[MS-MQMR]:
Message Queuing (MSMQ):
Queue Manager Management Protocol Specification

Revision Summary

Date	Revision History	Revision Class	Comments
05/11/2007	0.1		MCPP Milestone 4 Initial Availability
08/10/2007	1.0	Major	Updated and revised the technical content.
09/28/2007	1.0.1	Editorial	Revised and edited the technical content.
10/23/2007	1.0.2	Editorial	Revised and edited the technical content.

Date	Revision History	Revision Class	Comments
11/30/2007	1.0.3	Editorial	Revised and edited the technical content.
01/25/2008	1.0.4	Editorial	Revised and edited the technical content.

Table of Contents

1 Introduction

The Message Queuing (MSMQ): Queue Manager Management Protocol is a **remote procedure call (RPC)**-based protocol used for management operations on the **MSMQ** server, including monitoring the MSMQ installation and the **queues**.

Operations that a client MAY perform using this protocol include:

- Getting information on MSMQ installation and queues.
- Performing actions on an MSMQ installation.
- Performing actions on a queue.

1.1 Glossary

The following terms are defined in [MS-GLOS]:

Administrator
Dynamic Endpoint
Endpoint
Globally Unique Identifier (GUID)
Handle
Interface Definition Language (IDL)
Opnum
Remote Procedure Call (RPC)
RPC Transport
Universally Unique Identifier (UUID)

The following terms are defined in [MS-MQMQ]:

Active Queue
Connector Queue
Distribution List
Foreign Queue
Format Name
Microsoft Message Queuing (MSMQ)
MSMQ
MSMQ Directory Service Server
MSMQ Site
Outgoing Queue
Path Name
Private Queue
Queue
Queue Journal
Queue Manager
Subqueue
Transactional Message
Transactional Queue

The following terms are specific to this document:

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as described in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site, http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

[C706] The Open Group, "DCE 1.1: Remote Procedure Call", C706, August 1997, http://www.opengroup.org/public/pubs/catalog/c706.htm

[MS-DTYP] Microsoft Corporation, "Windows Data Types", January 2007.

[MS-ERREF] Microsoft Corporation, "Windows Error Codes", January 2007.

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary", March 2007.

[MS-MQMQ] Microsoft Corporation, "Message Queuing (MSMQ): Data Structures", August 2007.

[MS-MQQB] Microsoft Corporation, "Message Queuing (MSMQ): Message Queuing Binary Protocol Specification", August 2007.

[MS-RPCE] Microsoft Corporation, "Remote Procedure Call Protocol Extensions", January 2007.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.ietf.org/rfc/rfc2119.txt

1.2.2 Informative References

[MSDN-MIDL] Microsoft Corporation, "Microsoft Interface Definition Language (MIDL)", http://msdn2.microsoft.com/en-us/library/ms950375.aspx

[MSDN-MQEIC] Microsoft Corporation, "Message Queuing Error and Information Codes", http://msdn2.microsoft.com/en-gb/library/ms700106.aspx

1.3 Protocol Overview (Synopsis)

The Message Queuing (MSMQ): Queue Manager Management Protocol allows an MSMQ client application to perform management operations on an MSMQ server.

This protocol can be used to get the following information.

- Queue properties, such as:
  - The **path name** of a queue.
  - The **format name** of a queue.
  - Whether a queue is (or is not) located on a computer, or is a **transactional queue** or a **foreign queue**.
  - The retransmit interval for messages in an **outgoing queue** for which no order acknowledgment has been received.
  - The number of **subqueues** in a specified queue.<1>

- The names of the subqueues in a specified queue.<2>
- The version and build information for the computer operating system and the MSMQ installation.

- Current queue state, such as:
  - The number of messages in a queue or in a **queue journal**.
  - The number of message bytes in a queue or in a queue journal.
  - The connection state of an outgoing queue.
  - The list of the **active queues** on a computer.
  - The name of the current **MSMQ Directory Service server** for a computer.
  - Whether a **queue manager** on a computer is disconnected from the network.
  - The list of the path names of all the **private queues** registered on a computer.
- Auditing information, such as:
  - The connection state history of a queue.<3>
  - The number of messages sent from a computer to a queue for which no order acknowledgment has been received.
  - The number of messages sent from a computer to a queue for which an order acknowledgment has been received, but a receive acknowledgment message has not been received.
  - The date and time when the last order acknowledgment for a message sent from a computer to a queue was received.
  - The time when MSMQ will attempt to retransmit a message from a computer to a queue.
  - The number of times that the last message in the corresponding outgoing queue on a computer was sent.
  - The number of times that the last order acknowledgment for a message sent from a computer to a queue has been received.
  - The number of message bytes stored in all the queues on a computer.<4>
- Sequence information, such as:
  - The address or a list of possible addresses for routing messages to the destination queue in the next hop.
  - The next message to be sent from a computer to a queue.
  - The last message that was sent from a computer to a queue for which no order acknowledgment has been received.
  - The first message sent from a computer to a queue for which no order acknowledgment has been received.

- An array of arrays of information on the **transactional messages** sent from all source computers to a queue on a target computer. Each element of the overall array is an array (vector) containing one of the following pieces of information for all of the source computers.
  - The format names used to open a queue when the last messages were sent.
  - The **globally unique identifiers (GUIDs)** of the sending queue managers.
  - The last sequence identifiers.
  - The sequence numbers of the last messages sent to a queue by one or more sending queue managers.
  - The times when each sending queue manager last accessed a queue.
  - The number of times that the last messages were rejected.

The protocol can also be used to perform actions on a computer, such as:

- Connecting the queue manager on a computer to a network and an MSMQ Directory Service server.
- Disconnecting the queue manager on a computer from a network and an MSMQ Directory Service server.
- Deleting empty message files.

The protocol can also be used to perform actions on a queue, such as:

- Pausing the sending of messages from a computer. The queue manager will not send messages to the applicable destination queue until a resume action is initiated.
- Restarting the sending of messages after a pause action is initiated.
- Resending the pending transaction sequence (as specified in [MS-MQQB]).

This is an RPC-based protocol. The server does not maintain client state information. The protocol operation is stateless.

This is a simple request-response protocol. For each received method request, the server executes the requested method and returns a completion status to the client. This is a stateless protocol; each method call is independent of any previous method calls.

1.4 Relationship to Other Protocols

The Message Queuing (MSMQ): Queue Manager Management Protocol is dependent on RPC over TCP/IP for its transport. This protocol uses RPC, as specified in section 2.1.

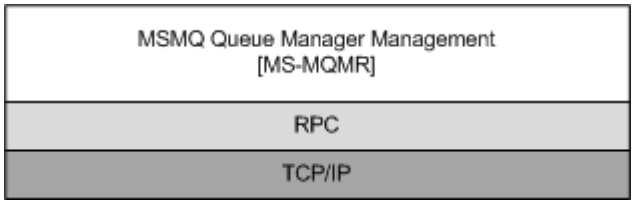

Figure 1: Protocol relationships

No other protocol currently depends on the Message Queuing (MSMQ): Queue Manager Management Protocol.

1.5 Prerequisites/Preconditions

The Message Queuing (MSMQ): Queue Manager Management Protocol is an RPC interface and, as a result, has the prerequisites specified in [MS-RPCE] as being common to RPC interfaces.

This protocol does not include any means for a client to discover the name of a remote computer that supports MSMQ . This protocol also does not include a means to discover the port number that a specific MSMQ server uses. It is assumed that the client has obtained the relevant name and port number through another means.

MSMQ clients MUST know the names of one or more remote computers that support MSMQ . MSMQ clients and servers MUST know the port number that is being used by the **MSMQ site**.

1.6 Applicability Statement

The Message Queuing (MSMQ): Queue Manager Management Protocol is used for administration of queues. The operations exposed allow IT administrators to locally or remotely perform management operations as well as to retrieve properties that describe how an MSMQ system is operating. This allows operations staff to monitor the health and activity load flowing through an MSMQ system.

1.7 Versioning and Capability Negotiation

There are no versioning issues for this protocol.

1.8 Vendor-Extensible Fields

The Message Queuing (MSMQ): Queue Manager Management Protocol uses HRESULTs, as specified in [MS-ERREF] section 2.1. Vendors can define their own HRESULT values provided that the C bit (0x20000000) is set for each vendor-defined value, indicating that the value is a customer code.

1.9 Standards Assignments

Parameter	Value	Reference
RPC interface **UUID**	{41208ee0-e970-11d1-9b9e-00e02c064c39}	[C706]
Interface version	1.0	[C706]

2 Messages

The following sections specify how Message Queuing (MSMQ): Queue Manager Management Protocol messages are transported and the common data types for this protocol.

2.1 Transport

The Message Queuing (MSMQ): Queue Manager Management Protocol uses the following remote procedure call (RPC) protocol sequence: RPC over TCP/IP (ncacn_ip_tcp), as specified in [MS-RPCE].

This protocol uses RPC **dynamic endpoints**, as specified in Part 4 of [C706].

This protocol MUST use the universally unique identifier (UUID), as specified in section 1.9.

All structures are defined in the **IDL** syntax and are marshaled as specified in [C706] sections 4, 5, and 6. The IDL is specified in section 6.

2.2 Common Data Types

The Message Queuing (MSMQ): Queue Manager Management Protocol MUST indicate to the RPC runtime that it is to support the NDR transfer syntax only, as specified in Part 4 of [C706].

In addition to RPC base types (as specified in [C706], [MS-DTYP], and [MS-RPCE]) the data types given below are defined in the Microsoft Interface Definition Language (MIDL) specification for this RPC interface.

The following table summarizes the types that are defined either in this specification or in [MS-MQMQ].

Structure	Description
DL_ID	A **distribution list** queue identifier.
MGMT_OBJECT	A structure containing information on a queue, a machine, or a session.
MULTICAST_ID	A multicast queue identifier.
OBJECTID	A structure that uniquely distinguishes a repository object from all other repository objects represented in a repository database.
QUEUE_FORMAT	Identifies the type of queue being managed and provides the appropriate connection address information.

Enumeration	Description
MgmtObjectType	Identifies the type of management object being used.
QUEUE_FORMAT_TYPE	Identifies the type of message queue being used.

2.2.1 Structures

2.2.1.1 DL_ID

This specification uses the DL_ID type, as specified in [MS-MQMQ] section 2.2.9.

2.2.1.2 MGMT_OBJECT

The **MGMT_OBJECT** structure defines information on a queue, a computer, or a session. The structure includes an embedded discriminated union.

```
typedef struct _MGMT_OBJECT {
  MgmtObjectType type;
  [switch_is(type)] union {
    [case(MGMT_QUEUE)]
      QUEUE_FORMAT* pQueueFormat;
    [case(MGMT_MACHINE)]
      DWORD Reserved1;
    [case(MGMT_SESSION)]
      DWORD Reserved2;
  };
} MGMT_OBJECT;
```

type: An integer discriminator for the embedded discriminated union. The value of this field MUST be 1, 2, or 3, as specified in section 2.2.2.1.

pQueueFormat: A pointer to a QUEUE_FORMAT structure (as specified in section 2.2.1.5) that describes the type of the queue.

Reserved1: A 32-bit unsigned integer.<5>

Reserved2: A 32-bit unsigned integer.<6>

2.2.1.3 MULTICAST_ID

This specification uses the MULTICAST_ID type, as specified in [MS-MQMQ] section **2.2.10**.<7>

2.2.1.4 OBJECTID

The OBJECTID structure (as specified in [MS-MQMQ] section **2.2.8**) uniquely distinguishes a repository object from all other repository objects represented in a repository database.

2.2.1.5 QUEUE_FORMAT

The QUEUE_FORMAT structure (as specified in [MS-MQMQ] section **2.2.7**) describes the type of queue being managed, and provides the appropriate connection address information.

2.2.2 Enumerators

2.2.2.1 MgmtObjectType

The **MgmtObjectType** enumeration identifies the type of management object (as specified in section 2.2.1.2) being used.

```
typedef  enum __MgmtObjectType
{
  MGMT_MACHINE = 1,
  MGMT_QUEUE = 2,
  MGMT_SESSION = 3
} MgmtObjectType;
```

MGMT_MACHINE: A machine management object.

MGMT_QUEUE: A queue management object.

MGMT_SESSION: A session management object.

2.2.2.2 QUEUE_FORMAT_TYPE

The QUEUE_FORMAT_TYPE enumeration ([MS-MQMQ] section 2.2.6) identifies the type of name format being used. The QUEUE_FORMAT structure ([MS-MQMQ] section 2.2.7) uses the values for the discriminated union parameter *m_qft*.

2.2.3 Property Identifiers

The **R_QMMgmtGetInfo** method uses property identifiers and corresponding property values. Property identifiers and properties are specified in [MS-MQMQ].

2.2.3.1 Management Machine Property Identifiers

Value	Meaning
PROPID_MGMT_MSMQ_ACTIVEQUEUES (0x00000001)	Retrieves a list of the active queues on a computer.
PROPID_MGMT_MSMQ_PRIVATEQ (0x00000002)	Retrieves a list of the path names of all the private queues registered on the computer.
PROPID_MGMT_MSMQ_DSSERVER (0x00000003)	Retrieves the name of the current MSMQ Directory Service server for the computer.
PROPID_MGMT_MSMQ_CONNECTED (0x00000004)	Indicates whether the queue manager on a computer has been disconnected from the network.
PROPID_MGMT_MSMQ_TYPE (0x00000005)	Retrieves the version and build information for the computer operating system and MSMQ installation.
PROPID_MGMT_MSMQ_BYTES_IN_ALL_QUEUES (0x00000006)	Retrieves the number of message bytes stored in all the queues on the computer.

2.2.3.2 Management Queue Property Identifiers

Value	Meaning
PROPID_MGMT_QUEUE_PATHNAME (0x00000001)	Retrieves the path name of a queue.

Value	Meaning
PROPID_MGMT_QUEUE_FORMATNAME (0x00000002)	Retrieves the format name of a queue.
PROPID_MGMT_QUEUE_TYPE (0x00000003)	Retrieves a string that indicates whether a queue is a public, private, system, connector, or multicast outgoing queue.
PROPID_MGMT_QUEUE_LOCATION (0x00000004)	Retrieves a string that indicates whether a queue is located on the computer.
PROPID_MGMT_QUEUE_XACT (0x00000005)	Retrieves a string that indicates whether a queue is transactional.
PROPID_MGMT_QUEUE_FOREIGN (0x00000006)	Retrieves a string that indicates whether a queue is a foreign queue.
PROPID_MGMT_QUEUE_MESSAGE_COUNT (0x00000007)	Retrieves the number of messages in a queue.
PROPID_MGMT_QUEUE_BYTES_IN_QUEUE (0x00000008)	Retrieves the number of message bytes in a queue.
PROPID_MGMT_QUEUE_JOURNAL_MESSAGE_COUNT (0x00000009)	Retrieves the number of messages in a queue journal.
PROPID_MGMT_QUEUE_BYTES_IN_JOURNAL (0x0000000A)	Retrieves the number of message bytes in a queue journal.
PROPID_MGMT_QUEUE_STATE (0x0000000B)	Retrieves the connection state of an outgoing queue .
PROPID_MGMT_QUEUE_NEXTHOPS (0x0000000C)	Retrieves the address or a list of possible addresses for routing messages to a destination queue in the next hop.
PROPID_MGMT_QUEUE_EOD_LAST_ACK (0x0000000D)	Retrieves the sequence information on the last message sent from a computer to a queue for which an order acknowledgment was received.
PROPID_MGMT_QUEUE_EOD_LAST_ACK_TIME (0x0000000E)	Retrieves the date and time when the last order acknowledgment was received for a message sent from a computer to a queue.
PROPID_MGMT_QUEUE_EOD_LAST_ACK_COUNT (0x0000000F)	Retrieves the number of times that the last order acknowledgment was received for a message sent from a computer to a queue.
PROPID_MGMT_QUEUE_EOD_FIRST_NON_ACK (0x00000010)	Retrieves the sequence information on the first message sent from a computer to a queue for which no order acknowledgment has been received.
PROPID_MGMT_QUEUE_EOD_LAST_NON_ACK (0x00000011)	Retrieves the sequence information on the last message sent from a computer to a queue for which no order acknowledgment has been

Value	Meaning
	received.
PROPID_MGMT_QUEUE_EOD_NEXT_SEQ (0x00000012)	Retrieves the sequence information on the next message to be sent from a computer to a queue.
PROPID_MGMT_QUEUE_EOD_NO_READ_COUNT (0x00000013)	Retrieves the number of messages sent from a computer to a queue for which an order acknowledgment has been received.
PROPID_MGMT_QUEUE_EOD_NO_ACK_COUNT (0x00000014)	Retrieves the number of messages sent from a computer to a queue for which no order acknowledgment has been received.
PROPID_MGMT_QUEUE_EOD_RESEND_TIME (0x00000015)	Retrieves the time when **Message Queuing (MSMQ)** will attempt to retransmit a message from a computer to a queue.
PROPID_MGMT_QUEUE_EOD_RESEND_INTERVAL (0x00000016)	Retrieves the resend interval for the messages in the outgoing queue for which no order acknowledgment has been received.
PROPID_MGMT_QUEUE_EOD_RESEND_COUNT (0x00000017)	Retrieves the number of times that the last message was sent in the corresponding outgoing queue on a computer.
PROPID_MGMT_QUEUE_EOD_SOURCE_INFO (0x00000018)	Retrieves information on the transactional messages sent from all source computers to a queue.<8>
PROPID_MGMT_QUEUE_CONNECTION_HISTORY (0x00000019)	Retrieves queue connection state history.<9>
PROPID_MGMT_QUEUE_SUBQUEUE_COUNT (0x0000001A)	Retrieves a count of the number of subqueues for a given queue.<10>
PROPID_MGMT_QUEUE_SUBQUEUE_NAMES (0x0000001B)	Retrieves a list of subqueues for a given queue.<11>

3 Protocol Details

The Message Queuing (MSMQ): Queue Manager Management Protocol is used for performing management operations on the MSMQ installation and a queue.

The client side of this protocol is simply a pass-through. That is, there are no timers or other states required on the client side. Calls made by a higher-layer protocol or an application are passed directly to the transport, and the results returned by the transport are passed directly back to the higher-layer protocol or application.

The client MUST have **administrator** privileges on the server machine.

This protocol permits establishing a connection to a remote procedure call (RPC) server. For each connection, the server uses the underlying RPC protocol to retrieve the identity of the invoking client call, as specified in [MS-RPCE] section 3.3.3.4.3. The server should use this identity to perform method-specific access checks, as specified in section 3.1.4.

The methods comprising this RPC interface all return 0x00000000 on success and a non-zero implementation-specific error code on failure. Unless otherwise specified in the following sections, a server-side implementation of this protocol may choose any non-zero Win32 error value to signify an error condition, as specified in section 1.8. The client side of the Message Queuing (MSMQ): Queue Manager Management Protocol does not need to interpret the error codes returned from the server; instead, the client side can return the error code unprocessed to the invoking application without taking any protocol action.

Note The phrases client side and server side refer to the initiating and receiving ends of the protocol, respectively, rather than to client or server versions of an operating system. The receiving end of the protocol—the server side—behaves the same regardless of whether the server side is running on a client or server.

3.1 qmmgmt Server Details

3.1.1 Abstract Data Model

No abstract data model is required.

3.1.2 Timers

The Message Queuing (MSMQ): Queue Manager Management Protocol layer uses no timers. RPC does, however, use timers internally, as specified in [MS-RPCE].

3.1.3 Initialization

Software that utilizes **qmmgmt** MUST establish an RPC connection to the client prior to utilizing this protocol, as specified in [MS-RPCE] and section 2.1.

3.1.4 Message Processing Events and Sequencing Rules

The Message Queuing (MSMQ): Queue Manager Management Protocol MUST indicate to the RPC runtime that it is to perform a strict NDR data consistency check at target level 6.0, as specified in [MS-RPCE] section 3.

This protocol MUST indicate to the RPC runtime that it is to reject a NULL unique or full pointer with non-zero conformant value, as specified in [MS-RPCE] section 3.

This protocol MUST indicate to the RPC runtime via the strict_context_handle attribute that it is to reject use of context **handles** created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

This interface includes the following methods.

Methods in RPC Opnum Order

Method	Description
R_QMMgmtGetInfo (section 3.1.4.1)	Called by the client. In response, the server returns information on a queue or the MSMQ installation on the server. Opnum: 0
R_QMMgmtAction (section 3.1.4.2)	Called by the client. In response, the server performs a queue management function specified by the supplied **MGMT_OBJECT** structure. Opnum: 1

All methods MUST NOT throw exceptions.

3.1.4.1 R_QMMgmtGetInfo (Opnum 0)

The **R_QMMgmtGetInfo** method requests information on an MSMQ installation on a server, or on a specific queue.

```
HRESULT R_QMMgmtGetInfo(
  [in] handle_t hBind,
  [in] const MGMT_OBJECT* pObjectFormat,
  [in, range(1,128)] DWORD cp,
  [in, size_is(cp)] ULONG aProp[],
  [in, out, size_is(cp)] PROPVARIANT apVar[]
);
```

hBind: An RPC binding handle, as specified in [MS-RPCE] section 2.

pObjectFormat: A pointer to an **MGMT_OBJECT** structure that defines the queue or computer on which to return information.

cp: The length (in elements) of the arrays *aProp* and *apVar*. MUST be at least 1, and MUST be no greater than 128.

aProp: Points to an array of property identifiers associated with the array of property values. This array MUST contain at least one element. Each element MUST specify a value from the property identifiers table, as specified in section 2.2.3. Each element MUST specify the property identifier for the corresponding property value at the same element index in *apVar*. This array and the array to which *apVar* points MUST be of the same length.

apVar: Points to an array that specifies the property values associated with the array of property identifiers. Each element in this array specifies the property value for the corresponding property identifier at the same element index in the array to which *aProp* points. This array MUST contain at least one element. The property value in each element

MUST correspond accordingly to the property identifier from *aProp*, as specified in section 2.2.3, and MUST be set to VT_NULL<12> (as specified in [MS-MQMQ] section 2.2.12) before each call to **R_QMMgmtGetInfo**. This array and the array to which *aProp* points MUST be of the same length.

Return Values: On success, this method MUST return MQ_OK (0x00000000).

MQ_OK (0x00000000)
MQ_ERROR (0xC00E0001)

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [MS-RPCE].

If an error occurs, the server MUST return a failure **HRESULT**, and MUST NOT set any [out] parameter values.

The **opnum** field value for this method MUST be 0, and is received at a dynamically-assigned **endpoint** supplied by the RPC endpoint mapper, as specified in [MS-RPCE].

If the *pObjectFormat* parameter specifies an **MgmtObjectType** of MGMT_MACHINE, the server MUST return only those properties that pertain to the MSMQ installation. If *pObjectFormat* specifies an **MgmtObjectType** of MGMT_QUEUE, the server MUST return only those properties that pertain to a queue. If *pObjectFormat* specifies an **MgmtObjectType** of MGMT_SESSION, the call MUST fail, and the error message MAY be MQ_ERROR_INVALID_PARAMETER (0xC00E0006).<13>

If the *pObjectFormat* parameter specifies a computer, and one or more of the properties specified in *aProp* are different than those specified in section 2.2.3.1, the call MAY fail with MQ_ERROR_ILLEGAL_PROPID (0xC00E0039). If the *pObjectFormat* parameter specifies a queue, and one or more of the properties specified in *aProp* are different than those specified in section 2.2.3.2, the call MAY fail with MQ_ERROR_ILLEGAL_PROPID (0xC00E0039).<14>

MSMQ properties are specified in [MS-MQMQ] section 2.

For MSMQ error codes, see [MSDN-MQEIC]. The structure and sequence of data on the wire are specified in [C706] Transfer Syntax NDR.

3.1.4.2 R_QMMgmtAction (Opnum 1)

The **R_QMMgmtAction** method requests the server to perform a management function on a specific queue or MSMQ installation.

```
HRESULT R_QMMgmtAction(
  [in] handle_t hBind,
  [in] const MGMT_OBJECT* pObjectFormat,
  [in, string] const wchar_t* lpwszAction
);
```

hBind: An RPC binding handle, as specified in [MS-RPCE] section 2.

pObjectFormat: A pointer to a **MGMT_OBJECT** structure that specifies the queue or computer to which the action is being applied.

lpwszAction: A pointer to a null-terminated Unicode string that specifies the action to perform on the computer. The *lpwszAction* value MUST be one of the following (the value is not case sensitive).

Value	Meaning
"CONNECT"	A machine action. Connects the computer to the network and the MSMQ Directory Service server.
"DISCONNECT"	A machine action. Disconnects the computer from the network and the MSMQ Directory Service server.
"TIDY"	A machine action. Cleans up empty message files. MSMQ does this every six hours. It is helpful when a large number of messages are deleted (purged or received by an application), and the application needs the disk space immediately.
"PAUSE"	A queue action. Valid for outgoing queues only. Stops the sending of messages from the computer. The queue manager will not send messages to the applicable destination queue until a RESUME action is initiated.
"RESUME"	A queue action. Valid for outgoing queues only. Restarts the sending of messages after a PAUSE action is initiated.
"EOD_RESEND"	A queue action. Resends the pending transaction sequence.

Return Values: On success, this method MUST return MQ_OK (0x00000000).

MQ_OK (0x00000000)
MQ_ERROR (0xC00E0001)

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [MS-RPCE].

If *pObjectFormat* specifies an **MgmtObjectType** of MGMT_SESSION, the call MUST fail, and the error message MAY be MQ_ERROR_INVALID_PARAMETER (0xC00E0006).<15>

If an error occurs, the server MUST return a failure HRESULT.

The opnum field value for this method MUST be 1 and is received at a dynamically-assigned endpoint supplied by the RPC endpoint mapper, as specified in [MS-RPCE].

For MSMQ error codes, see [MSDN-MQEIC]. The structure and sequence of data on the wire are specified in the Transfer Syntax NDR section in [C706].

3.1.5 Timer Events

No protocol timer events are required on the server beyond the timers required in the underlying **RPC transport**.

3.1.6 Other Local Events

There are no local events used on the server beyond the events maintained in the underlying RPC transport.

3.2 qmmgmt Client Details

3.2.1 Abstract Data Model

The client maintains an RPC binding handle that it passes to each of the following methods:

- **R_QMMgmtAction**
- **R_QMMgmtGetInfo**

The procedure for acquiring a binding handle is specified in [C706].

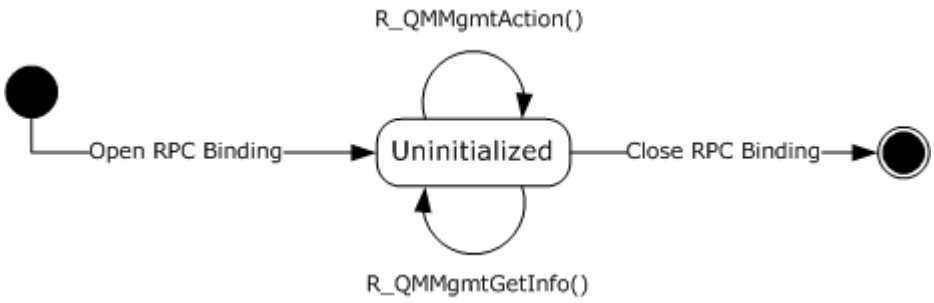

Figure 2: RPC binding and method calls

3.2.2 Timers

The Message Queuing (MSMQ): Queue Manager Management Protocol layer uses no timers. RPC does, however, use timers internally (as specified in [MS-RPCE]).

3.2.3 Initialization

Software that utilizes **qmmgmt** MUST establish an RPC connection to the server prior to utilizing this protocol, as specified in [MS-RPCE] and section 2.1.

3.2.4 Message Processing Events and Sequencing Rules

The client side of the Message Queuing (MSMQ): Queue Manager Management Protocol requires no special processing or interpretation of data or error messages beyond those required by the underlying RPC protocol.

When a method completes, the client MUST return without modification all values returned by the RPC to the upper layer.

The client MUST ignore errors returned from the RPC server and MUST notify the higher layer of the error received. The client SHOULD ignore all out-parameter values when any failure **HRESULT** is returned.

This protocol MUST indicate to the RPC runtime that it is to perform a strict NDR data consistency check at target level 6.0, as specified in [MS-RPCE] section 3.

This protocol MUST indicate to the RPC runtime that it is to reject a NULL unique or full pointer with nonzero conformant value, as specified in [MS-RPCE] section 3.

3.2.5 Timer Events

There are no timer events.

3.2.6 Other Local Events

There are no local events.

4 Protocol Examples

The following example pseudocode demonstrates how to pause a public queue by using the **R_QMMgmtAction** method.

```
////////////////////////////////////////////////////////////////////
INIT qf of type QUEUE_FORMAT
INIT mgmtObj of type MGMT_OBJECT

SET qf.m_qft to QUEUE_FORMAT_TYPE_PUBLIC
SET qf.m_SuffixAndFlags to 0
SET qf.m_gPublicID to GUID of the public queue

SET mgmtObj.type to MGMT_QUEUE;
SET mgmtObj.pQueueFormat to qf;

CALL R_QMMgmtAction with RPC binding handle, mgmtObj
     and action const "PAUSE"
////////////////////////////////////////////////////////////////////
```

5 Security

The following sections specify security considerations for implementers of the Message Queuing (MSMQ): Queue Manager Management Protocol.

5.1 Security Considerations for Implementers

As specified in section 3, this protocol allows a client with administrator privileges to connect to the server. Security is dependent on the server performing security checks for each invocation of the server interface methods specified in this document. Any security bug in the server implementation of this protocol could be exploitable.

5.2 Index of Security Parameters

No security parameters are specified for this protocol.

6 Appendix A: Full IDL

For ease of implementation, the full IDL is provided below, where "ms-dtyp.idl" refers to the IDL found in [MS-DTYP] Appendix A, and "ms-mqmq.idl" refers to the IDL found in [MS-MQMQ] Appendix A. The syntax uses the IDL syntax extensions defined in [MS-RPCE] sections 2.2.4 and 3.1.1.5.1. For example, as noted in [MS-RPCE] section 2.2.4.9, a pointer_default declaration is not required, and pointer_default(unique) is assumed.

```
import "ms-dtyp.idl";
import "ms-mqmq.idl";

[
    uuid(41208ee0-e970-11d1-9b9e-00e02c064c39),
    version(1.0),
    pointer default(unique)
]
interface qmmgmt
{
    typedef enum   MgmtObjectType {
        MGMT MACHINE = 1,
        MGMT QUEUE = 2,
        MGMT_SESSION = 3,
    } MgmtObjectType;

    typedef enum   QUEUE FORMAT TYPE
    {
        QUEUE FORMAT TYPE UNKNOWN= 0,
        QUEUE_FORMAT_TYPE_PUBLIC = 1,
        QUEUE_FORMAT_TYPE_PRIVATE= 2,
        QUEUE_FORMAT_TYPE_DIRECT= 3,
        QUEUE FORMAT TYPE MACHINE= 4,
        QUEUE FORMAT TYPE CONNECTOR= 5,
        QUEUE FORMAT TYPE DL= 6,
        QUEUE FORMAT TYPE MULTICAST= 7,
        QUEUE_FORMAT_TYPE_SUBQUEUE= 8
    } QUEUE_FORMAT_TYPE;

    typedef struct  OBJECTID {
        GUID Lineage;
        DWORD Uniquifier;
    } OBJECTID;

    typedef struct  MULTICAST ID {
        ULONG m address;
        ULONG m port;
    } MULTICAST_ID;

    typedef struct  DL ID {
        GUID       m DlGuid;
        wchar t *  m pwzDomain;
    } DL ID;

    typedef struct __QUEUE_FORMAT {
        unsigned char m qft;
        unsigned char m SuffixAndFlags;
        USHORT m reserved;

        [switch_is(m_qft)] union {
            [case(QUEUE_FORMAT_TYPE_UNKNOWN)]
                ;    // No member is set. Selected when an m qft value
                     // of 0 is returned.
            [case(QUEUE_FORMAT_TYPE_PUBLIC)]
```

```
                GUID      m gPublicID;
            [case(QUEUE_FORMAT_TYPE_PRIVATE)]
                OBJECTID  m_oPrivateID;
            [case(QUEUE FORMAT TYPE DIRECT)]
                wchar t*  m pDirectID;
            [case(QUEUE FORMAT TYPE MACHINE)]
                GUID      m gMachineID;
            [case(QUEUE_FORMAT_TYPE_CONNECTOR)]
                GUID      m_GConnectorID;
            [case(QUEUE FORMAT TYPE DL)]
                DL ID     m DlID;
            [case(QUEUE FORMAT TYPE MULTICAST)]
                MULTICAST ID m MulticastID;
            [case(QUEUE_FORMAT_TYPE_SUBQUEUE)]
                wchar_t * m_pDirectSubqueueID;
        };
    } QUEUE FORMAT;

    typedef struct  MGMT OBJECT {
        MgmtObjectType type;
        [switch is(type)] union
        {
            [case(MGMT QUEUE)]
                QUEUE FORMAT* pQueueFormat;
            [case(MGMT_MACHINE)]
                DWORD         Reserved1;
            [case(MGMT_SESSION)]
                DWORD         Reserved2;
        };
    } MGMT OBJECT;


    /*====================================================================
        QM Management functions
    ====================================================================*/

    HRESULT R QMMgmtGetInfo(
        [in] handle_t hBind,
        [in] const MGMT_OBJECT* pObjectFormat,
        [in, range(1,128)] DWORD cp,
        [in, size is (cp)] ULONG aProp[],
        [in, out, size is(cp)] PROPVARIANT apVar[]
        );

    HRESULT R QMMgmtAction(
        [in] handle t hBind,
        [in] const MGMT OBJECT* pObjectFormat,
        [in, string] const wchar t * lpwszAction
        );
}
```

7 Appendix B: Windows Behavior

The information in this specification is applicable to the following versions of Windows:

- Windows NT
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows Vista
- Windows Server 2008

Exceptions, if any, are noted below. Unless otherwise specified, any statement of optional behavior in this specification prescribed using the terms SHOULD or SHOULD NOT implies Windows behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that Windows does not follow the prescription.

<1> Section 1.3: Implemented in the Windows Vista operating system and the Windows Server 2008.

<2> Section 1.3: Implemented in Windows Vista and Windows Server 2008.

<3> Section 1.3: Implemented in Windows Vista and Windows Server 2008.

<4> Section 1.3: Implemented in Windows Server 2003, Windows Vista, and Windows Server 2008.

<5> Section 2.2.1.2: The value of this member is ignored by Windows.

<6> Section 2.2.1.2: The value of this member is ignored by Windows.

<7> Section 2.2.1.3: Available only for servers implemented on the Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP.

<8> Section 2.2.3.2: Implemented in Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP.

<9> Section 2.2.3.2: Implemented in Windows Vista and Windows Server 2008.

<10> Section 2.2.3.2: Implemented in Windows Vista and Windows Server 2008.

<11> Section 2.2.3.2: Implemented in Windows Vista and Windows Server 2008.

<12> Section 3.1.4.1: If a server cannot retrieve a property value that corresponds to an occurrence of the PROPID_MGMT_MSMQ_DSSERVER property identifier (section 2.2.3.1) in *aProp*, then the server sets the corresponding *apVar* entry to VT_NULL.

<13> Section 3.1.4.1: The Windows NT and Windows 2000 implementations return MQ_ERROR (0xC00E0001).

<14> Section 3.1.4.1: Implemented in Windows Server 2008, Windows Vista and Windows Server 2003.

<15> Section 3.1.4.2: The Windows NT and Windows 2000 implementations return MQ_ERROR (0xC00E0001).

8 Index

T

V

W