos_qattr_s_t {
        uuid_t          id;             /* ID of the queue */
        [ptr,string]
          char          *name;          /* name of the queue */
        [ptr,string]
          char          *annotation;    /* annotation */
        mos_string_list_t
        aliases;        /* alias names of the queue */
        mos_que_attr_persistence_t
        persistence;
        boolean         enq_enabled;    /* enqueue: enable/disable */
        boolean         deq_enabled;    /* dequeue: enable/disable */
        unsigned32      flags;          /* flags */
        unsigned32      len;            /* current length of queue [ro] */
        unsigned32      max_msgsize;    /* max message size */
        unsigned32      max_qlen;       /* max length of queue */
        utc_t           created;        /* time created [ro] */
        utc_t           empty_timeout;  /* idle/empty time before removing */
        utc_t           last_activity;  /* time of last activity [ro] */
        uuid_t          acl;            /* ACL uuid - internal use [ro] */
    } mos_qattr_t;
```

## 5. QUEUE MANAGER RPC OPERATIONS

### 5.1. Management (Queue Manager) Operations

#### 5.1.1. mos_mgmt_create_queue()

Create a queue.

#### 5.1.2. mos_mgmt_get_id_list()

Return a list of the IDs of all queues.

#### 5.1.3. mos_mgmt_rename()

Rename a queue within queue manager.

#### 5.1.4. mos_mgmt_move()

Move messages from one queue to another, retaining all attributes. This is within the same queue manager.

### 5.2. Queue (Data) Operations

#### 5.2.1. mos_q_enqueue()

Enqueue a message in a given queue. The message is placed in the queue according to priority.

#### 5.2.2. mos_q_dequeue()

Dequeue a message (body and attributes) from a queue, optionally dequeuing it and optionally blocking until a message is available. Also optionally accepts a selection filter.

#### 5.2.3. mos_q_dequeue_next()

Simpler version mos_q_dequeue. Dequeues next available message and attributes from a queue.

#### 5.2.4. mos_q_peek()

Similar to mos_q_dequeue, but only a copy of the message attributes and body are returned; it is not dequeued.

#### 5.2.5. mos_q_purge()

Purge a message with a given ID.

#### 5.2.6. mos_q_retrieve_attr()

Retrieve message attributes from a queue, optionally blocking until a message is available. Also optionally accepts a selection filter. Similar to mos_q_dequeue, but a message body is not returned and the message is not dequeued.

#### 5.2.7. mos_q_retrieve_attr_by_id()

Retrieve message attributes from a queue, optionally blocking until a message is available. Also optionally accepts a selection filter. Similar to mos_q_dequeue, but a message body is not returned and the message is not dequeued.

#### 5.2.8. mos_q_ping()

Determine if a queue with a given name exists. Its ID is returned, since queues may have aliases, and UUIDs are unique. This is a simplified version of mos_q_retrieve_attr, but optimized for just determining if a queue is there. It is indended for the API's "attach" call.

#### 5.2.9. mos_q_get_id_list()

Return a list of IDs of messages on a queue that match the given selection filter.

#### 5.2.10. mos_q_get_id_list_all()

Return a list of IDs of all messages on a queue.

#### 5.2.11. mos_q_delete_queue()

Delete a queue.

#### 5.2.12. mos_q_get_attr_by_name() and mos_q_get_attr_by_id()

Return attributes of the queue described by the given name or UUID.

### 5.2.13. mos_q_set_attr_by_name() and mos_q_set_attr_by_id()

Set attributes of the queue described by the given name or UUID.

## 6. APPLICATION PROGRAMMING INTERFACE (API)

The API is broken into these categories:

(a) API Control

(b) Naming

(c) Conversation Management

(d) Security

(e) Marshalling

(f) Message Attribute Utilities

(g) Message Selection Filter Utilities

(h) Queue Attribute Utilities

(i) Queue Management Utilities

When a full DCE name is not specified for a queue in the various APIs, the API will choose a default queue manager, based on the following mechanisms. The search order is:

(a) Determine if caller has previously obtained or set a default queue manager via the API with mos_rsrc_dflt_quemgr_get() or mos_rsrc_dflt_quemgr_set(), respectively.

(b) Environment variable `MOS_DFLT_QUEMGR_NAME`.

(c) Per-host profile or hostdata object: `/.:/hosts/$HOSTNAME/profile`.

(d) Per-cell profile: `/.:/cell-profile`.

What follow are C function prototypes from our in-progress implementation (mosif.h). This describes the APIs and their parameters. There is a separate, more-detailed specification of the API.

### 6.1. API Control

Client interfaces for API control to support the Message Oriented Service Application Programming Interface(MOS-API).

### 6.1.1. mos_done

Releases all resources used by a MOS application.

```
void
mos_done(
    /* [out] */          error_status_t *        status
);
```

### 6.2. Naming

Client interfaces for Naming to support the Message Oriented Service Application Programming Interface(MOS-API).

### 6.2.1. mos_rsrc_dflt_quemgr_get

Returns the default queue manager used by a MOS application.

```
void
mos_rsrc_dflt_quemgr_get(
    /* [in] */          sec_rgy_name_t          security_name,
    /* [out] */         mos_rsrc_name_t         name,
```

```
    /* [out] */       mos_handle_t *         resource_handle,
    /* [out] */       error_status_t *       status
);
```

### 6.2.2. mos_rsrc_dflt_quemgr_set

Sets the default queue manager used by a MOS application.

```
void
mos_rsrc_dflt_quemgr_set(
    /* [in] */        mos_rsrc_name_t        name,
    /* [in] */        sec_rgy_name_t         security_name,
    /* [out] */       mos_handle_t *         resource_handle,
    /* [out] */       error_status_t *       status
);
```

### 6.2.3. mos_rsrc_attach

Finds one existing resource by name in the namespace. A resource is a name in the namespace that may be a queue, a Q-mgr, a destination peer application, a specific NSI entry, group, or profile name.

```
void
mos_rsrc_attach(
    /* [in] */        mos_rsrc_name_t        name,
    /* [in] */        sec_rgy_name_t         security_name,
    /* [out] */       mos_handle_t *         resource_handle,
    /* [out] */       error_status_t *       status
);
```

### 6.2.4. mos_rsrc_detach

Releases one resource and any allocated memory within MOS associated with the named resource. It is the responsibility of the caller to release any resources obtained via mos_rsrc_attach(), mos_rsrc_dflt_quemgr_get(), mos_rsrc_dflt_quemgr_set(), mos_que_mgmt_create(), or mos_msg_attr_notice_set().

```
void
mos_rsrc_detach(
    /* [in, out] */   mos_handle_t *         resource_handle,
    /* [out] */       error_status_t *       status
);
```

### 6.2.5. mos_rsrc_name_get

Returns the queue manager name, its security group name, its principal name, and if applicable the queue name itself for the specified resource handle.

```
void
mos_rsrc_name_get(
    /* [in] */        mos_handle_t           resource_handle,
    /* [out] */       sec_rgy_name_t         security_name,
    /* [out] */       sec_rgy_name_t         quemgr_prin_name,
    /* [out] */       mos_rsrc_name_t        quemgr_name,
    /* [out] */       mos_rsrc_name_t        que_name,
    /* [out] */       error_status_t *       status
);
```

### 6.3. Conversation Management

Client interfaces for Conversation Management to support the Message Oriented Service Application Programming Interface(MOS-API).

### 6.3.1. mos_msg_enqueue

Enqueue one message to a queue.

```
void
mos_msg_enqueue(
    /* [in] */        mos_handle_t            resource_handle,
    /* [in] */        unsigned32              flags,
    /* [in] */        mos_msg_attr_t          msg_attributes,
    /* [in] */        void *                  msg_buffer,
    /* [out] */       uuid_t *                msg_id,
    /* [out] */       error_status_t *        status
);
```

### 6.3.2. mos_msg_dequeue

Dequeue one message from a queue.

```
void
mos_msg_dequeue(
    /* [in] */     mos_handle_t                resource_handle,
    /* [in] */     unsigned32                  flags,
    /* [in] */     mos_msg_selfilter_t         msg_selection_filter,
    /* [out] */    mos_msg_attr_t *            msg_attributes,
    /* [out] */    mos_msg_buf_t *             msg_buffer,
    /* [out] */    error_status_t *            status
);
```

### 6.3.3. mos_msg_peek

Returns a copy of one message from a queue.

```
void
mos_msg_peek(
    /* [in] */     mos_handle_t                resource_handle,
    /* [in] */     unsigned32                  flags,
    /* [in] */     mos_msg_selfilter_t         msg_selection_filter,
    /* [out] */    mos_msg_attr_t *            msg_attributes,
    /* [out] */    mos_msg_buf_t *             msg_buffer,
    /* [out] */    error_status_t *            status
);
```

### 6.3.4. mos_msg_delete

Deletes one message by message ID from a queue.

```
void
mos_msg_delete(
    /* [in] */        mos_handle_t            resource_handle,
    /* [in] */        uuid_t *                msg_id,
    /* [out] */       error_status_t *        status
);
```

### 6.3.5. mos_msg_move

Move one message by message ID from one queue to another.

```
void
mos_msg_move(
    /* [in] */        mos_handle_t            src_resource_handle,
    /* [in] */        mos_handle_t            dest_resource_handle,
```

```
    /* [in] */          uuid_t *                 msg_id,
    /* [out] */         error_status_t *         status
);
```

### 6.4. Message Attribute Utilities

Client interfaces for Message Attribute Utilities to support the Message Oriented Service Application Programming Interface(MOS-API).

#### 6.4.1. mos_msg_attr_alloc

Allocates memory within MOS and returns an opaque pointer to a message attributes structure with defaults set.

```
void
mos_msg_attr_alloc(
    /* [out] */         mos_msg_attr_t *         msg_attributes,
    /* [out] */         error_status_t *         status
);
```

#### 6.4.2. mos_msg_attr_peek

Returns an opaque pointer to the message attributes on an existing message in the specified queue. Applies to Message Queuing model only.

```
void
mos_msg_attr_peek(
    /* [in] */          mos_handle_t             resource_handle,
    /* [in] */          unsigned32               flags,
    /* [in] */          mos_msg_selfilter_t      msg_selection_filter,
    /* [out] */         mos_msg_attr_t *         msg_attributes,
    /* [out] */         error_status_t *         status
);
```

#### 6.4.3. mos_msg_attr_free

Releases memory used for message attributes. It is the responsibility of the caller to release the memory if allocated by mos_msg_attr_alloc() or mos_msg_attr_peek().

```
void
mos_msg_attr_free(
    /* [in,out] */      mos_msg_attr_t *         msg_attributes,
    /* [out] */         error_status_t *         status
);
```

#### 6.4.4. mos_msg_attr_msgid_get

Return message ID from a message attributes structure.

```
void
mos_msg_attr_msgid_get(
    /* [in] */          mos_msg_attr_t            msg_attributes,
    /* [out] */         uuid_t *                  msg_id,
    /* [out] */         error_status_t *          status
);
```

#### 6.4.5. mos_msg_attr_enqtime_get

Return message enqueue time stamp in absolute time from a message attributes structure.

```
void
mos_msg_attr_enqtime_get(
    /* [in] */          mos_msg_attr_t           msg_attributes,
```

```
    /* [out] */         utc_t *                 msg_enq_time,
    /* [out] */         error_status_t *        status
);
```

### 6.4.6. mos_msg_attr_msgtype_get

Return message type from a message attributes structure.

```
void
mos_msg_attr_msgtype_get(
    /* [in] */          mos_msg_attr_t          msg_attributes,
    /* [out] */         mos_msg_attr_msgtype_t * msg_type,
    /* [out] */         error_status_t *        status
);
```

### 6.4.7. mos_msg_attr_msgtype_set

Set message type in a message attributes structure.

```
void
mos_msg_attr_msgtype_set(
    /* [in] */          mos_msg_attr_msgtype_t  msg_type,
    /* [in,out] */      mos_msg_attr_t          msg_attributes,
    /* [out] */         error_status_t *        status
);
```

### 6.4.8. mos_msg_attr_priority_get

Return priority from a message attributes structure.

```
void
mos_msg_attr_priority_get(
    /* [in] */          mos_msg_attr_t          msg_attributes,
    /* [out] */         unsigned32 *            priority,
    /* [out] */         error_status_t *        status
);
```

### 6.4.9. mos_msg_attr_priority_set

Set priority in a message attributes structure.

```
void
mos_msg_attr_priority_set(
    /* [in] */          unsigned32              priority,
    /* [in,out] */      mos_msg_attr_t          msg_attributes,
    /* [out] */         error_status_t *        status
);
```

### 6.4.10. mos_msg_attr_ttl_get

Return message expiration(time-to-live) in absolute time from a message attributes structure.

```
void
mos_msg_attr_ttl_get(
    /* [in] */          mos_msg_attr_t          msg_attributes,
    /* [out] */         utc_t *                 ttl,
    /* [out] */         error_status_t *        status
);
```

### 6.4.11. mos_msg_attr_ttl_set

Set message expiration(time-to-live) in absolute time in a message attributes structure.

```
void
mos_msg_attr_ttl_set(
    /* [in] */         utc_t *               ttl,
    /* [in,out] */     mos_msg_attr_t        msg_attributes,
    /* [out] */        error_status_t *      status
);
```

### 6.4.12. mos_msg_attr_ttr_get

Return time-to-receive in absolute time from a message attributes structure.

```
void
mos_msg_attr_ttr_get(
    /* [in] */         mos_msg_attr_t        msg_attributes,
    /* [out] */        utc_t *               ttr,
    /* [out] */        error_status_t *      status
);
```

### 6.4.13. mos_msg_attr_ttr_set

Set time-to-receive in absolute time in a message attributes structure.

```
void
mos_msg_attr_ttr_set(
    /* [in] */         utc_t *               ttr,
    /* [in,out] */     mos_msg_attr_t        msg_attributes,
    /* [out] */        error_status_t *      status
);
```

### 6.4.14. mos_msg_attr_persistence_get

Return persistence from a message attributes structure.

```
void
mos_msg_attr_persistence_get(
    /* [in] */         mos_msg_attr_t                  msg_attributes,
    /* [out] */        mos_msg_attr_persistence_t *    persistence,
    /* [out] */        error_status_t *                status
);
```

### 6.4.15. mos_msg_attr_persistence_set

Set persistence in a message attributes structure.

```
void
mos_msg_attr_persistence_set(
    /* [in] */         mos_msg_attr_persistence_t      persistence,
    /* [in,out] */     mos_msg_attr_t                  msg_attributes,
    /* [out] */        error_status_t *                status
);
```

### 6.4.16. mos_msg_attr_notice_get

Return notice options from a message attributes structure.

```
void
mos_msg_attr_notice_get(
    /* [in] */         mos_msg_attr_t        msg_attributes,
    /* [out] */        unsigned32 *          notice_flags,
```

```
    /* [out] */        mos_rsrc_name_t         notice_destination,
    /* [out] */        sec_rgy_name_t          notice_security_name,
    /* [out] */        error_status_t *        status
);
```

#### 6.4.17. mos_msg_attr_notice_set

Set notice options in a message attributes structure and return a handle to the notice destination.

```
void
mos_msg_attr_notice_set(
    /* [in] */         unsigned32              notice_flags,
    /* [in] */         mos_rsrc_name_t         notice_destination,
    /* [in] */         sec_rgy_name_t          notice_security_name,
    /* [in,out] */     mos_msg_attr_t          msg_attributes,
    /* [out] */        mos_handle_t *          resource_handle,
    /* [out] */        error_status_t *        status
);
```

### 6.5. Security

Client interfaces for Security to support the Message Oriented Service Application Programming Interface(MOS-API).

#### 6.5.1. mos_msg_attr_protectlvl_get

Return the end-to-end protection level of initial sender from the message attribute structure. Applies to Message Queuing model only.

```
void
mos_msg_attr_protectlvl_get(
    /* [in] */         mos_msg_attr_t          msg_attributes,
    /* [out] */        unsigned32 *            protect_level,
    /* [out] */        error_status_t *        status
);
```

#### 6.5.2. mos_msg_attr_protectlvl_set

Sets the end-to-end protection level of initial sender from the message attribute structure. Applies to Message Queuing model only.

```
void
mos_msg_attr_protectlvl_set(
    /* [in] */         unsigned32              protect_level,
    /* [in,out] */     mos_msg_attr_t          msg_attributes,
    /* [out] */        error_status_t *        status
);
```

#### 6.5.3. mos_msg_attr_secid_get

Return the security identity of initial sender of a message from the message attribute structure. Applies to Message Queuing model only.

```
void
mos_msg_attr_secid_get(
    /* [in] */         mos_msg_attr_t          msg_attributes,
    /* [out] */        sec_id_pa_t *           sec_id,
    /* [out] */        error_status_t *        status
);
```

### 6.5.4. mos_rsrc_protectlvl_get

Returns the protection level for communications made to a resource. This is not to be confused with end-to-end protection levels.

```
void
mos_rsrc_protectlvl_get(
    /* [in] */        mos_handle_t            resource_handle,
    /* [out] */       unsigned32 *            protect_level,
    /* [out] */       error_status_t *        status
);
```

### 6.5.5. mos_rsrc_protectlvl_set

Sets and registers the protection level for communications made to a resource. This is not to be confused with end-to-end protection levels.

```
void
mos_rsrc_protectlvl_set(
    /* [in] */        unsigned32              protect_level,
    /* [in/out] */    mos_handle_t            resource_handle,
    /* [out] */       error_status_t *        status
);
```

### 6.6. Marshalling

Client interfaces for Marshalling to support the Message Oriented Service Application Programming Interface (MOS-API).

This is the datatype passed to mos_api_datatype_register(). It is meant for the idl-generated encoding/decoding function.

```
typedef void (*mos_msg_attr_convfn_t)(
    /* [in] */         idl_es_handle_t        handle,
    /* [in,out] */     void *                 data,
    /* [in,out] */     error_status_t *       status
);
```

### 6.6.1. mos_msg_attr_datatype_get

Return application-specific data type from message attributes structure.

```
void
mos_msg_attr_datatype_get(
    /* [in] */         mos_msg_attr_t           msg_attributes,
    /* [out] */        uuid_t *                 datatype_id,
    /* [out] */        error_status_t *         status
);
```

### 6.6.2. mos_msg_attr_datatype_set

Set application-specific data type from message attributes structure.

```
void
mos_msg_attr_datatype_set(
    /* [in] */         uuid_t *                 datatype_id,
    /* [in,out] */     mos_msg_attr_t           msg_attributes,
    /* [out] */        error_status_t *         status
);
```

### 6.6.3. mos_api_datatype_register

Register one application-specific data type and its associated conversion function with MOS to encode/decode the data.

```
void
mos_api_datatype_register(
    /* [in] */          uuid_t *                        datatype_id,
    /* [in] */          mos_msg_attr_convfn_t           conv_fn,
    /* [out] */         error_status_t *                status
);
```

### 6.6.4. mos_api_datatype_unregister

Unregister one application-specific data type and its associated conversion function with MOS.

```
void
mos_api_datatype_unregister(
    /* [in] */          uuid_t *                        datatype_id,
    /* [out] */         error_status_t *                status
);
```

### 6.6.5. mos_msg_datatype_decode

Decodes the message buffer returned by mos_msg_dequeue() or mos_msg_peek().

```
void
mos_msg_datatype_decode(
    /* [in] */          uuid_t *                        datatype_id,
    /* [in] */          mos_msg_buf_t                   msg_buffer,
    /* [out] */         void *                          decoded_msg_buffer,
    /* [out] */         error_status_t *                status
);
```

### 6.6.6. mos_msg_buf_free

Releases memory used for the message buffer. It is the responsibility of the caller to release the memory if allocated by mos_msg_dequeue() or mos_msg_peek().

```
void
mos_msg_buf_free(
    /* [in,out] */      mos_msg_buf_t *                 msg_buffer,
    /* [out] */         error_status_t *                status
);
```

### 6.7. Message Selection Filter Utilities

Client interfaces for Message Selection Filter Utilities to support the Message Oriented Service Application Programming Interface(MOS-API).

### 6.7.1. mos_msg_selfilter_alloc

Allocates memory within MOS and returns an opaque pointer to a selection filter structure. The selection filter is comprised of one or more masks that follow the rules of the logical AND operator among all specified criteria when retrieving messages.

```
void
mos_msg_selfilter_alloc(
    /* [out] */          mos_msg_selfilter_t *   selection_filter,
    /* [out] */          error_status_t *        status
);
```

### 6.7.2. mos_msg_selfilter_free

Releases memory used for the selection filter when retrieving messages. It is the responsibility of the caller to release the memory if allocated by mos_msg_selfilter_alloc().

```
void
mos_msg_selfilter_free(
    /* [in,out] */      mos_msg_selfilter_t *   selection_filter,
    /* [out] */         error_status_t *        status
);
```

### 6.7.3. mos_msg_selmask_add

Adds one mask to the list of masks in the selection filter structure. A mask is comprised of a selection criteria type, a selection criteria value, and the relational operator to be performed.

```
void
mos_msg_selmask_add(
    /* [in] */          mos_msg_selmask_type_t   mask_type,
    /* [in] */          mos_msg_selmask_value_t  mask_value,
    /* [in] */          mos_msg_selmask_op_t     mask_op,
    /* [in,out] */      mos_msg_selfilter_t      selection_filter,
    /* [out] */         error_status_t *         status
);
```

## 6.8. Queue Attribute Utilities

Client interfaces for Queue Attribute Utilities to support the Message Oriented Service Application Programming Interface(MOS-API).

### 6.8.1. mos_que_attr_alloc

Allocates memory within MOS and returns an opaque pointer to a queue attributes structure with defaults set.

```
void
mos_que_attr_alloc(
    /* [out] */         mos_que_attr_t *         que_attributes,
    /* [out] */         error_status_t *         status
);
```

### 6.8.2. mos_que_attr_peek

Returns an opaque pointer to the queue attributes on an existing queue.

```
void
mos_que_attr_peek(
    /* [in] */          mos_handle_t             resource_handle,
    /* [out] */         mos_que_attr_t *         que_attributes,
    /* [out] */         error_status_t *         status
);
```

### 6.8.3. mos_que_attr_free

Releases memory used for queue attributes. It is the responsibility of the caller to release the memory if allocated by mos_que_attr_alloc() or mos_que_attr_peek().

```
void
mos_que_attr_free(
    /* [in,out] */      mos_que_attr_t *         que_attributes,
    /* [out] */         error_status_t *         status
);
```

Commit queue attribute modifications on an existing queue.

```
void
mos_que_attr_commit(
    /* [in] */         mos_handle_t             resource_handle,
    /* [in] */         mos_que_attr_t           queue_attributes,
    /* [out] */        error_status_t *         status
);
```

### 6.8.4. mos_que_attr_queid_get

Return queue ID from queue attributes structure.

```
void
mos_que_attr_queid_get(
    /* [in] */         mos_que_attr_t           que_attributes,
    /* [out] */        uuid_t *                 que_id,
    /* [out] */        error_status_t *         status
);
```

### 6.8.5. mos_que_attr_quecursize_get

Return current queue size from queue attributes structure.

```
void
mos_que_attr_quecursize_get(
    /* [in] */         mos_que_attr_t           que_attributes,
    /* [out] */        unsigned32 *             que_cur_size,
    /* [out] */        error_status_t *         status
);
```

### 6.8.6. mos_que_attr_createtime_get

Return queue creation time stamp in absolute time from queue attributes structure.

```
void
mos_que_attr_createtime_get(
    /* [in] */         mos_que_attr_t           que_attributes,
    /* [out] */        utc_t *                  que_create_time,
    /* [out] */        error_status_t *         status
);
```

### 6.8.7. mos_que_attr_activetime_get

Return queue last activity time stamp in absolute time from queue attributes structure.

```
void
mos_que_attr_activetime_get(
    /* [in] */         mos_que_attr_t           que_attributes,
    /* [out] */        utc_t *                  que_active_time,
    /* [out] */        error_status_t *         status
);
```

### 6.8.8. mos_que_attr_alias_list

Return the list of queue aliases by name from queue attributes structure. The first entry in the list is the queue name itself.

```
void
mos_que_attr_alias_list(
    /* [in] */         mos_que_attr_t            que_attributes,
    /* [in] */         unsigned32                space_avail,
```

```
        /* [out] */         unsigned32 *              num_returned,
        /* [out, size_is(space_avail), length_is(*num_returned)] */
                            mos_rsrc_name_t           que_alias_list[],
        /* [out] */         unsigned32 *              num_left,
        /* [out] */         error_status_t *          status
    );
```

### 6.8.9. mos_que_attr_alias_add

Adds one queue alias to the list of queue aliases in the queue attributes structure.

```
    void
    mos_que_attr_alias_add(
        /* [in] */          mos_rsrc_name_t           link_name,
        /* [in] */          mos_rsrc_name_t           que_alias_name,
        /* [in,out] */      mos_que_attr_t            que_attributes,
        /* [out] */         error_status_t *          status
    );
```

### 6.8.10. mos_que_attr_alias_remove

Removes one queue alias from the list of queue aliases in the queue attributes structure.

```
    void
    mos_que_attr_alias_remove(
        /* [in] */          mos_rsrc_name_t           link_name,
        /* [in] */          mos_rsrc_name_t           que_alias_name,
        /* [in,out] */      mos_que_attr_t            que_attributes,
        /* [out] */         error_status_t *          status
    );
```

### 6.8.11. mos_que_attr_alias_reset

Sets the queue alias list to NULL in the queue attributes structure.

```
    void
    mos_que_attr_alias_reset(
        /* [in,out] */      mos_que_attr_t            que_attributes,
        /* [out] */         error_status_t *          status
    );
```

### 6.8.12. mos_que_attr_annotation_get

Return queue annotation from queue attributes structure.

```
    void
    mos_que_attr_annotation_get(
        /* [in] */         mos_que_attr_t             que_attributes,
        /* [out] */        mos_que_attr_annotation_t  annotation,
        /* [out] */        error_status_t *           status
    );
```

### 6.8.13. mos_que_attr_annotation_set

Set queue annotation in queue attributes structure.

```
    void
    mos_que_attr_annotation_set(
        /* [in] */          mos_que_attr_annotation_t annotation,
        /* [in,out] */      mos_que_attr_t            que_attributes,
        /* [out] */         error_status_t *          status
    );
```

### 6.8.14. mos_que_attr_quemaxsize_get

Return maximum queue size from queue attributes structure.

```
void
mos_que_attr_quemaxsize_get(
    /* [in] */          mos_que_attr_t              que_attributes,
    /* [out] */         unsigned32 *                que_maxsize,
    /* [out] */         error_status_t *            status
);
```

### 6.8.15. mos_que_attr_quemaxsize_set

Set maximum queue size in queue attributes structure.

```
void
mos_que_attr_quemaxsize_set(
    /* [in] */          unsigned32                  que_maxsize,
    /* [in,out] */      mos_que_attr_t              que_attributes,
    /* [out] */         error_status_t *            status
);
```

### 6.8.16. mos_que_attr_msgmaxsize_get

Return maximum message size on queue from queue attributes structure.

```
void
mos_que_attr_msgmaxsize_get(
    /* [in] */          mos_que_attr_t              que_attributes,
    /* [out] */         unsigned32 *                msg_maxsize,
    /* [out] */         error_status_t *            status
);
```

### 6.8.17. mos_que_attr_msgmaxsize_set

Set maximum message size on queue in queue attributes structure.

```
void
mos_que_attr_msgmaxsize_set(
    /* [in] */          unsigned32                  msg_maxsize,
    /* [in,out] */      mos_que_attr_t              que_attributes,
    /* [out] */         error_status_t *            status
);
```

### 6.8.18. mos_que_attr_persistence_get

Return queue persistence from queue attributes structure.

```
void
mos_que_attr_persistence_get(
    /* [in] */          mos_que_attr_t                      que_attributes,
    /* [out] */         mos_que_attr_persistence_t *        persistence,
    /* [out] */         error_status_t *                    status
);
```

### 6.8.19. mos_que_attr_persistence_set

Set queue persistence in queue attributes structure.

```
void
mos_que_attr_persistence_set(
    /* [in] */          mos_que_attr_persistence_t          persistence,
    /* [in,out] */      mos_que_attr_t                      que_attributes,
```

```
        /* [out] */         error_status_t *              status
    );
```

### 6.8.20. mos_que_attr_idletimeout_get

Return queue idle timeout in relative time from queue attributes structure.

```
    void
    mos_que_attr_idletimeout_get(
        /* [in] */          mos_que_attr_t            que_attributes,
        /* [out] */         utc_t *                   que_idle_timeout,
        /* [out] */         error_status_t *          status
    );
```

### 6.8.21. mos_que_attr_idletimeout_set

Set queue idle timeout in relative time in queue attributes structure.

```
    void
    mos_que_attr_idletimeout_set(
        /* [in] */          utc_t *                   que_idle_timeout,
        /* [in,out] */      mos_que_attr_t            que_attributes,
        /* [out] */         error_status_t *          status
    );
```

### 6.8.22. mos_que_attr_allowenq_get

Return boolean that describes whether enqueuing is allowed or not from queue attributes structure.

```
    void
    mos_que_attr_allowenq_get(
        /* [in] */          mos_que_attr_t            que_attributes,
        /* [out] */         boolean *                 allow_enq,
        /* [out] */         error_status_t *          status
    );
```

### 6.8.23. mos_que_attr_allowenq_set

Turn enqueuing on or off in queue attributes structure.

```
    void
    mos_que_attr_allowenq_set(
        /* [in] */          boolean                   allow_enq,
        /* [in,out] */      mos_que_attr_t            que_attributes,
        /* [out] */         error_status_t *          status
    );
```

### 6.8.24. mos_que_attr_allowdeq_get

Return boolean that describes whether dequeuing is allowed or not from queue attributes structure.

```
    void
    mos_que_attr_allowdeq_get(
        /* [in] */          mos_que_attr_t            que_attributes,
        /* [out] */         boolean *                 allow_deq,
        /* [out] */         error_status_t *          status
    );
```

### 6.8.25. mos_que_attr_allowdeq_set

Turn dequeuing on or off in queue attributes structure.

```
    void
```

```
mos_que_attr_allowdeq_set(
    /* [in] */          boolean                     allow_deq,
    /* [in,out] */      mos_que_attr_t              que_attributes,
    /* [out] */         error_status_t *            status
);
```

### 6.9. Queue Management Utilities

Client interfaces for Queue Management Utilities to support the Message Oriented Service Application Programming Interface (MOS-API).

#### 6.9.1. mos_que_mgmt_create

Create one queue and commit its queue attributes to be managed by one queue manager.

```
void
mos_que_mgmt_create(
    /* [in] */          mos_handle_t          qmgr_handle,
    /* [in] */          mos_rsrc_name_t       name,
    /* [in] */          mos_que_attr_t        que_attributes,
    /* [out] */         mos_handle_t *        que_handle,
    /* [out] */         error_status_t *      status
);
```

#### 6.9.2. mos_que_mgmt_delete

Delete one queue managed by the specified queue manager.

```
void
mos_que_mgmt_delete(
    /* [in] */          mos_que_delete_op_t   flags,
    /* [in,out] */      mos_handle_t *        resource_handle,
    /* [out] */         error_status_t *      status
);
```

#### 6.9.3. mos_que_mgmt_move

Move or rename one queue within a Q-mgr or from one Q-mgr to another.

```
void
mos_que_mgmt_move(
    /* [in] */          mos_handle_t          dest_qmgr_handle,
    /* [in] */          mos_rsrc_name_t       new_que_name,
    /* [in] */          unsigned32            flags,
    /* [in/out] */      mos_handle_t *        move_que_handle,
    /* [out] */         error_status_t *      status
);
```

#### 6.9.4. mos_que_mgmt_quename_list

Return list of queues by name managed by specified Q-mgr.

```
void
mos_que_mgmt_quename_list(
    /* [in] */          mos_handle_t          resource_handle,
    /* [in] */          unsigned32            space_avail,
    /* [out] */         unsigned32 *          num_returned,
    /* [out, size_is(space_avail), length_is(*num_returned)] */
                        mos_rsrc_name_t       que_name_list[],
    /* [out] */         unsigned32 *          num_left,
    /* [out] */         error_status_t *      status
```

```
);
```

### 6.9.5. mos_que_mgmt_msgid_list

Return list of messages by UUID from the specified queue that satisfy the selection criteria.

```
void
mos_que_mgmt_msgid_list(
    /* [in] */       mos_handle_t          resource_handle,
    /* [in] */       mos_msg_selfilter_t   msg_selection_filter,
    /* [in] */       unsigned32            space_avail,
    /* [out] */      unsigned32 *          num_returned,
    /* [out, size_is(space_avail), length_is(*num_returned)] */
                     uuid_t                msg_id_list[],
    /* [out] */      unsigned32 *          num_left,
    /* [out] */      error_status_t *      status
);
```

## 7. ADMINISTRATIVE INTERFACE

Ideally, we would use `dcecp` to administer the message queuing system. There are several obstacles to this, however. Most critical is that non-source licensees do not have the `dcecp` source.

If `dcecp` allowed dynamic loading of customer-defined modules, this would be a perfect candidate. Unfortunately, this is not the case.

Instead, we will provide a stand-alone control program in a `dcecp` shell. In other words, it will be a stripped-down `dcecp` that only manipulates message queue objects. Customers of this project who have DCE source will be able to integrate our source into the DCE source with some effort.

The administrative functions described below are written as if they are incorporated into `dcecp`.

### 7.1. dcecp commands

`dcecp` will use the object `queue` for message queues. The following is a brief list of `dcecp` verbs supported. Following that are details of each command. What is shown is information `dcecp` will display when the `help` verb is given.

```
catalog       Returns names of all queues on the specified queue manager.
create        Creates the specified queue.
delete        Deletes the specified queue.
dequeue       Dequeues a message from the specified queue.
enqueue       Enqueues a message to the specified queue.
list          Returns a list of messages on the specified queue.
modify        Changes the attributes of the specified queue.
move          Moves an entire queue or a queue member to another queue.
remove        Removes a message from the specified queue.
show          Returns the attributes of the specified queue.
```

Two TCL variables are used by the various commands if the value is not explicitly set on the command line: `mos_dflt_quemgr` for the default queue manager to use, and `mos_quemgr_group` for the security group which will be checked against the groups of which the queue manager is a member. For all queue operations, if the queue manager name is not given, the default queue manager is used.

#### 7.1.1. queue catalog

Returns names of all queues on the specified queue manager.

```
-simplename     List all queues for a queue manager with simple pathnames.
-group          Checks security group name where named resource is member.
```

### 7.1.2. queue create

Creates the specified queue. If the queue name is not given, a name will be generated; it is a string version of the queue's ID (a UUID).

```
-qmaxlength     Queue maximum size.
-qmsgmaxsize    Message maximum size on queue.
-qpersistence   Queue persistence (msgpersistence|volatile|persistent).
-qallowenq      Queue enqueue service (yes|no).
-qallowdeq      Queue dequeue service (yes|no).
-qannotation    Queue annotation.
-qaliases       Queue alias(es) for the queue name.
-qidletimeout   Empty queue idle timeout in relative utc format.
-group          Checks security group name where named resource is member.
-attribute      Add queue attribute(s) to the queue being created.
```

### 7.1.3. queue delete

Deletes the specified queue.

```
-force          Force queue deletion even if queue is non-empty.
-group          Checks security group name where named resource is member.
```

### 7.1.4. queue dequeue

Dequeues a message from the specified queue.

```
-mid            Message ID criteria (UUID string).
-mtype          Message type criteria (data|notice).
-mpriority      Message priority criteria (integer).
-mdatatype      Message application-specific datatype criteria (UUID string).
-mprotectlvl    Message end-to-end protection level criteria
                  (default|none|authn|integrity|privacy).
-msender        Message sender's principal name criteria.
-mttr           Message ttr (time-to-receive) criteria.
-deqmode        Dequeue mode (nowaitdeq|deqwait).
-group          Checks security group name where named resource is member.
-criteria       List of message selection criteria.
```

### 7.1.5. queue enqueue

Enqueues a message to the specified queue.

```
-mpriority      Message priority.
-mtype          Message type (data|notice).
-mdatatype      Message application-specific datatype (uuid).
-mopnotice      Notice option(s) (none|enqueue|dequeue) on a message.
                  A combination of notice options are allowed.
-mqnotice       Notice destination for a notice sent by the messaging system.
-mgrpnotice     Notice security group name where notice destination is member.
-mpersistence   Message persistence (volatile|persistent).
-mexpiration    Message expiration time in absolute utc format.
-mreceivetime   Message ttr (time to receive) time in absolute utc format.
-mprotectlvl    Message end-to-end protection level
  (default|none|authn|integrity|privacy).
```

```
-body           Message body specified as an ASCII text string.
-enqmode        Enqueue mode (async|nowaitenq|enqwait|deqwait).
-group          Checks security group name where named resource is member.
-attribute      Add message attributes(s) to the message being enqueued.
```

### 7.1.6. queue list

Returns a list of messages on the specified queue.

```
-mid            Message ID criteria (UUID string).
-mtype          Message type criteria (data|notice).
-mpriority      Message priority criteria (integer).
-mdatatype      Message application-specific datatype criteria (UUID string).
-mprotectlvl    Message end-to-end protection level criteria
                       (default|none|authn|integrity|privacy).
-msender        Message sender's principal name criteria.
-mttr           Message ttr (time-to-receive) criteria.
-group          Checks security group name where named resource is member.
-criteria       List of message selection criteria.
```

### 7.1.7. queue modify

Changes the attributes of the specified queue.

```
-qmaxlength     Queue maximum size.
-qmsgmaxsize    Message maximum size on queue.
-qpersistence   Queue persistence (msgpersistence|volatile|persistent).
-qallowenq      Queue enqueue service (yes|no).
-qallowdeq      Queue dequeue service (yes|no).
-qannotation    Queue annotation.
-qaliases       Set a new list of queue alias(es) for the queue name.
-qidletimeout   Empty queue idle timeout in relative utc format.
-group          Checks security group name where named resource is member.
-change         Change attribute(s) on an existing queue.
```

### 7.1.8. queue move

Moves an entire queue or a queue member to another queue.

```
-to             Destination queue name.
-referral       Source and destination queue names kept by source qmgr.
-mid            Message ID to be moved.
-srcgroup       Checks security group name where source qmgr is member.
-destgroup      Checks security group name where destination qmgr is member
```

### 7.1.9. queue remove

Removes a message from the specified queue.

```
-mid            Message ID to be removed.
-group          Checks security group name where named resource is member.
```

### 7.1.10. queue show

Returns the attributes of the specified queue.

```
-mid            Returns the attributes of the specified message ID.
-body           Returns the message body only.
-all            Returns both the message attributes and message body.
-group          Checks security group name where named resource is member.
```

## 7.2. Access Control Lists

The following permissions are supported by the management interface:

- i queue creation (insert)
- m manage server (enable and disable server, etc)
- r read queue manager attributes and status
- c control (change ACL)
- t test permissions

The following permissions are supported by the queue data interface:

- D delete a queue
- M manage (change) queue attributes
- R read queue attributes
- e enqueue a message
- d dequeue a message
- r read a message's attributes
- p purge (delete) a message, without receiving its contents
- c control (change ACL)
- t test permissions

# 8. REFERENCES

Note that some of the OSF-RFC's below may be revised after the present one is published; further, some relevant new RFC's may be published. It is incumbent on the consumer of this RFC to track these developments.

[RFC 95] Guidry, Michael, Estrem, ''DCE/NEXT: REQUIREMENTS SUMMARY'', OSF-RFC 95.0, August 1996.

[RFC Index] W. Tuvell, ''Index to OSF-RFC's'', OSF-RFC Index, *Latest version*.

# 9. AUTHORS' ADDRESS

John Bowe
The Open Group
11 Cambridge Center
Cambridge, MA 02142-1405
USA
Email: bowe@osf.org
Telephone: +1-617-621-7269

Julie Salamone
The Open Group
Email: salamone@osf.org
Telephone: +1-617-621-8707

11 Cambridge Center
Cambridge, MA 02142-1405
USA