[MS-MQMP]:
Message Queuing (MSMQ):
Queue Manager Client 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	2.0	Major	Updated and revised the technical content.
10/23/2007	3.0	Major	Revised method error code return text and made

Date	Revision History	Revision Class	Comments
			changes to cursor state diagrams.
11/30/2007	4.0	Major	Updated and revised the technical content.
01/25/2008	4.1.1	Editorial	Revised and edited the technical content.

Table of Contents

1 Introduction

The Message Queuing (MSMQ): Queue Manager Client Protocol is a Microsoft proprietary RPC-based protocol, which enables communication between **message queuing client** applications and an **MSMQ Queue Manager**. Operations that a client would perform using this protocol include:

- Managing local private **queues**.
- Opening and closing local and remote queue handles.
- Enlisting, committing, and aborting internal **transactions**.
- Enlisting the queue manager in **external transactions**.
- Purging queues.
- Creating local and remote **cursor**.
- Sending **messages**.
- Reading messages.

1.1 Glossary

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

Authentication Level
Client
Dynamic Endpoint
Endpoint
Globally Unique Identifier (GUID)
Interface Definition Language (IDL)
Network Data Representation (NDR)
Opnum
Remote Procedure Call (RPC)
RPC Protocol Sequence
RPC Transport
Security Descriptor
Server
Session
Universally Unique Identifier (UUID) or Globally Unique Identifier (GUID)
UTC

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

Resource Manager
Transaction
Work

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

Administration Queue
Cursor
Dead-Letter Queue
Dependent Client
External Transaction

Foreign Queue
Internal Transaction
Message
Message Queuing
MSMQ
MSMQ Queue Manager
MSMQ Server
MSMQ Supporting Server
MSMQ 1.0 Digital Signature
MSMQ 2.0 Digital Signature
MSMQ 3.0 Digital Signature
Order Queue
Outgoing Queue
Path Name
Private Queue
Queue
Queue Journal
Queue Property
Remote Queue
Remote Read
XML Digital Signature

The following terms are specific to this document:

Cryptographic Service Provider (CSP): A program that generates digital signatures.

Legacy Remote Read Sequence: The sequence of calls used to perform a **remote read** using the qmcomm and qm2qm Message Queuing (MSMQ): Queue Manager to Queue Manager Protocol **RPC** interfaces. **Dependent clients** always utilize the legacy remote read sequence rather than the call sequence provided by the newer RemoteRead **RPC** interface specified in [MS-MQQP].

Unit of Work (UOW): The set of operations associated with a transaction. For definitions of **Work** and Transaction, see [MS-DTCO].

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-DTCO] Microsoft Corporation, "MSDTC Connection Manager: OleTx Transaction Protocol Specification", July 2007.

[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-MQMA] Microsoft Corporation, "Message Queuing (MSMQ): Architecture Protocol Specification", August 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-MQQP] Microsoft Corporation, "Message Queuing (MSMQ): Queue Manager to Queue Manager Protocol Specification", August 2007.

[MS-MQRR] Microsoft Corporation, "Message Queuing (MSMQ): Queue Manager Remote Read Protocol Specification", June 2007.

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

[RC4] The RC4 Encryption Algorithm. RSA Data Security, Inc., http://www.rsa.com/node.aspx?id=1204

Note To obtain this stream cipher that is licensed by RSA Data Security, you need to contact this company.

[RFC1319] Kaliski, B., "The MD2 Message-Digest Algorithm", RFC 1319, April 1992, http://www.ietf.org/rfc/rfc1319.txt

[RFC1320] Rivest, R. "The MD4 Message-Digest Algorithm", RFC 1320, April 1992, http://www.ietf.org/rfc/rfc1320.txt

[RFC1321] Rivest, R. "The MD5 Message-Digest Algorithm", RFC 1321, April 1992, http://www.ietf.org/rfc/rfc1321.txt

[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

[RFC2268] Rivest, R., "A Description of the RC2(r) Encryption Algorithm", RFC 2268, March 1998, http://www.ietf.org/rfc/rfc2268.txt

[RFC3174] Eastlake III, D. and Jones, P., "US Secure Hash Algorithm 1 (SHA1)", RFC 3174, September 2001, http://www.ietf.org/rfc/rfc3174.txt

[RFC4234] Crocker, D., Ed. and Overell, P., "Augmented BNF for Syntax Specifications: ABNF", RFC 4234, October 2005, http://www.ietf.org/rfc/rfc4234.txt

1.2.2 Informative References

[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 Client Protocol provides a means to implement **dependent client** and **supporting server** functionality for **MSMQ**. A client application uses this

protocol to perform basic message queuing operations on a supporting server, such as creating queues, altering **queue properties**, sending messages, and receiving messages.

1.4 Relationship to Other Protocols

This protocol is dependent upon **RPC** for its transport. This protocol uses RPC, as specified in section 2.1.

This protocol collaborates with the functionality provided by the qm2qm RPC interface, as specified in [MS-MQQP]. Additionally, MSDTC Connection Manager: OleTx Transaction Protocol is used by applications to orchestrate external transaction scenarios for this protocol.

1.5 Prerequisites/Preconditions

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

It is assumed that a Message Queuing (MSMQ): Queue Manager Client Protocol client has obtained the name of a remote computer that supports the Message Queuing (MSMQ): Queue Manager Client Protocol before this protocol is invoked. This specification does not address how this information is acquired. In the context of a **remote read** operation, this protocol provides the name of a remote **server**, as described in sections 3.1.4.1 and 3.1.4.17.

1.6 Applicability Statement

This protocol provides functionality for message queuing applications running on dependent clients to perform operations on a remote supporting server.

The server side of the Message Queuing (MSMQ): Queue Manager Client Protocol is applicable for implementation by a queue manager providing supporting server services to dependent clients. The client side of this protocol is applicable for implementation by client libraries providing message queuing services to applications, or by a queue manager delegating requests on behalf of a client.

Due to performance and security limitations, this protocol is deprecated and suitable only for interoperability with existing legacy servers and clients. Implementers of new message queuing applications are encouraged to invoke the MSMQ COM API remotely via DCOM in preference to the capabilities specified by the Message Queuing (MSMQ): Queue Manager Client Protocol.

1.7 Versioning and Capability Negotiation

This protocol supports a mechanism for explicitly negotiating the RPC **endpoint** to be used. For more information, see section 3.1.4.24.

1.8 Vendor-Extensible Fields

This protocol uses HRESULTs, as specified in [MS-ERREF] section 2. Vendors are free to choose their own values for this field, as long as the C bit (0x20000000) is set, indicating that it is a customer code.

1.9 Standards Assignments

Parameter	Value	Reference
RPC Interface **UUID** for qmcomm interface	fdb3a030-065f-11d1-bb9b-00a024ea5525	As specified in [C706].

Parameter	Value	Reference
RPC Interface UUID for qmcomm2 interface	76d12b80-3467-11d3-91ff-0090272f9ea3	As specified in [C706].
Interface Version	1.0	As specified in [C706].
Port Information	This protocol uses RPC **dynamic endpoints** as defined in [C706] Part 4, as well as a fixed endpoint as described in section 2.1.	As specified in [C706].

2 Messages

The following sections specify how Message Queuing (MSMQ): Queue Manager Client Protocol messages are transported and common data types.

2.1 Transport

This protocol SHOULD use the following **RPC protocol sequence**: RPC over TCP/IP (ncacn_ip_tcp), as specified in [MS-RPCE].<1> This protocol MAY use the RPC over SPX (ncacn_spx) protocol sequence if TCP/IP is unavailable.<2>

This protocol SHOULD use RPC dynamic endpoints, as specified in [C706] part 4. This protocol MAY use an RPC static endpoint, as specified in [C706] part 4.

This protocol allows any user to establish a connection to the RPC server. For each connection, the server uses the underlying RPC protocol to retrieve the identity of the invoking client as specified in the second bullet point of [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.

2.2 Common Data Types

All structures are defined in the **IDL** syntax and are marshaled as specified in [C706] part 3. The IDL is specified in section 6.

Note that LPWSTR or WCHAR* types specified in an IDL structure that are annotated with the **[string]** attribute MUST be null-terminated, as specified in [C706].

HRESULT: This specification uses the HRESULT type, as specified in [MS-ERREF] section 2.

Note that throughout this specification, the phrase "a failure HRESULT" means any HRESULT where the Severity (S) bit is set, as specified by [MS-ERREF]. When this specification mandates the return of "a failure HRESULT" from a method, the specific error code is not relevant to the protocol, as long as the Severity bit is set. In this circumstance, the server MAY return MQ_ERROR (0xC00E0001), or any other HRESULT value where the Severity bit is set, such as a context-specific Message Queuing error code, as specified in [MS-MQMQ] section 2.7.

GUID and UUID: This type specifies a **globally unique identifier**, as specified in [MS-DTYP] section **2.3.4**.

QUEUE_FORMAT and OBJECTID: These structures are defined in [MS-MQMQ] section 2.2.

PCTX_RRSESSION_HANDLE_TYPE: This type represents a remote queue handle, as specified in [MS-MQQP] section 2.2.1.1.

In addition to RPC base types and definitions specified in [C706] and [MS-RPCE], additional data types are defined below.

The following table summarizes the types that are defined in this specification.

Data type name	Description
RPC_INT_XACT_HANDLE	A context handle representing an **internal transaction**.
RPC_QUEUE_HANDLE	A context handle representing a queue object.

Data type name	Description
PCTX_OPENREMOTE_HANDLE_TYPE	A context handle representing a remote queue object.
CACTransferBufferV1	A structure used for sending and receiving messages.
CACTransferBufferV2	A structure containing the CACTransferBufferV1 structure used for sending and receiving messages with additional transaction tracking capabilities.
CACCreateRemoteCursor	A structure used for creating a cursor on a message queue.
OBJECT_FORMAT	A structure containing a QUEUE_FORMAT structure (as specified in [MS-MQMQ]) and a specification as to whether the QUEUE_FORMAT is local or remote.
XACTUOW	Identifies the **unit of work** for a transactional operation.

2.2.1 Data Types

2.2.1.1 Handle Data Types

2.2.1.1.1 RPC_INT_XACT_HANDLE

A remote procedure call (RPC) context handle representing an internal transaction, as specified in [C706] section 14.2.16.6. A client MUST call **R_QMEnlistInternalTransaction** to create an **RPC_INT_XACT_HANDLE** handle, and **R_QMCommitTransaction** or **R_QMAbortTransaction** to delete an **RPC_INT_XACT_HANDLE** handle. See sections 3.1.4.14, 3.1.4.15, and 3.1.4.16.

This type is declared as follows:

```
typedef [context_handle] void* RPC_INT_XACT_HANDLE;
```

2.2.1.1.2 RPC_QUEUE_HANDLE

An RPC context handle representing a queue object, as specified in [C706] section 14.2.16.6. A client MUST call **rpc_QMOpenQueueInternal** to create an **RPC_QUEUE_HANDLE** handle, and **rpc_ACCloseHandle** to close an **RPC_QUEUE_HANDLE** handle. See sections 3.1.4.17 and 3.1.4.18.

This type is declared as follows:

```
typedef [context_handle] void* RPC_QUEUE_HANDLE;
```

2.2.1.1.3 PCTX_OPENREMOTE_HANDLE_TYPE

An RPC context handle representing a queue object at a queue manager other than the supporting server, as specified in [C706] section 14.2.16.6. A client MUST call **R_QMOpenRemoteQueue** to create a **PCTX_OPENREMOTE_HANDLE_TYPE** handle, and **R_QMCloseRemoteQueueContext** to close a **PCTX_OPENREMOTE_HANDLE_TYPE** handle. See sections 3.1.4.2 and 3.1.4.3.

This type is declared as follows:

```
typedef [context_handle] void* PCTX_OPENREMOTE_HANDLE_TYPE;
```

2.2.2 Structures

2.2.2.1 XACTUOW

The **XACTUOW** structure uniquely identifies the unit of work (UOW) for a transactional operation. For an external transaction, this value MUST be acquired from the transaction coordinator. For an internal transaction, a client MUST create a unique random value for each transaction.<3>

```
typedef struct _XACTUOW
 {
  unsigned char rgb[16];
} _XACTUOW;
```

rgb: MUST be an array of bytes containing a globally unique identifier (GUID).

2.2.2.2 CACTransferBufferV1

The **CACTransferBufferV1** structure is used to send and receive messages via MSMQ.

Following is the layout of the **CACTransferBufferV1** structure with IDL annotations followed by descriptions of the structure members.

```
typedef struct CACTransferBufferV1 {
  [range(0,2)] DWORD uTransferType;
  [switch_is(uTransferType)] union {
    [case(CACTB_SEND)]
      struct {
      struct QUEUE_FORMAT* pAdminQueueFormat;
      struct QUEUE_FORMAT* pResponseQueueFormat;
    } Send;
    [case(CACTB_RECEIVE)]
      struct {
      DWORD RequestTimeout;
      DWORD Action;
      DWORD Asynchronous;
      DWORD Cursor;
      [range(0,1024)] DWORD ulResponseFormatNameLen;
      [size_is(,ulResponseFormatNameLen)]
        WCHAR** ppResponseFormatName;
      DWORD* pulResponseFormatNameLenProp;
      [range(0,1024)] DWORD ulAdminFormatNameLen;
```

```
    [size is(,ulAdminFormatNameLen)]
      WCHAR** ppAdminFormatName;
    DWORD* pulAdminFormatNameLenProp;
    [range(0,1024)] DWORD ulDestFormatNameLen;
    [size is(,ulDestFormatNameLen)]
      WCHAR** ppDestFormatName;
    DWORD* pulDestFormatNameLenProp;
    [range(0,1024)] DWORD ulOrderingFormatNameLen;
    [size_is(,ulOrderingFormatNameLen)]
      WCHAR** ppOrderingFormatName;
    DWORD* pulOrderingFormatNameLenProp;
  } Receive;
  [case(CACTB CREATECURSOR)]
    struct CACCreateRemoteCursor CreateCursor;
};
unsigned short* pClass;
OBJECTID** ppMessageID;
[size is(,20), length is(,20)]
  unsigned char** ppCorrelationID;
DWORD* pSentTime;
DWORD* pArrivedTime;
unsigned char* pPriority;
unsigned char* pDelivery;
unsigned char* pAcknowledge;
unsigned char* pAuditing;
DWORD* pApplicationTag;
[size_is(,ulAllocBodyBufferInBytes), length_is(,ulBodyBufferSizeInBytes)]
  unsigned char** ppBody;
DWORD ulBodyBufferSizeInBytes;
DWORD ulAllocBodyBufferInBytes;
DWORD* pBodySize;
[size_is(,ulTitleBufferSizeInWCHARs), length_is(,ulTitleBufferSizeInWCHARs)]
  WCHAR** ppTitle;
DWORD ulTitleBufferSizeInWCHARs;
DWORD* pulTitleBufferSizeInWCHARs;
DWORD ulAbsoluteTimeToQueue;
DWORD* pulRelativeTimeToQueue;
DWORD ulRelativeTimeToLive;
DWORD* pulRelativeTimeToLive;
unsigned char* pTrace;
DWORD* pulSenderIDType;
[size is(,uSenderIDLen)] unsigned char** ppSenderID;
DWORD* pulSenderIDLenProp;
DWORD* pulPrivLevel;
DWORD ulAuthLevel;
unsigned char* pAuthenticated;
DWORD* pulHashAlg;
DWORD* pulEncryptAlg;
[size_is(,ulSenderCertLen)] unsigned char** ppSenderCert;
DWORD ulSenderCertLen;
DWORD* pulSenderCertLenProp;
[size is(,ulProvNameLen)] WCHAR** ppwcsProvName;
DWORD ulProvNameLen;
DWORD* pulAuthProvNameLenProp;
DWORD* pulProvType;
long fDefaultProvider;
[size is(,ulSymmKeysSize)] unsigned char** ppSymmKeys;
DWORD ulSymmKeysSize;
DWORD* pulSymmKeysSizeProp;
unsigned char bEncrypted;
unsigned char bAuthenticated;
unsigned short uSenderIDLen;
[size_is(,ulSignatureSize)] unsigned char** ppSignature;
DWORD ulSignatureSize;
DWORD* pulSignatureSizeProp;
```

```
  GUID** ppSrcQMID;
  XACTUOW* pUow;
  [size_is(,ulMsgExtensionBufferInBytes), length_is(,ulMsgExtensionBufferInBytes)]
    unsigned char** ppMsgExtension;
  DWORD ulMsgExtensionBufferInBytes;
  DWORD* pMsgExtensionSize;
  GUID** ppConnectorType;
  DWORD* pulBodyType;
  DWORD* pulVersion;
} ;
```

uTransferType: The **uTransferType** member must specify which of the **Send**, **Receive**, or **CreateCursor** union members is present in the **CACTransferBufferV1** structure. The **uTransferType** member value MUST be assigned from the list below:

Value	Meaning
CACTB_SEND 0x00000000	A send operation (that is, a message placed into a queue for delivery) is to be performed.
CACTB_RECEIVE 0x00000001	A receive operation (that is, a message is to be read from a queue) is to be performed.
CACTB_CREATECURSOR 0x00000002	A cursor creation is to be performed.

Send: The Send structure is present in the **CACTransferBufferV1** structure when the value of the **uTransferType** member is 0x00000000 (CACTB_SEND). The Send structure is defined inline to the **CACTransferBufferV1** structure. The Send structure members are defined as follows:

pAdminQueueFormat: The **pAdminQueueFormat** member is a **QUEUE_FORMAT** structure as specified in [MS-MQMQ] section 2.2.7. If present, the **pAdminQueueFormat** member describes the **Administration queue** that is to be used for send operation acknowledgments.

pResponseQueueFormat: The **pResponseQueueFormat** member is a **QUEUE_FORMAT** structure as specified in [MS-MQMQ] section 2.2.7. If present, the **pResponseQueueFormat** member describes the queue that is to be used for application-specific responses. As an application-specific value, this field SHOULD be ignored by the server.

Receive: The Receive structure is present in the **CACTransferBufferV1** structure when the value of the **uTransferType** member is 0x00000001 (CACTB_RECEIVE). The Receive structure is defined inline to the **CACTransferBufferV1** structure. The Receive structure members are defined as follows:

RequestTimeout: The **RequestTimeout** member specifies the amount of time (in milliseconds) to wait for a message to be returned before returning a failure.

Action: The **Action** member specifies the type of receive operation that is to be performed. The **Action** member MUST specify one of the values: 0x00000000 (MQ_ACTION_RECEIVE), 0x80000000 (MQ_ACTION_PEEK_CURRENT), or 0x80000001 (MQ_ACTION_PEEK_NEXT).

Name	Value
MQ_ACTION_RECEIVE	0x00000000

Name	Value
MQ_ACTION_PEEK_CURRENT	0x80000000
MQ_ACTION_PEEK_NEXT	0x80000001

Asynchronous: The **Asynchronous** member is used as a Boolean variable to indicate if the receive is to be performed asynchronously. An **Asynchronous** member value of 0x00000000 MUST be interpreted as specifying FALSE (receive operation is not to be performed asynchronously) and all other values MUST be interpreted as TRUE (receive operation is to be performed asynchronously).

Cursor: A Cursor handle obtained from **rpc_ACCreateCursorEx**. A Cursor member can be used to reference a specific position within the message queue, rather than the first message in the queue, from which the message will be retrieved.

ulResponseFormatNameLen: The **ulResponseFormatNameLen** member specifies the size (in count of Unicode characters) of the string allocated for the **ppResponseFormatName** member. The **ulResponseFormatNameLen** member MUST have a value in the range of 0 to 1024, inclusive.

ppResponseFormatName: A NULL-terminated Unicode string containing a format name (as specified in [MS-MQMQ]) which indicates an application-defined queue which may be used for response messages. This value is used only by MSMQ applications, and it MUST be ignored by MSMQ servers.

pulResponseFormatNameLenProp: The **pulResponseFormatNameLenProp** member specifies the size (in count of Unicode characters) of the string contained in the **ppResponseFormatName** member.

ulAdminFormatNameLen: The **ulAdminFormatNameLen** member specifies the size (in count of Unicode characters) of the string allocated for the **ppAdminFormatName** member. The **ulAdminFormatNameLen** member MUST have a value in the range of 0 to 1024, inclusive.

ppAdminFormatName: A NULL-terminated Unicode string containing a format name (as specified in [MS-MQMQ]) which indicates an application-defined administration queue to which acknowledgment messages will be directed.

pulAdminFormatNameLenProp: The **pulAdminFormatNameLenProp** member specifies the size (in count of Unicode characters) of the string contained in the **ppAdminFormatName** member.

ulDestFormatNameLen: The **ulDestFormatNameLen** member specifies the size (in count of Unicode characters) of the string allocated for the **ppDestFormatName** member. The **ulDestFormatNameLen** member MUST have a value in the range of 0 to 1024, inclusive.

ppDestFormatName: A NULL-terminated Unicode string containing a format name (as specified in [MS-MQMQ]) which indicates the name of a message's destination queue.

pulDestFormatNameLenProp: The **pulDestFormatNameLenProp** member specifies the size (in count of Unicode characters) of the string contained in the **ppDestFormatName** member.

ulOrderingFormatNameLen: The **ulOrderingFormatNameLen** member specifies the size (in count of Unicode characters) of the string allocated for the **ppOrderingFormatName** member. The **ulOrderingFormatNameLen** member MUST have a value in the range of 0 to 1024, inclusive.

ppOrderingFormatName: A NULL-terminated Unicode string containing a format name (as specified in [MS-MQMQ]) which indicates the name of the MSMQ **order queue** that tracks the ordering of transactional messages.

pulOrderingFormatNameLenProp: The **pulOrderingFormatNameLenProp** member specifies the size (in count of Unicode characters) of the string contained in the **ppOrderingFormatName** member.

CreateCursor: The **CreateCursor** member contains information for creating a cursor which may be used when receiving messages from a queue. The **CreateCursor** member is present in the **CACTransferBufferV1** structure when the value of the **uTransferType** member is 0x00000002 (CACTB_CREATECURSOR). The **CreateCursor** member is not used by any of the methods defined by the qmcomm and qmcomm2 interfaces.

pClass: This field indicates the message classification, such as a positive acknowledgment, a system-generated report message, or a normal application-generated message. It contains a 16-bit structure as defined below:

See [MS-MQQB] section 2.2.1.5 for message class value definitions.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5
Class Code									Reserved				H	R	S

Value	Meaning
Class Code	Specifies the type of the acknowledgment. This field uniquely classifies the message type within the groupings defined by the fields described above. If the H bit is set, this field contains an HTTP status code.
Reserved 0x0000	MUST be set to all zeros. Clients and servers must ignore the **Reserved** member.
H 0x0000 — 0x0001	Specifies whether HTTP is being used or not. A value of 0 MUST be used to specify that HTTP is not being used and a value of 1 MUST be used to specify that HTTP is being used. If 1, the **Class Code** field contains an HTTP status code.
R 0x0000 — 0x0001	Specifies the stage at which the acknowledgment is to occur. A value of 0 MUST be used to specify that the acknowledgment is for the delivery (arrival) stage. A value of 1 MUST be used to specify that the acknowledgment is for the receive stage.
S 0x0000 — 0x0001	Specifies the type of acknowledgment. A value of 0 MUST be used to specify that normal (positive acknowledgment) message processing has occurred. A value of 1 MUST be used to specify that abnormal (negative acknowledgment) message processing has occurred.

ppMessageID: The **ppMessageID** member, if present, specifies a value that can be used to correlate response messages to sent messages.

ppCorrelationID: If present the **ppCorrelationID** member is an array of bytes containing an **OBJECTID** structure (as specified in [MS-MQMQ]). The **ppCorrelationID** member, if present, contains a value copied from the **ppMessageID** member of a previous request and can be used to correlate responses with previously sent messages. The size (in count of bytes) of **ppCorrelationID** MUST NOT exceed 20.

pSentTime: The **pSentTime** member is formatted in **UTC**. The **pSentTime** member specifies the time that the message was sent.

pArrivedTime: The **pArrivedTime** member is formatted in UTC. The **pArrivedTime** member specifies the time the message was received.

pPriority: The **pPriority** member is a single byte. The **pPriority** member specifies the processing priority for the message with larger values indicating a higher priority. The byte value MUST be in the range of 0x00 to 0x07. If no priority is set, the default priority value of 0x03 is used. The **pPriority** member is ignored for transactional messages. Messages that are not part of a transaction will be processed in arrival sequence within priority. The **pPriority** member is ignored if the message is a part of a transaction.

pDelivery: The **pDelivery** member is a single byte. The **pDelivery** member MUST specify a value of 0x00 or 0x01.

Value	Meaning
0x00	A value of 0x00 specifies that the message is not recoverable and can remain in volatile storage and is subject to loss in the event of a system crash.
0x01	A value of 0x01 specifies that the message is recoverable and is to be written to non-volatile storage as it moves through the network to its destination and can survive a system crash. Recoverable messages do not have to be part of a transaction.

pAcknowledge: The **pAcknowledge** member is a single byte. The **pAcknowledge** member value specifies the types of acknowledgment messages that are to be generated for this message. Acknowledgment messages are returned in the administration queue. The **pAcknowledge** member value MUST be assigned from the list below:

Value	Meaning
MQMSG_ACKNOWLEDGMENT_NONE 0x00	No acknowledgment needed.
MQMSG_ACKNOWLEDGMENT_POS_ARRIVAL 0x01	Positive acknowledgment is to be sent when the message is placed in the destination queue.
MQMSG_ACKNOWLEDGMENT_POS_RECEIVE 0x02	Positive acknowledgment is to be sent when the message is received from the destination queue.
MQMSG_ACKNOWLEDGMENT_NEG_ARRIVAL 0x04	Negative acknowledgment is to be sent when the message fails to arrive at the destination queue.
MQMSG_ACKNOWLEDGMENT_NACK_REACH_QUEUE	Negative acknowledgment is to be sent

Value	Meaning
0x04	when the message fails to arrive at the destination queue.
MQMSG_ACKNOWLEDGMENT_FULL_REACH_QUEUE 0x05	Positive acknowledgment is to be sent when the message is placed in the destination queue and/or negative acknowledgment is to be sent when the message fails to arrive at the destination queue.
MQMSG_ACKNOWLEDGMENT_NEG_RECEIVE 0x08	Negative acknowledgment is to be sent when the message fails to be received from the destination queue.
MQMSG_ACKNOWLEDGMENT_NACK_RECEIVE 0x0C	Negative acknowledgment is to be sent when the message fails to arrive at the destination queue or when a receive for the message from the destination queue fails.
MQMSG_ACKNOWLEDGMENT_FULL_RECEIVE 0x0E	Positive acknowledgment is to be sent when the message is received from the destination queue and a negative acknowledgment is to be sent when the message fails to arrive at the destination queue or a negative acknowledgment is to be sent when a receive for the message from the destination queue fails.

pAuditing: The **pAuditing** member is a single byte. The **pAuditing** member value specifies the conditions under which copies of the message are to be stored as the message is routed to the destination queue. The **pAuditing** member value MUST be assigned from the list below:

Value	Meaning
MQMSG_JOURNAL_NONE 0x00	Do not store copies.
MQMSG_DEADLETTER 0x01	Store copy in **dead-letter queue** on failure.
MQMSG_JOURNAL 0x02	Store copy in **queue journal** upon successful delivery to next computer.
MQMSG_DEADLETTER|MQMSG_JOURNAL 0x03	Store copy in queue journal upon successful delivery to next computer. Store copy in dead-letter queue on failure.

pApplicationTag: The **pApplicationTag** member value is a user-provided item that is passed through unmodified to the message receiving application. A common use of the **pApplicationTag** member value is to indicate to the receiving application the type of data contained in the **ppMsgExtension** member.

ppBody: The **ppBody** member is an array of bytes. When the **ppBody** member is present it contains the user message payload.

ulBodyBufferSizeInBytes: The **ulBodyBufferSizeInBytes** member specifies the size (in count of bytes) of the data present in the **ppBody** member. The value of the

ulBodyBufferSizeInBytes member must be less than or equal to the value in the **ulAllocBodyBufferInBytes** member.

ulAllocBodyBufferInBytes: The **ulAllocBodyBufferInBytes** member specifies the size (in count of bytes) of the buffer that is allocated to contain the **ppBody** member.

pBodySize: The **pBodySize** member specifies the size (in count of bytes) of the data present in the **ppBody** member after an encryption or decryption operation has been performed on the **ppBody** member. The value of the **pBodySize** member must be less than or equal to the value in the **ulAllocBodyBufferInBytes** member.

ppTitle: The **ppTitle** member, when present, is a Unicode string. The **ppTitle** member specifies a title associated with the message.

ulTitleBufferSizeInWCHARs: The **ulTitleBufferSizeInWCHARs** member specifies the size (in count of Unicode characters) of the **ppTitle** member. The **ulTitleBufferSizeInWCHARs** member MUST NOT exceed 250.

pulTitleBufferSizeInWCHARs: The **pulTitleBufferSizeInWCHARs** member specifies the actual size (in count of Unicode characters) of the string, if present, in the **ppTitle** member Unicode string.

ulAbsoluteTimeToQueue: The **ulAbsoluteTimeToQueue** member value provided by the client specifies the number of seconds within which the message must reach the destination queue or be discarded. Internally, **ulAbsoluteTimeToQueue** is converted to a UTC time using the clock of the system on which the queue manager is executing.

pulRelativeTimeToQueue: The **pulRelativeTimeToQueue** member specifies the number of seconds within which the response message must reach the destination queue or be discarded.

ulRelativeTimeToLive: The **ulRelativeTimeToLive** member value specifies the number of seconds within which the message must be received from the destination queue or be discarded. Internally, **ulRelativeTimeToLive** is converted to a UTC time using the clock of the system on which the queue manager is executing.

pulRelativeTimeToLive: The **pulRelativeTimeToLive** member specifies the number of seconds remaining before the response message will be discarded if it is not received from the destination queue.

pTrace: The **pTrace** member must be a single byte and indicates whether or not tracing is active.

Value	Meaning
0x00	A value of 0x00 MUST be used to specify that tracing is not active.
0x01	A value of 0x01 MUST be used to specify that tracing is active.

pulSenderIDType: The **pulSenderIDType** member MUST specify the type of the **ppSenderID** member contents. The **pulSenderIDType** member value MUST be assigned from the list below:

Value	Meaning
MQMSG_SENDERID_TYPE_NONE 0x00000000	No sender ID is present.
MQMSG_SENDERID_TYPE_SID 0x00000001	The sender ID is a SID.
MQMSG_SENDERID_TYPE_QM 0x00000002	The sender ID is the GUID assigned to a queue manager.

ppSenderID: The **ppSenderID** member must be an array of bytes. When the value of the **pulSenderIDType** member is 0x00000000 (MQMSG_SENDERID_TYPE_NONE), the **ppSenderID** member MUST NOT be present. If the value of the **pulSenderIDType** member is 0x00000001 (MQMSG_SENDERID_TYPE_SID), the **ppSenderID** member MUST contain a SID. If the value of the **pulSenderIDType** member is 0x00000002 (MQMSG_SENDERID_TYPE_QM), the **ppSenderID** member MUST contain a valid MSMQ Site GUID.

pulSenderIDLenProp: The **pulSenderIDLenProp** member specifies the size (in count of bytes) of the data actually present in the **ppSenderID** member.

pulPrivLevel: The **pulPrivLevel** member specifies the privacy level that is used for processing the message. The **pulPrivLevel** member value MUST be assigned from the list below:

Value	Meaning
MQMSG_PRIV_LEVEL_NONE 0x00000000	The message is not private.
MQMSG_PRIV_LEVEL_BODY_BASE 0x00000001	The message is private and the Microsoft Cryptographic Service Provider (CSP) will use a 40-bit encryption key to encrypt and decrypt the message body.
MQMSG_PRIV_LEVEL_BODY_ENHANCED 0x00000002	The message is private and the Microsoft CSP will use a 128-bit encryption key to encrypt and decrypt the message body.

ulAuthLevel: The **ulAuthLevel** member is used only in local interprocess communication, and therefore has no meaning when this protocol is used over a network. Servers MUST ignore this field, and clients may specify any value.

pAuthenticated: The **pAuthenticated** member is a single byte. The **pAuthenticated** member value is used to determine the **level of authentication** that has been performed on the message. The **pAuthenticated** member value MUST be assigned from the list below:

Value	Meaning
MQMSG_AUTHENTICATION_NOT_REQUESTED 0x00	Authentication has not been performed.
MQMSG_AUTHENTICATED_SIG10 0x01	Authentication has been performed using an MSMQ 1.0 digital signature.
MQMSG_AUTHENTICATED_SIG20 0x03	Authentication has been performed using an MSMQ 2.0 digital signature.

Value	Meaning
MQMSG_AUTHENTICATED_SIG30 0x05	Authentication has been performed using an MSMQ 3.0 digital signature.
MQMSG_AUTHENTICATED_SIGXML 0x09	Authentication has been performed using an XML digital signature

pulHashAlg: The **pulHashAlg** member specifies the hashing algorithm that is to be used in the digital signing process and by the authentication process. The **pulHashAlg** member value MUST be assigned from the list below:

Value	Meaning
MQMSG_CALG_MD2 0x00008001	Use the MD2 hashing algorithm as specified in [RFC1319].
MQMSG_CALG_MD4 0x00008002	Use the MD4 hashing algorithm as specified in [RFC1320].
MQMSG_CALG_MD5 0x00008003	Use the MD5 hashing algorithm as specified in [RFC1321].
MQMSG_CALG_SHA1 0x00008004	Use the SHA-1 hashing algorithm as specified in [RFC3174].

pulEncryptAlg: The **pulEncryptAlg** member specifies that the encryption algorithm is to be used to encrypt and decrypt the message body. The **pulEncryptAlg** member value MUST be assigned from the list below:

Value	Meaning
MQMSG_CALG_RC2 0x00006602	Use the RC2 encryption algorithm as specified in [RFC2268].
MQMSG_CALG_RC4 0x00006801	Use the RC4 encryption algorithm as specified in [RC4].

ppSenderCert: The **ppSenderCert** member is an array of bytes. If not NULL, the **ppSenderCert** member MUST contain the message sender's X509 certificate. The byte length of the buffer MUST be indicated by **ulSenderCertLen**.

ulSenderCertLen: The **ulSenderCertLen** member specifies the byte length of the certificate contained in **ppSenderCert**.

pulSenderCertLenProp: The **pulSenderCertLenProp** member specifies the length (in count of bytes) of the certificate contained in **ppSenderCert**.

ppwcsProvName: The **ppwcsProvName** member is a Unicode string. If present the **ppwcsProvName** member specifies the name of the **Cryptographic Service Provider (CSP)** that is used to generate digital signatures for the message.

ulProvNameLen: The **ulProvNameLen** member specifies the size (in count of Unicode characters) of the buffer that was allocated to contain the **ppwcsProvName** string.

pulAuthProvNameLenProp: The **pulAuthProvNameLenProp** member specifies the size (in count of Unicode characters) of the CSP name contained in **ppwcsProvName**, plus the size of an enhanced signature appended to the **ppSignature** buffer. Rules for computing and understanding values for this field are defined in sections 3.1.5.3 and 3.1.5.4.

pulProvType: The **pulProvType** member specifies the type of CSP that is named by **ppwcsProvName**.<4>

fDefaultProvider: The **fDefaultProvider** member specifies if the CSP named by **ppwcsProvName** is a default CSP. A value of 0x00000000 MUST be used to specify that the **ppwcsProvName** is not the default name and all other values MUST be interpreted as specifying that the **ppwcsProvName** is the default name.

ppSymmKeys: The **ppSymmKeys** member is an array of bytes. The **ppSymmKeys** member, if present, contains an encrypted symmetric key.

ulSymmKeysSize: The **ulSymmKeysSize** member specifies the size (in count of bytes) of the buffer that was allocated to contain the **ppSymmKeys** member.

pulSymmKeysSizeProp: The **pulSymmKeysSizeProp** member specifies the size (in count of bytes) of the **ppSymmKeys** member.

bEncrypted: The **bEncrypted** member is a single byte. The **bEncrypted** member specifies if the message body is encrypted or is not encrypted. A **bEncrypted** member value of 0x00 MUST be interpreted as specifying that the message is not encrypted (FALSE) and all other values MUST be interpreted as specifying that the message is encrypted (TRUE).

bAuthenticated: The **bAuthenticated** member is a single byte. The **bAuthenticated** member specifies if the message has been authenticated or has not been authenticated. A **bAuthenticated** member value of 0x00 MUST be used to specify that the message has not been authenticated (FALSE) and all other values MUST be interpreted as specifying that the message has been authenticated (TRUE).

uSenderIDLen: The **uSenderIDLen** member specifies the maximum size (in count of bytes) that is available to contain data in the **ppSenderID** member.

ppSignature: The **ppSignature** member is an array of bytes. The **ppSignature** member contains the signature(s) used to authenticate the message.<5>

ulSignatureSize: The **ulSignatureSize** member specifies the size (in count of bytes) allocated to hold the **ppSignature** member.

pulSignatureSizeProp: The **pulSignatureSizeProp** member specifies the size (in count of bytes) of the authentication signature(s) in the **ppSignature** member.

ppSrcQMID: The **ppSrcQMID** member is a GUID. The member contains the GUID assigned to the MSMQ installation that is the source of the message.

pUow: The **pUow** member is an **XACTUOW** structure as defined above in section 2.2.2.1 If not NULL, this field identifies a transaction for a Send or Receive operation.

ppMsgExtension: The **ppMsgExtension** member is an array of bytes. The **ppMsgExtension** member, when present, contains application-specific data. The **ppMsgExtension** member is primarily used to pass information to **foreign queues**.

ulMsgExtensionBufferInBytes: The **ulMsgExtensionBufferInBytes** member specifies the size (in count of bytes) of the buffer allocated for the **ppMsgExtension** array.

pMsgExtensionSize: The **pMsgExtensionSize** member specifies the size (in count of bytes) of the data actually contained in the **ppMsgExtension** array.

ppConnectorType: The **ppConnectorType** member, if present, is a GUID. The **ppConnectorType** member specifies the identifier of a foreign queue that is used to communicate with a foreign messaging system.

pulBodyType: The **pulBodyType** member value MUST be one of the valid values allowed for a **VARTYPE** as specified in [MS-MQMQ].

pulVersion: The **pulVersion** member specifies the MSMQ packet version.<6>

2.2.2.3 CACTransferBufferV2

The **CACTransferBufferV2** structure is used to send and receive messages via MSMQ.

Following is the layout of the **CACTransferBufferV2** structure followed by descriptions of the structure members.

```
typedef struct {
  struct CACTransferBufferV1 old;
  unsigned char* pbFirstInXact;
  unsigned char* pbLastInXact;
  OBJECTID** ppXactID;
} CACTransferBufferV2;
```

old: The **CACTransferBufferOld** MUST be a **CACTransferBufferV1**, as defined in section 2.2.2.2.

pbFirstInXact: The **pbFirstInXact** member MUST be a single byte. The *pbFirstInXact* member MUST be set to a value of 0x00 (FALSE) when the associated message is not the first message in a transaction. A value other than 0x00 MUST be interpreted as indicating (TRUE) that the associated message is the first message in a transaction.

pbLastInXact: The **pbLastInXact** member MUST be a single byte. The **pbLastInXact** member MUST be set to a value of 0x00 (FALSE) when the associated message is not the last message in a transaction. A value other than 0x00 MUST be interpreted as indicating (TRUE) that the associated message is the last message in a transaction.

ppXactID: The **ppXactID** member, if present, MUST be an OBJECTID structure, as specified in [MS-MQMQ] section 2.2.8.

2.2.2.4 CACCreateRemoteCursor

The **CACCreateRemoteCursor** structure contains the elements necessary for creating a cursor on a queue.

```
typedef struct {
  DWORD hCursor;
  DWORD srv_hACQueue;
  DWORD cli_pQMQueue;
} CACCreateRemoteCursor;
```

hCursor: The **hCursor** member MUST be a handle to a cursor object.

srv_hACQueue: The **srv_hACQueue** member is a handle to a queue object on the remote computer. The value for this field returned from rpc_ACCreateCursorEx is passed to the *hQueue* parameter of **R_QMCreateRemoteCursor (section 3.1.4.4)** when invoked as part of a remote cursor creation call sequence.

cli_pQMQueue: The **cli_pQMQueue** member is a queue context value which identifies the queue for which the cursor is associated. The value for this field returned from rpc_ACCreateCursorEx is passed to the *pQueue* parameter of **R_QMGetRemoteQueueName** (as specified in 3.1.4.1) when invoked as part of a remote cursor creation call sequence.

2.2.2.5 OBJECT_FORMAT

An **OBJECT_FORMAT** structure wraps a pointer to a QUEUE_FORMAT structure, as specified in [MS-MQMQ].

```
typedef struct OBJECT_FORMAT {
  [range(1,2)] DWORD ObjType;
  [switch_is(ObjType)] union {
    [case(1)]
      struct QUEUE_FORMAT* pQueueFormat;
  };
} ;
```

ObjType: This value MUST be 0x00000001. The value 0x00000002 is defined for local-only use and MUST NOT appear on the wire.

pQueueFormat: This MUST point to a QUEUE_FORMAT structure, as specified in [MS-MQMQ].

3 Protocol Details

The client side of this protocol is simply a pass-through. That is, there are no additional timers or other states required on the client side of this protocol. Calls made by the higher-layer protocol or 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.

3.1 Qmcomm and Qmcomm2 Server Details

3.1.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

Servers MUST maintain the following data elements described in the following sections:

- Queue Table (section 3.1.1.1)
- Message List (section 3.1.1.2)
- QueueContextHandle Table (section 3.1.1.3)
- Cursor Table (section 3.1.1.4)
- Transaction Table (section 3.1.1.5)
- Transactional Operation List (section 3.1.1.6)
- Outstanding Receive List (section 3.1.1.7)

3.1.1.1 Queue Table

A table of queues deployed at the server, keyed by name. Each entry MUST contain:

- The name of the queue.
- A flag indicating if the queue is an **outgoing queue**.
- The current queue state, as defined in the queue state diagram in section 3.1.1.8.
- The queue shared open count, referred to as OpenCount in the queue state diagram in section 3.1.1.8.
- A table of queue properties, as specified in [MS-MQMQ] section 2.3.1.
- A SECURITY_DESCRIPTOR, as specified in [MS-DTYP] section **2.4.6**.
- A message list.

3.1.1.2 Message List

A list of messages in a queue, in ascending order by arrival time within message priority. Each entry MUST contain:

- A Lock flag indicating if the message is locked by a pending transactional operation.
- A table of message properties, as specified in [MS-MQMQ].

3.1.1.3 QueueContextHandle Table

A table of queue context handles, represented by the **RPC_QUEUE_HANDLE** or **PCTX_OPENREMOTE_HANDLE_TYPE** data type, keyed by the handle value. Each entry MUST contain:

- The context handle for the opened queue (**RPC_QUEUE_HANDLE** or **PCTX_OPENREMOTE_HANDLE_TYPE**).

  A client may first obtain a queue context handle of type **PCTX_OPENREMOTE_HANDLE_TYPE** from an arbitrary remote **MSMQ server**. This handle is in turn used to obtain a handle to that same queue of type **RPC_QUEUE_HANDLE**. This second handle is obtained from the client's supporting MSMQ Queue Manager (not on the same machine as the remote server). The handle of type **RPC_QUEUE_HANDLE** references a **remote queue** from the perspective of the supporting queue manager abstract data model (ADM). The associated handle of type **PCTX_OPENREMOTE_HANDLE_TYPE** references that same queue as a local queue from the perspective of the foreign server ADM. Any direct association between two such handles to the same queue is maintained by the client application and is not in either queue manager. See [MS-MQQP] for additional ADM elements that support association through **session** handles. A handle of type **PCTX_OPENREMOTE_HANDLE_TYPE** is typically kept only long enough to create a session handle at the remote queue manager.

  Thus, while handles of the **RPC_QUEUE_HANDLE** type may reference either local or remote queues in a queue manager ADM, handles of type **PCTX_OPENREMOTE_HANDLE_TYPE** may reference only local queues in a queue manager ADM. If a valid queue context handle of type **RPC_QUEUE_HANDLE** references a remote queue in some queue manager ADM, there must have simultaneously been a valid queue context handle of type **PCTX_OPENREMOTE_HANDLE_TYPE** to that same queue in the ADM of the remote queue manager when it was created. Both queue manager ADMs must have the remote queue session handle.

- A Queue Context Value **DWORD** that MUST uniquely identify the QueueContextHandle table entry. This element MAY <7> contain the same value as the **RPC_QUEUE_HANDLE** element. This value is supplied to the client as an out-parameter from **rpc_QMOpenQueueInternal**. Clients use this element to uniquely identify a queue when calling **rpc_ACReceiveMessageEx**.
- The Access Mode specified when the queue was opened.
- The Share Mode specified when the queue was opened.
- A reference which, when followed, enables inspection or manipulation of the represented queue, which may exist either at the local server or at a remote server. The form of the reference differs depending on the location of the represented queue.
  - A QueueContextHandle that represents a local queue MUST maintain the following elements:
    - A reference to the queue table entry, reserving a value to indicate that the associated queue has been deleted. Operations specifying this QueueContextHandle will affect the associated queue.
  - A QueueContextHandle that represents a remote queue MUST maintain the following elements:

- **RemoteQueueHandle**: A **PCTX_RRSESSION_HANDLE_TYPE** context handle obtained from the **RemoteQMOpenQueue** method of the [MS-MQQP] protocol. Message Receive operations specifying this **QueueContextHandle** are delegated to the remote queue. Note that the **PCTX_RRSESSION_HANDLE_TYPE** context handle should be explicitly closed by invoking the **RemoteQMCloseQueue** method of [MS-MQQP] prior to discarding this entry from the table. Failure to proactively close a **PCTX_RRSESSION_HANDLE_TYPE** results in delayed resource reclamation on the [MS-MQQP] server.

- **RemoteQueueBindingHandle**: A binding handle that refers to the client binding information for the server from which **RemoteQueueHandle** is obtained.

3.1.1.4 Cursor Table

A table of cursor handles, keyed by a cursor handle and **RPC_QUEUE_HANDLE** key pair. Note that entries in the Cursor Table may be shared with an implementation of [MS-MQQP], in which case the [MS-MQQP] implementation deletes entries from the Cursor Table in response to the **RemoteQMCloseCursor (Opnum 4)** method, as specified in [MS-MQQP] section **3.1.4.5**.

Each entry MUST contain:

- The cursor handle.
- The **RPC_QUEUE_HANDLE** specified when the cursor was created.
- A flag to indicate if this cursor is a local, or a proxy for a remote cursor handle.
- If the cursor is local, the following fields are necessary:
  - The current position of the cursor within the queue.
  - The current cursor state, as defined in the cursor state diagram in section 3.1.1.9.
- If the cursor is a proxy for a remote cursor, the following field is necessary:
  - A remote cursor handle. This value MAY be NULL if the Cursor has been created, but not associated with a remote cursor handle.

3.1.1.5 Transaction Table

A table of pending transaction, keyed by transaction identifier (**XACTUOW**). Each entry MUST contain:

- The transactions identifier (**XACTUOW**).
- For internal transactions, an **RPC_INT_XACT_HANDLE** context handle for the transaction.
- A transactional operation list.

3.1.1.6 Transactional Operation List

A FIFO-ordered list of operations to be performed within a specific transaction. Each entry MUST contain:

- The type of operation. One of the following:
  - SendMessage (initiated by rpc_ACSendMessageEx, described in section 3.1.5.2).

- ReceiveMessage (initiated by rpc_ACRecieveMessageEx, described in section 3.1.5.3).

- The client-provided CACTransferBuffer structure containing the properties of the message to be sent, or to receive the contents of a message being retrieved.
- The client-provided QueueContextHandle identifying the queue for which the operation is to be performed.
- If the operation requires a lock on a message in a queue, a reference to the message.

3.1.1.7 Outstanding Receive List

A FIFO-ordered list of **rpc_ACReceiveMessageEx** requests which were not immediately satisfied because messages were not present in the queue and the client specified a nonzero time-out. Each entry MUST contain:

- Parameters necessary to perform the receive or peek operation, as specified by the **rpc_ACReceiveMessageEx** invocation.
- A QueueContextHandle identifying the queue from which the message is to be read.
- An expiration time, determined from the ptb.old.Receive.RequestTimeout parameter of **rpc_ACReceiveMessageEx**, indicating when the server will cease waiting for a message to be added to the queue, then inform the client that a message was not available.

3.1.1.8 Queue State Diagram

The following diagram specifies the state transitions for a queue. In particular, it specifies the queue state transitions resulting from open and close events, and how the sharing mode parameter specified on open affects these state transitions. Successful open queue operations where the share mode is not exclusive increment the OpenCount counter. Successful close queue operations when the current state is Share Open decrement the OpenCount counter.

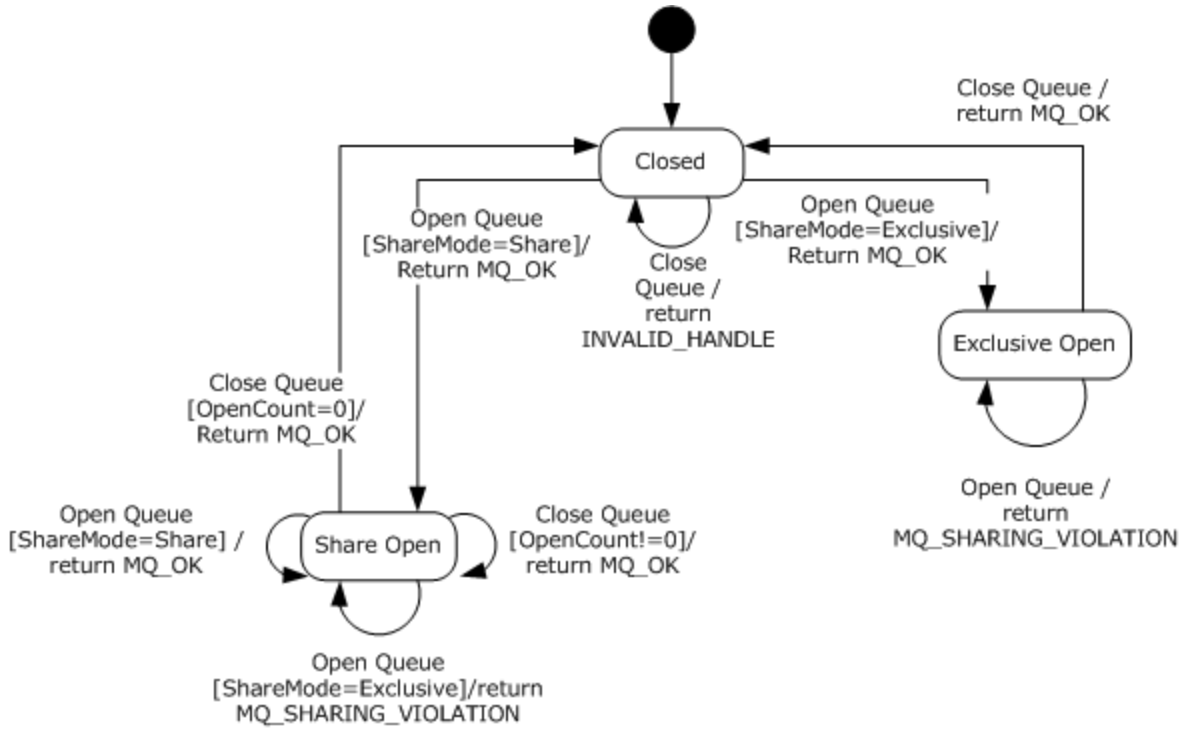

Figure 1: Open and close event transitions

For more information about the implementation of the state transitions: In the case of opening and closing a local queue, see section 4.1 for an example of the protocol sequence diagram.

- For more information about opening a local queue, see section 3.1.4.17 for rpc_QMOpenQueueInternal.
- For more information about closing a local queue, see section 3.1.4.18 for rpc_ACCloseHandle.

In the case of opening and closing a remote queue, see section 4.2 for an example of the protocol sequence diagram.

- For more information about opening a remote queue, see section 3.1.4.17 for rpc_QMOpenQueueInternal, section 3.1.4.2 for R_QMOpenRemoteQueue, and [MS-MQQP] section 3.1.4.3 for RemoteQMOpenQueue.
- For more information about closing a remote queue, see section 3.1.4.3 for R_QMCloseRemoteQueueContext, section 3.1.4.18 for rpc_ACCloseHandle, and [MS-MQQP] section 3.1.4.4 for RemoteQMCloseQueue.

3.1.1.9 Cursor State Diagram

The following diagram specifies the state transitions for a cursor. A cursor represents position in a queue. After a cursor is created, changes to its state are triggered only by the method **rpc_ACReceiveMessageEx**, from which three separate events can result: PeekCurrent, PeekNext, and Receive. Note that the state of the underlying queue is free to change with no effect on the associated cursors. Upon invocation of **rpc_ACReceiveMessageEx**, the state of the underlying queue is evaluated by the conditions specified below, and the cursor state is changed accordingly.

The diagram specifies how cursor state transitions are affected by PeekCurrent, PeekNext, and Receive events, as well as by the presence or absence of messages in the queue.

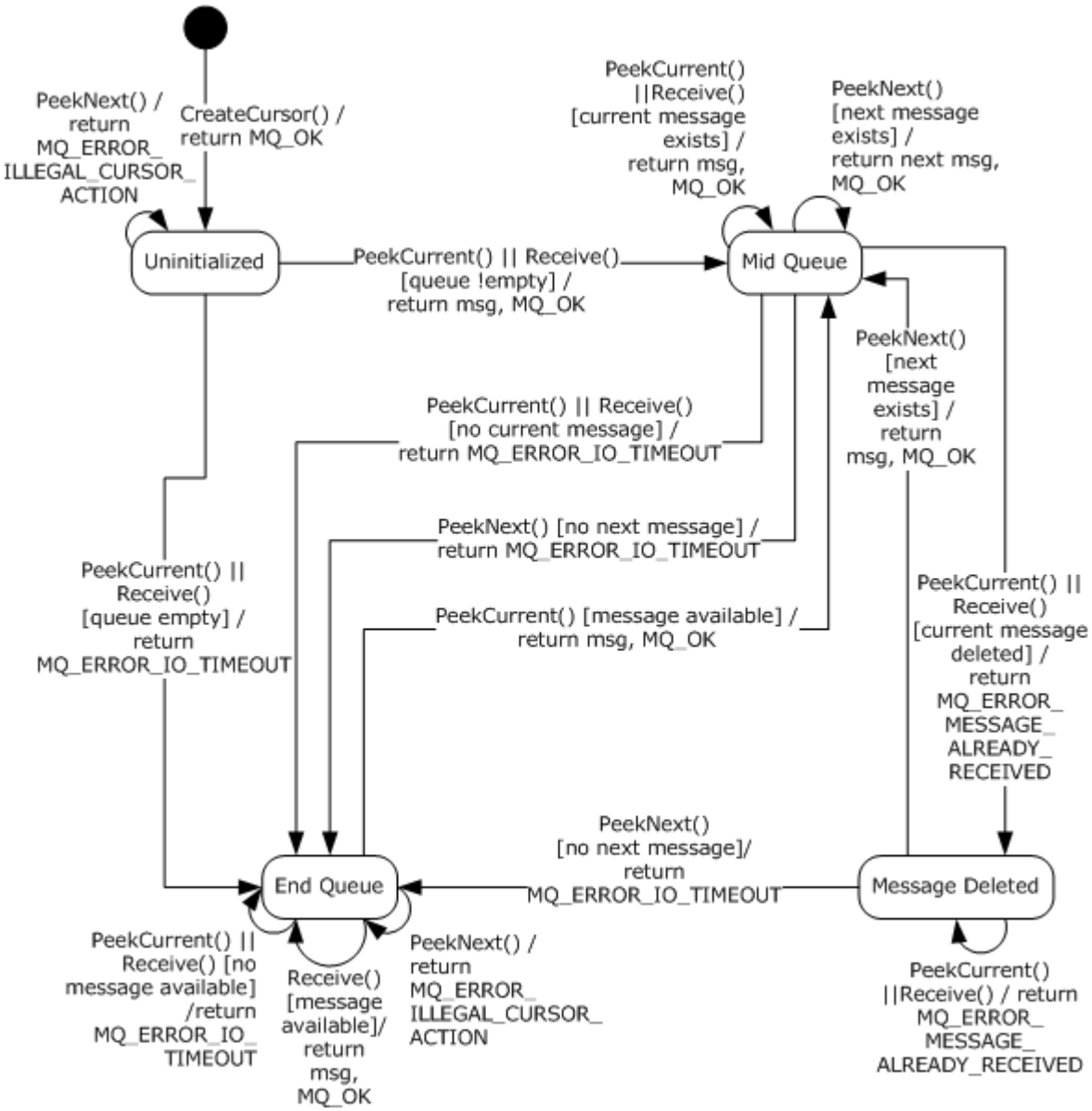

Figure 2: Cursor state transitions

Note that a cursor may be closed from any state by calling **rpc_ACCloseCursor**. For simplicity, the "closed" state is omitted from the diagram.

The cursor state is transitioned by calls to **rpc_ACReceiveMessageEx**. The parameters that are provided to **rpc_ACReceiveMessageEx** and the condition of the queue associated with the cursor determine which transition occurs. The following expressions and constants from the diagram are defined.

CreateCursor(): The method **rpc_ACCreateCursorEx** .

Receive(): The method **rpc_ACReceiveMessageEx** is called with *ptb.old.Receive.Action* equal to MQ_ACTION_RECEIVE (0x00000000).

PeekCurrent(): The method **rpc_ACReceiveMessageEx** is called with ptb.old.Receive.Action equal to MQ_ACTION_PEEK_CURRENT (0x80000000).

PeekNext(): The method **rpc_ACReceiveMessageEx** is called with ptb.old.Receive.Action equal to MQ_ACTION_PEEK_NEXT (0x80000001).

Current Message Exists: A condition where the cursor is pointing to a message.

No Current Message: A condition that results when the cursor is advanced, but no message exists at the new location; therefore, the cursor is not pointing to any message.

Next Message Exists: A condition where a message is available in the location immediately following the message currently identified by the cursor.

No Next Message: A condition where no message is available in the location immediately following the message that is currently identified by the cursor.

Queue Empty: A condition where the Message List for the Queue associated with the cursor contains zero messages.

Message Available: A condition where a message has become available at, or after, the current cursor position. Messages are added to a queue by **QMSendMessageInternalEx**, **rpc_ACSendMessageEx**, and by the process specified in [MS-MQQB] section 3.1.5.8.8.

No Message Available: A condition where no message is available at, or after, the current cursor position.

Current Message Deleted: A condition where the message that is previously identified by the cursor was deleted and is not available. Messages are deleted by calls to **rpc_ACReceiveMessageEx**, **rpc_ACPurgeQueue**, and R_EndReceive ([MS-MQRR] section 3.1.4.9).

The following error codes are specified in [MS-MQMQ] section 2.7.

- MQ_ERROR_ILLEGAL_CURSOR_ACTION (0xC00E001C)
- MQ_ERROR_IO_TIMEOUT (0xC00E001B)
- MQ_ERROR_MESSAGE_ALREADY_RECEIVED (0xC00E001D)

3.1.2 Timers

The server MUST maintain a timer for each call to **rpc_ACReceiveMessageEx (section 3.1.5.3)**, if the ptb.old.Receive.RequestTimeout method parameter is non-zero.

3.1.3 Initialization

The server MUST initialize the queue table from persistent storage. The server MAY use techniques such as demand-loading table entries when queues are first opened, and paging out unused entries, as long as the abstraction presented by the queue table is logically equivalent to having created one entry for each queue located at the server.

The server MUST listen on the RPC protocols, as specified in section 2.1.

3.1.4 Message Processing Events and Sequencing Rules for Qmcomm

This protocol SHOULD indicate to the RPC run time that it is to perform a strict **NDR** data consistency check at target level 6.0, as specified in [MS-RPCE] section 3.<8>

Note Gaps in the **opnum** numbering sequence represent local-only RPC methods that, for security reasons, MUST NOT be used over the wire.<9>

Methods in RPC Opnum Order

Method	Description
R_QMGetRemoteQueueName	Retrieves the name of the queue associated with the given **RPC_QUEUE_HANDLE**. Opnum: 1
R_QMOpenRemoteQueue	Opens a queue for remote read. Opnum: 2
R_QMCloseRemoteQueueContext	Closes a **PCTX_OPENREMOTE_HANDLE_TYPE**. Opnum: 3
R_QMCreateRemoteCursor	Creates a cursor for a remote queue. Opnum: 4
R_QMCreateObjectInternal	Creates a local **private queue**. Opnum: 6
R_QMSetObjectSecurityInternal	Updates the security configuration of a local private queue. Opnum: 7
R_QMGetObjectSecurityInternal	Retrieves the security configuration of a local private queue. Opnum: 8
R_QMDeleteObject	Deletes a local private queue. Opnum: 9
R_QMGetObjectProperties	Retrieves queue properties from local private queues. Opnum: 10
R_QMSetObjectProperties	Updates queue properties of local private queues. Opnum: 11
R_QMObjectPathToObjectFormat	Returns a complete format name for a private queue when only the **path name** is known to the caller. Opnum: 12
R_QMGetTmWhereabouts	Returns transaction manager whereabouts information. Opnum: 14
R_QMEnlistTransaction	Enlists the supporting server **resource manager** in an external transaction. Opnum: 15
R_QMEnlistInternalTransaction	Enlists the supporting server resource manager in an internal

Method	Description
	transaction. Opnum: 16
R_QMCommitTransaction	Commits an internal transaction. Opnum: 17
R_QMAbortTransaction	Aborts an internal transaction. Opnum: 18
rpc_QMOpenQueueInternal	Opens a queue for sending, reading, or purging messages. Opnum: 19
rpc_ACCloseHandle	Closes a queue handle. Opnum: 20
rpc_ACCloseCursor	Closes a cursor handle. Opnum: 22
rpc_ACSetCursorProperties	Associates a remote cursor handle with a local cursor handle. Opnum: 23
rpc_ACHandleToFormatName	Retrieves a format name for a queue handle. Opnum: 26
rpc_ACPurgeQueue	Purges an opened queue. Opnum: 27
R_QMQueryQMRegistryInternal	Retrieves string values from a supporting server. Opnum: 28
R_QMGetRTQMServerPort	Returns the RPC server port for use in subsequent method calls. Opnum: 31

3.1.4.1 R_QMGetRemoteQueueName (Opnum 1)

During the process of creating a remote cursor handle, a dependent client calls the **R_QMGetRemoteQueueName** method to retrieve the name of the remote queue associated with a queue handle. Note that the remote cursor creation process was revised for Windows Server 2003, Windows Vista, and Windows Server 2008 operating system; therefore, this method is obsolete, and the server SHOULD take no action and immediately raise the exception MQ_ERROR_ILLEGAL_OPERATION (0xc00e0064).<10> Behavior for Windows NT and Windows 2000 servers is defined below.

```
HRESULT R_QMGetRemoteQueueName(
  [in] handle_t hBind,
  [in] DWORD pQueue,
  [in, out, ptr, string] WCHAR** lplpRemoteQueueName
);
```

hBind: MUST be set to an RPC binding handle as described in [MS-RPCE] section 2.

pQueue: MUST be a DWORD that contains a local queue context value obtained from the **cli_pQMQueue** member of the structure returned by the **rpc_ACCreateCursorEx** method of the qmcomm2 interface. See section 4.4 for an example illustrating this value being obtained.

lplpRemoteQueueName: A pointer to a buffer to receive the NULL-terminated name of the remote queue associated with the pQueue context value.

Return Values: On success, this method MUST return MQ_OK (0x00000000); otherwise, the server MUST return a failure HRESULT, and the client MUST treat all failure HRESULTs identically. Additionally, if a failure HRESULT is returned, the client MUST disregard all out-parameter values.

Exceptions Thrown: Windows Server 2003 and Windows Server 2008: This method takes no action and immediately raises the exception MQ_ERROR_ILLEGAL_OPERATION (0xc00e0064).<11>

During the remote cursor creation sequence, the supporting server MAY indicate that the client MUST contact a remote queue manager to proceed.<12> In response, this method is called by the client to determine where to find the remote queue manager. Supporting servers SHOULD contact the remote queue manager on behalf of the dependent client, thus eliminating the purpose of this method.<13>

This method is called only by dependent clients. Servers SHOULD enforce limitations and security measures appropriate for dependent clients.<14>

This method is invoked at the dynamically assigned endpoint returned by the **R_QMGetRTQMServerPort** method when IP_HANDSHAKE (0x00000000) or IPX_HANDSHAKE (0x00000002) is the interface specified by the *fIP* parameter.

When processing this call, the server MUST do the following:

- Determine if input parameter values violate constraints specified above. If an invalid parameter is detected, the server MUST take no further action and return MQ_ERROR_INVALID_PARAMETER (0xc00e0006).
- Locate the entry in the queue table associated with the pQueue value. The key for the association is returned in the pdwQMContext out-parameter of the **rpc_QMOpenQueueInternal** method when the queue is opened.
- If found, set lplpRemoteQueueName to the queue name from the queue table entry and return MQ_OK (0x00000000).
- Else: return a failure HRESULT.

3.1.4.2 R_QMOpenRemoteQueue (Opnum 2)

The **R_QMOpenRemoteQueue** method opens a queue for remote read.

```
HRESULT R_QMOpenRemoteQueue(
  [in] handle_t hBind,
  [out] PCTX_OPENREMOTE_HANDLE_TYPE* pphContext,
  [out] DWORD* pdwContext,
  [in, unique] struct QUEUE_FORMAT* pQueueFormat,
  [in] DWORD dwCallingProcessID,
  [in] DWORD dwDesiredAccess,
  [in] DWORD dwShareMode,
  [in] GUID* pLicGuid,
  [in] DWORD dwMQS,
```

```
  [out] DWORD* dwpQueue,
  [out] DWORD* phQueue
);
```

hBind: MUST be set to an RPC binding handle as described in [MS-RPCE] section 2.

pphContext: A pointer to a variable to receive the queue context handle. The server MUST return the same value in *pdwContext*, *dwpQueue*, and *pphContext*.

pdwContext: A pointer to a variable to receive the queue context value, which MUST be the same as *pphContext*. MUST NOT be NULL.

pQueueFormat: A QUEUE_FORMAT structure as defined in [MS-MQMQ] section **2.2.7** that identifies the queue to be opened. MUST NOT be NULL, and MUST conform to the format name syntax rules defined in [MS-MQMQ].

dwCallingProcessID: MUST be ignored. Clients MAY pass 0x00000000.<15>

dwDesiredAccess: A DWORD that specifies the access mode requested for the queue. The access mode defines the set of operations that can be invoked using the returned queue handle. The value MUST be one of the following:

Value	Meaning
MQ_RECEIVE_ACCESS 0x00000001	The returned queue handle MUST only permit message peek, message receive (peek and delete), and queue purge operations.
MQ_PEEK_ACCESS 0x00000020	The returned queue handle MUST only permit message peek operations.

dwShareMode: Specifies the exclusivity level for the opened queue. The value MUST be one of the following:

Value	Meaning
MQ_DENY_NONE 0x00000000	The queue is not opened exclusively.
MQ_DENY_RECEIVE_SHARE 0x00000001	The queue is to be opened for exclusive read access. If the queue has already been opened for read access, the server MUST return STATUS_SHARING_VIOLATION (0xc0000043). If the queue is opened successfully for exclusive read access, subsequent attempts to open the same queue for read access MUST return STATUS_SHARING_VIOLATION (0xc0000043) until the queue has been closed.

pLicGuid: MUST be a pointer to a valid GUID that uniquely identifies the client.<16><17> The server MAY ignore this parameter.<18>

dwMQS: MUST be set by clients to indicate the client operating system category. Servers MAY ignore this value.<19> The following values are defined:

Are server connection licensing limitations enforced?/Value	Meaning
Yes 0x00000000<20>	None. The operating system (OS) version is not declared.
Yes 0x00000001	A non-Microsoft OS.
Yes 0x00000002	Any edition of Windows 95, Windows 98, or Windows Me.
Yes 0x00000003	Windows NT Workstation, Windows 2000 Workstation, Windows XP, or Windows Vista.
No 0x00000004	Windows NT Server, Windows 2000 Server, or Windows Vista and Windows Server 2008.
No 0x00000005	Any premium, advanced, or datacenter edition of an NT-class Windows Server OS.

dwpQueue: A pointer to a variable to receive the queue context value, which MUST be the same as *pphContext*. MUST NOT be NULL.

phQueue: A pointer to a variable to receive the queue context value, which MUST be the same as *pphContext*. MUST NOT be NULL.

Return Values: On success, this method MUST return MQ_OK (0x00000000); otherwise, the server MUST return a failure **HRESULT**, and the client MUST treat all failure HRESULTs identically. Additionally, if a failure HRESULT is returned, the client MUST disregard all out-parameter values.

MQ_OK (0x00000000)
MQ_ERROR_INVALID_PARAMETER (0xc00e0006)
STATUS_SHARING_VIOLATION (0xc0000043)

Exceptions Thrown: In addition to the exceptions thrown by the underlying RPC protocol [MS-RPCE], the method MAY throw HRESULT failure codes as RPC exceptions. The client MUST treat all thrown HRESULT codes identically. Additionally, the client MUST disregard all out-parameter values when any failure HRESULT is thrown.

This method is invoked at the dynamically assigned endpoint returned by the **R_QMGetRTQMServerPort** method when IP_HANDSHAKE (0x00000000) or IPX_HANDSHAKE (0x00000002) is the interface specified by the *fIP* parameter.

When processing this call, the server MUST:

- Determine if input parameter values violate constraints specified above. If an invalid parameter is detected, the server MUST take no further action and return MQ_ERROR_INVALID_PARAMETER (0xc00e0006).
- Locate the queue identified by *pQueueFormat* in the queue table. If the queue is not present in the queue table, return a failure HRESULT.
- Using the current state of the queue, process the request according to the following queue state table:

Current queue state	Requested share mode	Result	Effect on queue state
Closed	MQ_DENY_SHARE	Success	Transition to open exclusive state.
Closed	MQ_DENY_NONE	Success	Transition to open shared state.
Open Exclusive	MQ_DENY_NONE or MQ_DENY_RECEIVE_SHARE	Failure	No change
Open Shared	MQ_DENY_RECEIVE_SHARE	Failure	No change
Open Shared	MQ_DENY_NONE	Success	Remain in open shared state and increment shared open count.

- If the result indicated by the table above is one of success, create a PCTX_OPENREMOTE_HANDLE_TYPE and add an entry for it to the QueueContextHandle table. If the result is one of failure, return a failure HRESULT.
- Add a reference to the queue identified by *pQueueFormat* to the QueueContextHandle entry.
- Return the PCTX_OPENREMOTE_ HANDLE_TYPE via the *pphContext*, *phQueue*, *dwpQueue*, and *pdwContext* parameters.

3.1.4.3 R_QMCloseRemoteQueueContext (Opnum 3)

The **R_QMCloseRemoteQueueContext** method closes a remote queue handle originally obtained from **R_QMOpenRemoteQueue**.

```
void R_QMCloseRemoteQueueContext(
  [in, out] PCTX_OPENREMOTE_HANDLE_TYPE* pphContext
);
```

pphContext: An RPC context handle as defined in [MS-RPCE] section 2. This handle MUST have been acquired from the *pphContext* parameter of the **R_QMOpenRemoteQueue** method. On input, if this parameter is NULL, the server MUST take no action. On output, the server MUST set this parameter to NULL.

Return Values: None.

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

This method is invoked at the dynamically assigned endpoint returned by the **R_QMGetRTQMServerPort** method when IP_HANDSHAKE (0x00000000) or IPX_HANDSHAKE (0x00000002) is the interface specified by the *fIP* parameter.

When processing this call, the server MUST:

- Determine if input parameter values violate constraints specified above. If an invalid parameter is detected, the server MUST take no further action and return MQ_ERROR_INVALID_PARAMETER (0xc00e0006).
- Look up **PCTX_OPENREMOTE_HANDLE_TYPE** in the QueueContextHandle table.

- If found:
  - Find all entries in the cursor table associated with the QueueContextHandle and close them, as specified by section 3.1.4.19.
  - Remove the QueueContextHandle entry from the QueueContextHandle table.
- Else: return a failure HRESULT.

3.1.4.4 R_QMCreateRemoteCursor (Opnum 4)

The **R_QMCreateRemoteCursor** method creates a cursor context at the server for use during remote read.

```
HRESULT R_QMCreateRemoteCursor(
  [in] handle_t hBind,
  [in] struct CACTransferBufferV1* ptb1,
  [in] DWORD hQueue,
  [out] DWORD* phCursor
);
```

hBind: MUST be set to an RPC binding handle, as specified in [MS-RPCE] section 2.

ptb1: This parameter MUST be ignored. Clients SHOULD pass NULL.<21>

hQueue: A queue handle value acquired from **R_QMOpenRemoteQueue**.

phCursor: A pointer to a DWORD that receives the created cursor handle. MUST NOT be NULL on input.

Return Values: On success, this method MUST return MQ_OK (0x00000000); otherwise, the server MUST return a failure HRESULT, and the client MUST treat all failure HRESULTs identically. Additionally, if a failure HRESULT is returned, the client MUST disregard all out-parameter values.

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

This method is invoked at the dynamically assigned endpoint returned by the **R_QMGetRTQMServerPort** method when IP_HANDSHAKE (0x00000000) or IPX_HANDSHAKE (0x00000002) is the interface specified by the *fIP* parameter.

When processing this call, the server MUST:

- Determine if input parameter values violate constraints specified above. If an invalid parameter is detected, the server MUST take no further action and return MQ_ERROR_INVALID_PARAMETER (0xc00e0006).
- Look up hQueue in the QueueContextHandle table.
- If not found, return a failure HRESULT.
- Create a new cursor handle. The handle value MUST be unique across all cursor handles associated with the QueueContextHandle.
- Set phCursor to the value of the new cursor handle.

- Create a new entry in the cursor table, keyed by phCursor and hQueue. The cursor is local to the queue manager.
- Set the cursor position to the head of the queue.
- Set the cursor state to Uninitialized, as specified in section 3.1.1.9.

3.1.4.5 R_QMCreateObjectInternal (Opnum 6)

A dependent client calls the **R_QMCreateObjectInternal** method to create a new private queue located on the supporting server.

```
HRESULT R_QMCreateObjectInternal(
  [in] handle_t hBind,
  [in] DWORD dwObjectType,
  [in, string] const WCHAR* lpwcsPathName,
  [in, range(0, 524288)] DWORD SDSize,
  [in, unique, size_is(SDSize)] unsigned char* pSecurityDescriptor,
  [in, range(1, 128)] DWORD cp,
  [in, size_is(cp)] DWORD aProp[],
  [in, size_is(cp)] PROPVARIANT apVar[]
);
```

hBind: MUST be set to an RPC binding handle as described in [MS-RPCE] section 2.

dwObjectType: MUST be 0x00000001 in order to specify a queue.

lpwcsPathName: MUST be a pointer to a NULL-terminated string containing a path name for the queue to be created. The path name MUST identify a private queue local to the supporting server by including "." as the computer name, or by using the supporting server computer name.

SDSize: MUST be set to the byte length of the SECURITY_DESCRIPTOR buffer pointed to by *pSecurityDescriptor*. If *pSecurityDescriptor* is NULL, this parameter MUST be 0x00000000.

pSecurityDescriptor: Must be a pointer to an array of bytes containing a SECURITY_DESCRIPTOR structure. The SECURITY_DESCRIPTOR specifies the initial security configuration for the queue to be created. This value MAY be NULL, in which case the server MUST provide a default security configuration for the new queue. The SECURITY_DESCRIPTOR structure is defined in [MS-DTYP] section **2.4.6**.

cp: MUST be set to the size (in elements) of the arrays *aProp* and *apVar*. The arrays *aProp* and *apVar* MUST have an identical number of elements, and MUST contain at least one element.

aProp: MUST be an array of queue property identifiers that, together with the *apVar* array, specify the initial queue property values for the new queue. Each element MUST specify a value from the queue property identifiers table defined in [MS-MQMQ] section 2.3.1. Each element MUST specify the property identifier for the corresponding property value at the same element index in *apVar* and MUST contain at least one element. Each element MUST contain a queue property identifier; identifiers for other properties are not permitted.

If the queue identified by *lpwcsPathName* already exists, the server MUST NOT alter the existing queue.

apVar: MUST be an array that specifies the property values to associate with the new queue. Each element MUST specify the property value for the corresponding property identifier at the same element index in *aProp* and MUST contain at least one element.

Return Values: On success, this method MUST return MQ_OK (0x00000000); otherwise, the server MUST return a failure **HRESULT**,<22> and the client MUST treat all failure HRESULTs identically.

MQ_OK (0x00000000)
MQ_ERROR_INVALID_PARAMETER (0xc00e0006)

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

This method is called only by dependent clients. Servers SHOULD enforce limitations and security measures appropriate for dependent clients.<23>

This method is invoked at the dynamically assigned endpoint returned by the **R_QMGetRTQMServerPort** method when IP_HANDSHAKE (0x00000000) or IPX_HANDSHAKE (0x00000002) is the interface specified by the *fIP* parameter.

When processing this call, the server MUST:

- Determine if input parameter values violate constraints specified above. If an invalid parameter is detected, the server MUST take no further action and return MQ_ERROR_INVALID_PARAMETER (0xc00e0006).
- Look up *lpwcsPathName* in the queue table to determine if the queue already exists.
- If found, take no further action and return a failure HRESULT.
- Else:
  - Create a new entry in the queue table, given the *lpwcsPathName* name, the properties defined by *aProp* and *apVar*, and the SECURITY_DESCRIPTOR *pSecurityDescriptor*.
  - Initialize the queue state to closed, as specified in section 3.1.1.8.
  - Initialize the shared open count for the queue to zero.
  - Return MQ_OK (0x00000000).

3.1.4.6 R_QMSetObjectSecurityInternal (Opnum 7)

A dependent client calls the **R_QMSetObjectSecurityInternal** method to update the security configuration of a private queue located on the supporting server.

```
HRESULT R_QMSetObjectSecurityInternal(
  [in] handle_t hBind,
  [in] struct OBJECT_FORMAT* pObjectFormat,
  [in] DWORD SecurityInformation,
  [in, range(0, 524288)] DWORD SDSize,
  [in, unique, size_is(SDSize)] unsigned char* pSecurityDescriptor
);
```

hBind: MUST be set to an RPC binding handle as described in [MS-RPCE] section 2.

pObjectFormat: MUST point to an **OBJECT_FORMAT** structure that identifies an existing local private queue on the supporting server for which the security configuration will be updated. This MUST NOT be NULL. The **ObjType** member of the structure MUST be 0x00000001. The **pQueueFormat** member MUST NOT be NULL.

SecurityInformation: MUST contain a value from the SECURITY_INFORMATION enumeration which indicates the portions of the provided SECURITY_DESCRIPTOR to be applied to the queue identified by *pObjectFormat*. The SECURITY_INFORMATION enumeration is defined in [MS-MQMQ] section 2.2.3.

SDSize: MUST be set to the byte length of the buffer pointed to by *pSecurityDescriptor*.

pSecurityDescriptor: MUST be a pointer to an array of bytes containing a SECURITY_DESCRIPTOR structure (see [MS-DTYP] section **2.4.6**).

Return Values: On success, this method MUST return MQ_OK (0x00000000); otherwise, the server MUST return a failure **HRESULT**,<24> and the client MUST treat all failure HRESULTs identically. Additionally, if a failure HRESULT is returned, the client MUST disregard all out-parameter values.

MQ_OK (0x00000000)
MQ_ERROR_INVALID_PARAMETER (0xc00e0006)

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

This method is called only by dependent clients. Servers SHOULD enforce limitations and security measures appropriate for dependent clients.<25>

This method is invoked at the dynamically assigned endpoint returned by the **R_QMGetRTQMServerPort** method when IP_HANDSHAKE (0x00000000) or IPX_HANDSHAKE (0x00000002) is the interface specified by the *fIP* parameter.

When processing this call, the server MUST:

- Determine if input parameter values violate constraints specified above. If an invalid parameter is detected, the server MUST take no further action and return MQ_ERROR_INVALID_PARAMETER (0xc00e0006).
- Look up an entry in the queue table identified by *pObjectFormat*.
- If found:
  - Update the SECURITY_DESCRIPTOR for the Queue.
  - Return MQ_OK (0x00000000).
- Else: return a failure HRESULT.

3.1.4.7 R_QMGetObjectSecurityInternal (Opnum 8)

A dependent client calls the **R_QMGetObjectSecurityInternal** method to retrieve the security configuration of a private queue located on the supporting server.

```
HRESULT R_QMGetObjectSecurityInternal(
  [in] handle_t hBind,
  [in] struct OBJECT_FORMAT* pObjectFormat,
  [in] DWORD RequestedInformation,
  [out, size_is(nLength)] unsigned char* pSecurityDescriptor,
  [in, range(0, 524288)] DWORD nLength,
  [out] DWORD* lpnLengthNeeded
);
```

hBind: MUST be set to an RPC binding handle as described in [MS-RPCE] section 2.

pObjectFormat: MUST point to an **OBJECT_FORMAT** structure which identifies an existing local private queue on the supporting server for which the security configuration is to be retrieved. It MUST NOT be NULL. The **ObjType** member of the structure MUST be 0x00000001, and the **pQueueFormat** member MUST NOT be NULL.

RequestedInformation: MUST contain a value from the SECURITY_INFORMATION enumeration which indicates the portions of the SECURITY_DESCRIPTOR to be retrieved from the queue identified by *pObjectFormat*. The SECURITY_INFORMATION enumeration is defined in [MS-MQMQ] section 2.2.3.

pSecurityDescriptor: MUST be a pointer to an array of bytes into which the server MUST write a self-relative SECURITY_DESCRIPTOR structure. The server MUST NOT write more than *nLength* bytes to the buffer. If the buffer provided by the client is too small (as indicated by the *nLength* parameter) to contain the SECURITY_DESCRIPTOR for the queue identified by *pObjectFormat*, the server MUST return MQ_ERROR_SECURITY_DESCRIPTOR_TOO_SMALL (0xc00e0023). This parameter MAY be NULL if *nLength* is 0x00000000.

The SECURITY_DESCRIPTOR structure is defined in [MS-DTYP] section **2.4.6**.

nLength: MUST indicate the byte length of the buffer pointed to by *pSecurityDescriptor*. This value MAY be 0x00000000.

lpnLengthNeeded: MUST NOT be NULL. The server MUST set this DWORD to the byte length of the SECURITY_DESCRIPTOR structure for the queue identified by *pObjectFormat*.

Return Values: On success, this method MUST return MQ_OK (0x00000000); otherwise, the server MUST return a failure **HRESULT**,<26> and the client MUST treat all failure HRESULTs identically. Additionally, if a failure HRESULT is returned, the client MUST disregard all out-parameter values, with the following exception:

If *nLength* is less than the byte length of the buffer required to contain the SECURITY_DESCRIPTOR for the queue identified by *pObjectFormat*, the server MUST return the byte length of the buffer required to contain the SECURITY_DESCRIPTOR in the *lpnLengthNeeded* parameter and MUST return MQ_ERROR_SECURITY_DESCRIPTOR_TOO_SMALL (0xc00e0023).

MQ_OK (0x00000000)
MQ_ERROR_INVALID_PARAMETER (0xc00e0006)
MQ_ERROR_SECURITY_DESCRIPTOR_TOO_SMALL (0xc00e0023)

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

This method is called only by dependent clients. Servers SHOULD enforce limitations and security measures appropriate for dependent clients.<27>

This method is invoked at the dynamically assigned endpoint returned by the **R_QMGetRTQMServerPort** method when IP_HANDSHAKE (0x00000000) or IPX_HANDSHAKE (0x00000002) is the interface specified by the *fIP* parameter.

When processing this call, the server MUST:

- Determine if input parameter values violate constraints specified above. If an invalid parameter is detected, the server MUST take no further action and return MQ_ERROR_INVALID_PARAMETER (0xc00e0006).
- Look up an entry in the queue table identified by *pObjectFormat*.
- If found:
  - Retrieve the SECURITY_DESCRIPTOR for the queue.
  - If the SECURITY_DESCRIPTOR for the queue is longer than *nLength*, set *lpnLengthNeeded* to the SECURITY_DESCRIPTOR length and return MQ_ERROR_SECURITY_DESCRIPTOR_TOO_SMALL (0xc00e0023).
  - Otherwise, copy the SECURITY_DESCRIPTOR into the provided buffer, set *lpnLengthNeeded* to the length of the SECURITY_DESCRIPTOR, and return MQ_OK (0x00000000).
- Else: return a failure HRESULT.

3.1.4.8 R_QMDeleteObject (Opnum 9)

A dependent client calls **R_QMDeleteObject** to delete a private queue located on the supporting server.

```
HRESULT R_QMDeleteObject(
  [in] handle_t hBind,
  [in] struct OBJECT_FORMAT* pObjectFormat
);
```

hBind: MUST be set to an RPC binding handle as described in [MS-RPCE] section 2.

pObjectFormat: MUST point to an **OBJECT_FORMAT** structure that identifies an existing local private queue on the supporting server. MUST NOT be NULL. The **ObjType** member of the structure MUST be 0x00000001. The **pQueueFormat** member MUST NOT be NULL.

Return Values: On success, this method MUST return MQ_OK (0x00000000); otherwise, the server MUST return a failure HRESULT,<28> and the client MUST treat all failure HRESULTs identically.

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

This method is called only by dependent clients. Servers SHOULD enforce limitations and security measures appropriate for dependent clients.<29>

This method is invoked at the dynamically assigned endpoint returned by the **R_QMGetRTQMServerPort** method when IP_HANDSHAKE (0x00000000) or IPX_HANDSHAKE (0x00000002) is the interface specified by the *fIP* parameter.

When processing this call, the server MUST:

- Determine if input parameter values violate constraints specified above. If an invalid parameter is detected, the server MUST take no further action and return MQ_ERROR_INVALID_PARAMETER (0xc00e0006).
- Look up an entry in the queue table identified by pObjectFormat.
- If found:
  - Update any associated QueueContextHandle table entries to indicate that the associated queue is deleted.
  - Remove the queue from the queue table.
  - Return MQ_OK (0x00000000).
- Else: return a failure HRESULT.

3.1.4.9 R_QMGetObjectProperties (Opnum 10)

A dependent client calls **R_QMGetObjectProperties** to retrieve properties from a private queue located on a supporting server.

```
HRESULT R_QMGetObjectProperties(
  [in] handle_t hBind,
  [in] struct OBJECT_FORMAT* pObjectFormat,
  [in, range(1, 128)] DWORD cp,
  [in, size_is(cp )] DWORD aProp[],
  [in, out, size_is(cp )] PROPVARIANT apVar[]
);
```

hBind: MUST be set to an RPC binding handle as described in [MS-RPCE] section 2.

pObjectFormat: MUST point to an **OBJECT_FORMAT** structure which identifies an existing local private queue on the supporting server. MUST NOT be NULL. The **ObjType** member of the structure MUST be 0x00000001. The **pQueueFormat** member MUST NOT be NULL.

cp: MUST be set to the size (in elements) of the arrays *aProp* and *apVar*. The arrays *aProp* and *apVar* MUST have an identical number of elements, and MUST contain at least one element.

aProp: MUST be an array of queue property identifiers of properties to retrieve. Each element MUST specify a value from the queue property identifiers table defined in [MS-MQMQ] section

2.3.1. Each element MUST specify the queue property identifier for the corresponding queue property value at the same element index in *apVar*. MUST contain at least one element.

apVar: MUST contain at least one element. On input, each element MUST be initialized to the appropriate VARTYPE for the associated property specified by the same element in aProp, or VT_NULL. On success, the server MUST populate the elements of this array with property values for the properties identified by the corresponding elements of *aProp*.

Return Values: On success, this method MUST return MQ_OK (0x00000000); otherwise, the server MUST return a failure **HRESULT**,<30> and the client MUST treat all failure HRESULTs identically. Additionally, if a failure HRESULT is returned, the client MUST disregard all out-parameter values.

MQ_OK (0x00000000)
MQ_ERROR_INVALID_PARAMETER (0xc00e0006)

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

This method is called only by dependent clients. Servers SHOULD enforce limitations and security measures appropriate for dependent clients.<31>

This method is invoked at the dynamically assigned endpoint returned by the **R_QMGetRTQMServerPort** method when IP_HANDSHAKE (0x00000000) or IPX_HANDSHAKE (0x00000002) is the interface specified by the *fIP* parameter.

When processing this call, the server MUST:

- Determine if input parameter values violate constraints specified above. If an invalid parameter is detected, the server MUST take no further action and return MQ_ERROR_INVALID_PARAMETER (0xc00e0006).
- Look up an entry in the queue table identified by pObjectFormat.
- If Found:
  - Retrieve the properties for the queue.
- Else: return a failure HRESULT.

3.1.4.10 R_QMSetObjectProperties (Opnum 11)

The **R_QMSetObjectProperties** method is called by a dependent client to update properties of a local private queue.

```
HRESULT R_QMSetObjectProperties(
  [in] handle_t hBind,
  [in] struct OBJECT_FORMAT* pObjectFormat,
  [in, range(1, 128)] DWORD cp,
  [in, unique, size_is(cp)] DWORD aProp[],
  [in, unique, size_is(cp)] PROPVARIANT apVar[]
);
```

hBind: MUST be set to an RPC binding handle as described in [MS-RPCE] section 2.

pObjectFormat: MUST point to an **OBJECT_FORMAT** structure which identifies an existing local private queue on the supporting server. MUST NOT be NULL. The **ObjType** member of the structure MUST be 0x00000001. The **pQueueFormat** member MUST NOT be NULL.

cp: MUST be set to the size (in elements) of the arrays aProp and apVar. The arrays aProp and apVar MUST have an identical number of elements, and MUST contain at least one element.

aProp: MUST be an array of queue property identifiers for properties to be updated. Each element MUST specify a value from the queue property identifiers table defined in [MS-MQMQ] section 2.3.1. Each element MUST specify the queue property identifier for the corresponding queue property value at the same element index in apVar. MUST contain at least one element. Read-only and otherwise inappropriate properties MUST be ignored, and MUST NOT result in a failure.

apVar: MUST be an array that specifies the property values to update. Each element MUST specify the property value for the corresponding property identifier at the same element index in aProp. MUST contain at least one element. The vt (VARTYPE) member of each **PROPVARIANT** element MUST be set to the appropriate type for the property being updated; otherwise, the server MUST return MQ_ERROR_ILLEGAL_PROPERTY_VT (0xc00e0019). Queue properties and their appropriate VARTYPEs are specified by [MS-MQMQ] section 2.3.1.

Return Values: On success, this method MUST return MQ_OK (0x00000000); otherwise, the server MUST return a failure HRESULT,<32> and the client MUST treat all failure HRESULTs identically.

MQ_OK (0x00000000)
MQ_ERROR_INVALID_PARAMETER (0xc00e0006)
MQ_ERROR_ILLEGAL_PROPERTY_VT (0xc00e0019)

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

This method is called only by dependent clients. Servers SHOULD enforce limitations and security measures appropriate for dependent clients.<33>

This method is invoked at the dynamically assigned endpoint returned by the **R_QMGetRTQMServerPort** method when IP_HANDSHAKE (0x00000000) or IPX_HANDSHAKE (0x00000002) is the interface specified by the *fIP* parameter.

When processing this call, the server MUST:

- Determine if input parameter values violate constraints specified above. If an invalid parameter is detected, the server MUST take no further action and return MQ_ERROR_INVALID_PARAMETER (0xc00e0006).
- Look up an entry in the queue table identified by pObjectFormat.
- If Found:
  - Update the properties for the queue.
  - Return MQ_OK (0x00000000).
- Else: return a failure HRESULT.

3.1.4.11 R_QMObjectPathToObjectFormat (Opnum 12)

A dependent client calls **R_QMObjectPathToObjectFormat** to determine a format name for a queue identified by a given path name.

```
HRESULT R_QMObjectPathToObjectFormat(
  [in] handle_t hBind,
  [in, string] const WCHAR* lpwcsPathName,
  [in, out] struct OBJECT_FORMAT* pObjectFormat
);
```

hBind: MUST be set to an RPC binding handle as described in [MS-RPCE] section 2.

lpwcsPathName: MUST be a pointer to a NULL-terminated path name string, as defined by [MS-MQMQ] section 2.1.1. The path name MUST identify an existing private queue located on a supporting server.

pObjectFormat: MUST be a pointer to an **OBJECT_FORMAT** structure, as described in [MS-MQMQ]. On success, this structure MUST be populated with a direct format name or private format name for the queue identified by *lpwcsPathName*. This specification does not mandate the process through which a server produces a format name for a given path name.

On input, pObjectFormat MUST NOT be NULL, the **ObjType** member MUST be 0x00000001, and the **m_qft** member MUST be QUEUE_FORMAT_TYPE_UNKNOWN (0x00000000).

Return Values: On success, this method MUST return MQ_OK (0x00000000); otherwise, the server MUST return a failure **HRESULT**,<34> and the client MUST treat all failure HRESULTs identically. Additionally, if a failure HRESULT is returned, the client MUST disregard all out-parameter values.

MQ_OK (0x00000000)
MQ_ERROR_INVALID_PARAMETER (0xc00e0006)

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

This method is called only by dependent clients. Servers SHOULD enforce limitations and security measures appropriate for dependent clients.<35>

This method is invoked at the dynamically assigned endpoint returned by the **R_QMGetRTQMServerPort** method when IP_HANDSHAKE (0x00000000) or IPX_HANDSHAKE (0x00000002) is the interface specified by the *fIP* parameter.

When processing this call, the server MUST:

- Determine if input parameter values violate constraints specified above. If an invalid parameter is detected, the server MUST take no further action and return MQ_ERROR_INVALID_PARAMETER (0xc00e0006).

3.1.4.12 R_QMGetTmWhereabouts (Opnum 14)

A dependent client calls **R_QMGetTmWhereabouts** to obtain transaction manager whereabouts, as specified in [MS-DTCO], from the supporting server. The whereabouts enable callers to generate

exported transaction cookies, which are required to enlist the supporting server's resource manager in an external transaction.

```
HRESULT R_QMGetTmWhereabouts(
  [in] handle_t hBind,
  [in, range(0,131072)] DWORD cbBufSize,
  [out, size_is(cbBufSize)] unsigned char* pbWhereabouts,
  [out] DWORD* pcbWhereabouts
);
```

hBind: MUST be set to an RPC binding handle as described in [MS-RPCE] section 2.

cbBufSize: MUST be set to the byte length of the buffer pointed to by *pbWhereabouts*. If this value is 0x00000000, the server MUST ignore the *pbWhereabouts* parameter.

pbWhereabouts: On success, points to an array of bytes containing a Distributed Transaction Coordinator (DTC) SWhereabouts structure, as specified in [MS-DTCO].

pcbWhereabouts: On success, or if MQ_ERROR_USER_BUFFER_TOO_SMALL (0xc00e0028) is returned, pcbWhereabouts points to a DWORD containing the byte length of the SWhereabouts structure retrieved from the DTC; otherwise, this parameter MUST be ignored.

Return Values: If successful, this method MUST return MQ_OK (0x00000000), and place the entire SWhereabouts structure retrieved from the DTC into the pbWhereabouts buffer provided by the caller. The server MUST also return the byte length of the SWhereabouts structure in the *pcbWhereabouts* parameter.

If the server successfully retrieves the SWhereabouts structure from DTC, but the size of the buffer provided by the caller (as indicated by the *cbBufSize* parameter) is too small to contain the entire SWhereabouts structure, the server MUST take the following actions:

- The server MUST ignore the *pbWhereabouts* pointer.
- The server MUST set the *pcbWhereabouts* parameter to the size of the SWhereabouts structure retrieved from DTC.
- The server MUST return MQ_ERROR_USER_BUFFER_TOO_SMALL (0xc00e0028). This return value indicates to the caller that this function MAY succeed if a larger buffer is provided.

If input parameter values violate constraints specified above, the server MUST take no further action and return MQ_ERROR_INVALID_PARAMETER (0xc00e0006).

If any other error occurs, the server MUST return a failure HRESULT, and the client MUST treat all failure HRESULTs identically. Additionally, if any other failure HRESULT is returned, the client MUST disregard all out-parameter values.

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

This method is called only by dependent clients. Servers SHOULD enforce limitations and security measures appropriate for dependent clients.<36>

This method is invoked at the dynamically assigned endpoint returned by the **R_QMGetRTQMServerPort** method when IP_HANDSHAKE (0x00000000) or IPX_HANDSHAKE (0x00000002) is the interface specified by the *fIP* parameter.

3.1.4.13 R_QMEnlistTransaction (Opnum 15)

A dependent client calls the **R_QMEnlistTransaction** method to enlist the supporting server's resource manager in an external transaction.

```
HRESULT R_QMEnlistTransaction(
  [in] handle_t hBind,
  [in] struct XACTUOW* pUow,
  [in, range(0, 131072)] DWORD cbCookie,
  [in, size_is(cbCookie)] unsigned CHAR* pbCookie
);
```

hBind: MUST be set to an RPC binding handle as described in [MS-RPCE] section 2.

pUow: MUST point to an **XACTUOW** structure that identifies the external transaction to which the server is to enlist, as specified in section 2.2.2.1.

cbCookie: MUST be set to the byte length of the buffer pointed to by *pbCookie*.

pbCookie: MUST be a pointer to an array of bytes containing an exported transaction cookie, as specified in [MS-DTCO].

Return Values: On success, this method MUST return MQ_OK (0x00000000); otherwise, the server MUST return a failure HRESULT, and the client MUST treat all failure HRESULTs identically.

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

This method is called only by dependent clients. Servers SHOULD enforce limitations and security measures appropriate for dependent clients.<37>

This method is invoked at the dynamically assigned endpoint returned by the **R_QMGetRTQMServerPort** method when IP_HANDSHAKE (0x00000000) or IPX_HANDSHAKE (0x00000002) is the interface specified by the *fIP* parameter.

When processing this call, the server MUST:

- Determine if input parameter values violate constraints specified above. If an invalid parameter is detected, the server MUST take no further action and return MQ_ERROR_INVALID_PARAMETER (0xc00e0006).
- Look up the transaction in the transaction table, using the pUow transaction identifier.
- If found:
  - The transaction has already been enlisted. Take no action.
- Else:
  - Import and enlist the transaction using the provided cookie, as specified in [MS-DTCO].
  - Add a new entry to the transaction table, keyed by the pUow transaction identifier.
- Return MQ_OK (0x00000000).

3.1.4.14 R_QMEnlistInternalTransaction (Opnum 16)

A dependent client calls the **R_QMEnlistInternalTransaction** method to enlist the supporting server's resource manager in an internal transaction. The server returns a transaction handle associated with the given unit of work identifier (**XACTUOW**). The returned transaction handle is used when calling **R_QMCommitTransaction** or **R_QMAbortTransaction**. The **XACTUOW** structure is provided for calls to **rpc_ACSendMessageEx** and/or **rpc_ACReceiveMessageEx** of the qmcomm2 RPC interface.

```
HRESULT R_QMEnlistInternalTransaction(
  [in] handle_t hBind,
  [in] XACTUOW* pUow,
  [out] RPC_INT_XACT_HANDLE* phIntXact
);
```

hBind: MUST be set to an RPC binding handle as described in [MS-RPCE] section 2.

pUow: MUST point to an **XACTUOW** structure that uniquely identifies the internal transaction in which the server is to enlist.<38>

phIntXact: An **RPC_INT_XACT_HANDLE** identifying a new internal transaction context for the transaction identifier pUow.

Return Values: On success, this method MUST return MQ_OK (0x00000000); otherwise, the server MUST return a failure HRESULT, and the client MUST treat all failure HRESULTs identically. Additionally, if a failure HRESULT is returned, the client MUST disregard all out-parameter values.

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

This method is called only by dependent clients. Servers SHOULD enforce limitations and security measures appropriate for dependent clients.<39>

This method is invoked at the dynamically assigned endpoint returned by the **R_QMGetRTQMServerPort** method when IP_HANDSHAKE (0x00000000) or IPX_HANDSHAKE (0x00000002) is the interface specified by the *fIP* parameter.

When processing this call, the server MUST:

- Determine if input parameter values violate constraints specified above. If an invalid parameter is detected, the server MUST take no further action and return MQ_ERROR_INVALID_PARAMETER (0xc00e0006).
- Look up the transaction in the transaction table, using the pUow transaction identifier.
- If found:
  - The transaction has already been enlisted. Take no action.
- Else:
  - Add a new entry to the transaction table, keyed by the pUow transaction identifier.
  - Create a new **RPC_INT_XACT_HANDLE** for the transaction and return it via phIntXact.

- Return MQ_OK (0x00000000).

3.1.4.15 R_QMCommitTransaction (Opnum 17)

A dependent client calls the **R_QMCommitTransaction** method to commit an internal transaction.

```
HRESULT R_QMCommitTransaction(
  [in, out] RPC_INT_XACT_HANDLE* phIntXact
);
```

phIntXact: MUST be an **RPC_INT_XACT_HANDLE** identifying the internal transaction to commit. MUST NOT be NULL. The value of this handle MUST have been acquired from **R_QMEnlistInternalTransaction**. On return, the server MUST set this parameter to NULL.

Return Values: On success, this method MUST return MQ_OK (0x00000000) and set phIntXact to NULL; otherwise, the server MUST return a failure HRESULT, and the client MUST treat all failure HRESULTs identically.

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

This method is called only by dependent clients. Servers SHOULD enforce limitations and security measures appropriate for dependent clients.<40>

This method is invoked at the dynamically assigned endpoint returned by the **R_QMGetRTQMServerPort** method when IP_HANDSHAKE (0x00000000) or IPX_HANDSHAKE (0x00000002) is the interface specified by the *fIP* parameter.

When processing this call, the server MUST:

- Determine if input parameter values violate constraints specified above. If an invalid parameter is detected, the server MUST take no further action and return MQ_ERROR_INVALID_PARAMETER (0xc00e0006).
- Look up an entry in the transaction table with the associated **RPC_INT_XACT_HANDLE** value.
- If not found, return a failure HRESULT.
- Carry out the operations in the transaction operation list associated with the transaction.
- Remove the transaction entry from the transaction table.

3.1.4.16 R_QMAbortTransaction (Opnum 18)

A dependent client calls the **R_QMAbortTransaction** method to abort an internal transaction.

```
HRESULT R_QMAbortTransaction(
  [in, out] RPC_INT_XACT_HANDLE* phIntXact
);
```

phIntXact: MUST be an **RPC_INT_XACT_HANDLE** identifying the internal transaction to abort. MUST NOT be NULL. The value of this handle MUST have been acquired from **R_QMEnlistInternalTransaction**. On return, the server MUST set this parameter to NULL.

Return Values: On success, this method MUST return MQ_OK (0x00000000) and MUST set phIntXact to NULL; otherwise, the server MUST return a failure HRESULT, and the client MUST treat all failure HRESULTs identically.

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

This method is called only by dependent clients. Servers SHOULD enforce limitations and security measures appropriate for dependent clients.<41>

This method is invoked at the dynamically assigned endpoint returned by the **R_QMGetRTQMServerPort** method when IP_HANDSHAKE (0x00000000) or IPX_HANDSHAKE (0x00000002) is the interface specified by the *fIP* parameter.

When processing this call, the server MUST:

- Determine if input parameter values violate constraints specified above. If an invalid parameter is detected, the server MUST take no further action and return MQ_ERROR_INVALID_PARAMETER (0xc00e0006).
- Look up an entry in the transaction table with the associated **RPC_INT_XACT_HANDLE** value.
- If not found, return a failure HRESULT.
- Rollback the operations in the transaction operation list associated with the transaction.
- Remove the transaction entry from the transaction table.

3.1.4.17 rpc_QMOpenQueueInternal (Opnum 19)

A dependent client calls **rpc_QMOpenQueueInternal** to obtain a local queue context handle, to determine if a queue is located at a remote queue manager, or to obtain a local context handle for an opened remote queue. If the call to **RemoteQMOpenQueue** ([MS-MQQP] section 3.1.4.3) fails, the result must be returned to the client, and the remote open queue sequence is discontinued. In the case of failure, any state changes need to be rolled back.

```
HRESULT rpc_QMOpenQueueInternal(
  [in] handle_t hBind,
  [in] struct QUEUE_FORMAT* pQueueFormat,
  [in] DWORD dwDesiredAccess,
  [in] DWORD dwShareMode,
  [in] DWORD hRemoteQueue,
  [in, out, ptr, string] WCHAR** lplpRemoteQueueName,
  [in] DWORD* dwpQueue,
  [in] GUID* pLicGuid,
  [in, string] WCHAR* lpClientName,
  [out] DWORD* pdwQMContext,
  [out] RPC_QUEUE_HANDLE* phQueue,
  [in] DWORD dwRemoteProtocol,
  [in] DWORD dwpRemoteContext
);
```

hBind: MUST be set to an RPC binding handle as described in [MS-RPCE] section 2.

pQueueFormat: MUST be a pointer to a **QUEUE_FORMAT** structure as specified in [MS-MQMQ] section 2.2.7 which identifies an existing queue to be opened. MUST NOT be NULL and MUST conform to the format-name syntax rules defined in [MS-MQMQ].

dwDesiredAccess: A **DWORD** that specifies the access mode requested for the queue. The access mode defines the set of operations which can be invoked using the returned queue handle. The value MUST be one of the following:

Value	Meaning
MQ_RECEIVE_ACCESS 0x00000001	The server MUST permit only the following operations using the returned queue handle: ▪ Message peek ▪ Message receive (peek and delete) ▪ Queue purge
MQ_SEND_ACCESS 0x00000002	The server MUST permit only message send operations using the returned queue handle.
MQ_PEEK_ACCESS 0x00000020	The server MUST permit only message peek operations using the returned queue handle.
MQ_RECEIVE_ACCESS|MQ_ADMIN_ACCESS 0x00000081	The returned queue handle MUST perform operations on the outgoing queue associated with the queue identified by *pQueueFormat*. Additionally, the server MUST permit only the following operations using the returned queue handle: ▪ Message peek ▪ Message receive (peek and delete) ▪ Queue purge
MQ_PEEK_ACCESS|MQ_ADMIN_ACCESS 0x000000a0	The returned queue handle MUST perform operations on the outgoing queue associated with the queue identified by *pQueueFormat*. Additionally, the server MUST permit only message peek operations using the returned queue handle.

If *pQueueFormat* contains an HTTP or multicast format name, *dwDesiredAccess* MUST be MQ_SEND_ACCESS (0x00000002).

If *pQueueFormat* identifies a sub-queue, *dwDesiredAccess* MUST NOT be MQ_SEND_ACCESS (0x00000002).

If *pQueueFormat* identifies a system, journal, machine, or connector queue, *dwDesiredAccess* MUST be MQ_RECEIVE_ACCESS (0x00000001) or MQ_PEEK_ACCESS (0x00000020).

If *pQueueFormat* identifies a remote queue, *dwDesiredAccess* MUST be MQ_RECEIVE_ACCESS (0x00000001) or MQ_PEEK_ACCESS (0x00000020).

dwShareMode: Specifies the exclusivity level for the opened queue. The value MUST be one of the following:

Value	Meaning
MQ_DENY_NONE 0x00000000	The queue is not opened exclusively.
MQ_DENY_RECEIVE_SHARE 0x00000001	The queue is opened for exclusive read access. If the queue has already been opened for read access, the server MUST return a failure HRESULT. If the queue is opened successfully for exclusive read access, subsequent attempts to open the same queue for read access MUST return a failure HRESULT until the queue has been closed.

If *dwDesiredAccess* is MQ_SEND_ACCESS (0x00000002), dwShareMode MUST be MQ_DENY_NONE (0x00000000).

hRemoteQueue: MUST be 0x00000000, or a **DWORD** containing a remote queue handle value acquired from the **R_QMOpenRemoteQueue** method.

lplpRemoteQueueName: On input, the server MUST ignore *lplpRemoteQueueName*. If *hRemoteQueue* is 0x00000000, and the queue identified by *pQueueFormat* is located at a remote queue manager, the server MUST set this string to a NULL-terminated path name, from which the client can determine the computer name of the remote queue manager, as specified in [MS-MQMQ] section 2.3.1.18.

If *pQueueFormat* identifies a queue local to the supporting server, the server MUST set *lplpRemoteQueueName* to NULL.

dwpQueue: MUST be NULL, or MUST be a pointer to a **DWORD** containing a remote queue context value acquired from the **R_QMOpenRemoteQueue** method when the value passed for *hRemoteQueue* was acquired. If a nonzero value for *hRemoteQueue* is provided, this parameter MUST NOT be NULL.

pLicGuid: MUST be a pointer to a valid GUID which uniquely identifies the client.<42><43> The server MAY ignore this parameter.<44>

lpClientName: MUST be a NULL-terminated string containing the client's computer name.<45> Servers MAY use this parameter in concert with the *pLicGuid* parameter to implement limits on the number of unique clients which may open queue handles.<46> Implementing connection limits is optional and not recommended.

pdwQMContext: A pointer to a variable that uniquely identifies an opened queue session like the parameter *phQueue* below. The server, if returning a queue handle via *phQueue*, MUST set this **DWORD** to the Queue Context Value associated with the handle returned (as specified in section 3.1.1.3); otherwise, this parameter MUST be set to 0x00000000. When the client calls **rpc_ACReceiveMessageEx**, it specifies a queue by providing the value that is returned by this parameter. On return, the client MUST ignore *pdwQMContext* if the value returned via *lplpRemoteQueueName* is non-NULL.

phQueue: An **RPC_QUEUE_HANDLE** context handle. On return, the client MUST ignore phQueue if the value returned via *lplpRemoteQueueName* is non-NULL.

dwRemoteProtocol: Clients MUST set this parameter to 0x00000000. Servers SHOULD ignore this parameter.<47>

Value	Meaning
0x00000000	The TCP/IP protocol sequence is to be used.
0x00000003	The IPX/SPX protocol sequence is to be used.

dwpRemoteContext: This parameter MUST contain the same **DWORD** value pointed to by dwpQueue.

Return Values: On success, this method MUST return MQ_OK (0x00000000); otherwise, if an error occurs, the server MUST return a failure **HRESULT**,<48> and the client MUST treat all failure HRESULTs identically. Additionally, if a failure HRESULT is returned, the client MUST disregard all out-parameter values.

MQ_OK (0x00000000)
MQ_ERROR_INVALID_PARAMETER (0xc00e0006)

Exceptions Thrown: In addition to the exceptions thrown by the underlying RPC protocol, as specified in [MS-RPCE], the method MAY throw HRESULT failure codes as RPC exceptions. The client MUST treat all thrown HRESULT codes identically. Additionally, the client MUST disregard all out-parameter values when any failure HRESULT is thrown.

This method is called only by dependent clients. Servers SHOULD enforce limitations and security measures appropriate for dependent clients.<49>

This method is invoked at the dynamically assigned endpoint returned by the **R_QMGetRTQMServerPort** method when IP_HANDSHAKE (0x00000000) or IPX_HANDSHAKE (0x00000002) is the interface specified by the *fIP* parameter.

When processing this call, the server MUST do the following:

- Determine if input parameter values violate constraints specified above. If an invalid parameter is detected, the server MUST take no further action and return MQ_ERROR_INVALID_PARAMETER (0xc00e0006).
- Attempt to locate the queue identified by pQueueFormat in the queue table. Outgoing queues should be excluded from the search.
- If found:
  - Using the current state of the queue, process the request according to the following queue state table:

Current queue state	Requested share mode	Result	Effect on queue state
Closed	MQ_DENY_SHARE	Success	Transition to open exclusive state
Closed	MQ_DENY_NONE	Success	Transition to open shared state
Open Exclusive	MQ_DENY_NONE or MQ_DENY_RECEIVE_SHARE	Failure	No change
Open Shared	MQ_DENY_RECEIVE_SHARE	Failure	No change

Current queue state	Requested share mode	Result	Effect on queue state
Open Shared	MQ_DENY_NONE	Success	Remain in open shared state and increment shared open count

- If the result indicated by the table above is one of success, create an **RPC_QUEUE_HANDLE** and add an entry for it to the QueueContextHandle table. Set phQueue to the new RPC_QUEUE_HANDLE. Set lplpRemoteQueueName to NULL, indicating to the client that an exchange with a remote queue manager is not required.
- Else, return a failure HRESULT.

- Else: the queue identified by pQueueFormat is not found in the queue table.
  - If hRemoteQueue is nonzero:
    - Create a new **RPC_QUEUE_HANDLE** and add an entry for it to the QueueContextHandle table.
    - Bind to the qm2qm interface at the remote queue manager and pass hRemoteQueue to the **RemoteQMOpenQueue** ([MS-MQQP] section 3.1.4.3) method to acquire a remote queue session handle, as specified in [MS-MQQP] section 3.1.4.3. If the call to RemoteQMOpenQueue is unsuccessful for any reason (including transport failures and time-outs <50>), the result must be returned to the client, and the remote open queue sequence is discontinued. In the case of failure, any state changes need to be rolled back.
    - Set the **RemoteQueueBindingHandle** property of the new QueueContextHandle table entry to the RPC binding handle established above.
    - Set the **RemoteQueueHandle** property of the new QueueContextHandle table entry to the **PCTX_RRSESSION_HANDLE_TYPE** ([MS-MQQP] section 2.2.1.1) value returned by **RemoteQMOpenQueue** ([MS-MQQP] section 3.1.4.3).
    - Set phQueue to the new **RPC_QUEUE_HANDLE**. Set lplpRemoteQueueName to NULL.
  - Else: hRemoteQueue is 0x00000000.
    - If pQueueFormat identifies a queue located at a remote queue manager:
      - If dwDesiredAccess is MQ_SEND_ACCESS:
        - Attempt to locate an outgoing queue in the queue table which stores messages to be transferred to the remote destination queue.
        - If found:
          - Follow the steps above which specify the procedure for transitioning queue state and returning a QueueContextHandle.
        - Else:
          - Create a new entry in the queue table for an outgoing queue to contain messages to be stored while being transferred to the remote destination queue.
          - The queue state must be initialized to closed, as specified in section 3.1.1.1.

- The shared open count for the queue must be initialized to zero.
- Follow the steps above which specify the procedure for transitioning queue state and returning a QueueContextHandle.

- Else:
  - Set lplpRemoteQueueName to the path name for the remote queue. Set *phQueue* and *pdwQMContext* to NULL.

- Else: pQueueFormat does not identify any known queue.
  - Return a failure HRESULT.

3.1.4.18 rpc_ACCloseHandle (Opnum 20)

A dependent client calls the **rpc_ACCloseHandle** method to close context handles acquired from **rpc_QMOpenQueueInternal**.

```
HRESULT rpc_ACCloseHandle(
  [in, out] RPC_QUEUE_HANDLE* phQueue
);
```

phQueue: MUST be a context handle acquired from the *phQueue* parameter of the **rpc_QMOpenQueueInternal** method. On success, the server MUST set this parameter to NULL.

Return Values: On success, this method MUST return MQ_OK (0x00000000); otherwise, the server MUST return a failure **HRESULT**,<51> and the client MUST treat all failure HRESULTs identically. Additionally, if a failure HRESULT is returned, the client MUST disregard all out-parameter values.

MQ_OK (0x00000000)
MQ_ERROR_INVALID_PARAMETER (0xc00e0006)

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

This method is called only by dependent clients. Servers SHOULD enforce limitations and security measures appropriate for dependent clients.<52>

This method is invoked at the dynamically assigned endpoint returned by the **R_QMGetRTQMServerPort** method when IP_HANDSHAKE (0x00000000) or IPX_HANDSHAKE (0x00000002) is the interface specified by the *fIP* parameter.

When processing this call, the server MUST:

- Determine if input parameter values violate constraints specified above. If an invalid parameter is detected, the server MUST take no further action and return MQ_ERROR_INVALID_PARAMETER (0xc00e0006).
- Look up the **RPC_QUEUE_HANDLE** in the QueueContextHandle table.
- If not found, return a failure HRESULT.

- Close all entries in the cursor table associated with the QueueContextHandle as specified by section 3.1.4.19.
- Remove the QueueContextHandle entry from the QueueContextHandle table.
- If the QueueContextHandle table entry contains a populated **RemoteQueueHandle** property, proactively close the **RemoteQueueHandle** by invoking the RemoteQMCloseQueue method of the qm2qm RPC interface as specified in [MS-MQQP] section 3.1.4.4, using the binding handle contained in the associated RemoteQueueBindingHandle property. Upon completion of RemoteQMCloseQueue, successful or not, dispose of the RemoteQueueBindingHandle as appropriate.
  - Note: This method SHOULD<53> start a parallel process to perform the above operations and return S_OK without waiting for the process to complete. Since S_OK is returned to the client without regard for the activities in the parallel process, failures which occur in the parallel process will not be conveyed to the client, which will delay reclamation of resources in the [MS-MQQP] server.

3.1.4.19 rpc_ACCloseCursor (Opnum 22)

A dependent client calls the **rpc_ACCloseCursor** method to close a cursor handle acquired from the **rpc_ACCreateCursorEx** method of the qmcomm2 RPC interface.

```
HRESULT rpc_ACCloseCursor(
  [in] RPC_QUEUE_HANDLE hQueue,
  [in] DWORD hCursor
);
```

hQueue: MUST be the queue context handle passed to **rpc_ACCreateCursorEx** when the cursor was created.

hCursor: MUST contain a cursor handle value obtained from **rpc_ACCreateCursorEx**, or the reserved value 0x0000000b.

Return Values: On success, this method MUST return MQ_OK (0x00000000); otherwise, the server MUST return a failure **HRESULT**,<54> and the client MUST treat all failure HRESULTs identically. Additionally, if a failure HRESULT is returned, the client MUST disregard all out-parameter values.

MQ_OK (0x00000000)
MQ_ERROR_INVALID_PARAMETER (0xc00e0006)

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

If hCursor is a handle for a remote cursor, the server SHOULD inform the remote queue manager to close the remote cursor by invoking the QMRemoteCloseCursor method of the qm2qm RPC interface (as specified in [MS-MQQP]) on the remote server which owns the remote cursor.

This method is called only by dependent clients. Servers SHOULD enforce limitations and security measures appropriate for dependent clients.<55>

This method is invoked at the dynamically assigned endpoint returned by the **R_QMGetRTQMServerPort** method when IP_HANDSHAKE (0x00000000) or IPX_HANDSHAKE (0x00000002) is the interface specified by the *fIP* parameter.

When processing this call, the server MUST:

- Determine if input parameter values violate constraints specified above. If an invalid parameter is detected, the server MUST take no further action and return MQ_ERROR_INVALID_PARAMETER (0xc00e0006).
- If hCursor is 0x0000000b, take no further action and return MQ_OK (0x00000000).
- Else:
  - Look up the cursor entry in the cursor table, using the hQueue and hCursor key values.
  - If not found, return a failure HRESULT.
  - If there is a remote cursor handle associated with the cursor, close it by invoking the R_RemoteQMCloseCursor method of the qm2qm RPC interface as defined in [MS-MQQP] section **3.1.4.5**. To expedite returning control to the client, this operation MAY be performed asynchronously.
  - Remove the cursor entry from the cursor table.

3.1.4.20 rpc_ACSetCursorProperties (Opnum 23)

A dependent client calls the **rpc_ACSetCursorProperties** method to associate a remote cursor handle acquired from **R_QMCreateRemoteCursor** with a local cursor handle acquired from **rpc_ACCreateCursorEx**.

Note This method is obsolete. The server SHOULD take no action and return MQ_ERROR_ILLEGAL_OPERATION (0xc00e0064).<56> Behavior for Windows NT and Windows 2000 is specified here.

```
HRESULT rpc_ACSetCursorProperties(
  [in] RPC_QUEUE_HANDLE hProxy,
  [in] DWORD hCursor,
  [in] DWORD hRemoteCursor
);
```

hProxy: MUST be the queue context handle passed to **rpc_ACCreateCursorEx** when the cursor specified by hCursor was created.

hCursor: MUST be a local cursor handle value acquired from the **rpc_ACCreateCursorEx** method of the qmcomm2 interface.

hRemoteCursor: MUST be a handle value for a remote cursor acquired from **R_QMCreateRemoteCursor**.

Return Values: On success, this method MUST return MQ_OK (0x00000000); otherwise, the server MUST return a failure HRESULT, and the client MUST treat all failure HRESULTs identically.

This method is obsolete. Servers SHOULD take no action and return MQ_ERROR_ILLEGAL_OPERATION (0xc00e0064). Servers SHOULD contact the remote queue

manager on behalf of the client when **rpc_ACCreateCursorEx** is called to create a remote cursor.

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

During the dependent client cursor creation sequence, the supporting server MAY indicate that the client MUST contact a remote queue manager to proceed.<57> In response, the client MUST call **R_QMGetRemoteQueueName** to determine the remote queue manager name, and MUST then invoke **R_QMCreateRemoteCursor** at the remote queue manager. Next, the client MUST call this method to associate the handle obtained from **R_QMCreateRemoteCursor** with the original cursor handle obtained from **rpc_ACCreateCursorEx**.

This method is called only by dependent clients. Servers SHOULD enforce limitations and security measures appropriate for dependent clients.<58>

This method is invoked at the dynamically assigned endpoint returned by the **R_QMGetRTQMServerPort** method when IP_HANDSHAKE (0x00000000) or IPX_HANDSHAKE (0x00000002) is the interface specified by the *fIP* parameter.

When processing this call, the server MUST:

- Determine if input parameter values violate constraints specified above. If an invalid parameter is detected, the server MUST take no further action and return MQ_ERROR_INVALID_PARAMETER (0xc00e0006).
- Look up the cursor entry in the cursor table, using the hQueue and hCursor key values.
- If not found, return a failure HRESULT.
- If the cursor is already associated with a remote cursor handle, or if the cursor is associated with a local queue, return a failure HRESULT.
- Associate the hRemoteCursor handle with the cursor entry.

3.1.4.21 rpc_ACHandleToFormatName (Opnum 26)

A dependent client calls the **rpc_ACHandleToFormatName** method to retrieve a format name for a queue handle.

```
HRESULT rpc_ACHandleToFormatName(
  [in] RPC_QUEUE_HANDLE hQueue,
  [in, range(0, 524288)] DWORD dwFormatNameRPCBufferLen,
  [in, out, unique, size_is(dwFormatNameRPCBufferLen),
length_is(dwFormatNameRPCBufferLen)]
    WCHAR* lpwcsFormatName,
  [in, out] DWORD* pdwLength
);
```

hQueue: MUST be an **RPC_QUEUE_HANDLE** acquired from the *phQueue* parameter of **rpc_QMOpenQueueInternal**. Prior to this method being invoked, the queue MUST NOT have been deleted, and the queue handle MUST NOT have been closed.

dwFormatNameRPCBufferLen: Length of the buffer (in Unicode characters) provided for the *lpwcsFormatName* parameter.

lpwcsFormatName: Pointer to a Unicode character buffer into which the server writes the format name (as specified in [MS-MQMQ]) for the queue identified by the *hQueue* parameter. The character buffer MUST be NULL-terminated by the server prior to returning, even if the provided buffer is not large enough to contain the entire format name string. MAY be NULL if *dwFormatNameRPCBufferLen* is 0x00000000. MUST NOT be NULL if *dwFormatNameRPCBufferLen* is nonzero.

pdwLength: On input, the maximum number of Unicode characters to write to the *lpwcsFormatName* buffer. This value MUST be equal to the *dwFormatNameRPCBufferLen* parameter. On return, the server MUST update the value of this parameter to indicate the complete length of the format name string for the queue identified by *hQueue*, without regard for the size of the provided buffer.

Return Values: If the provided buffer is long enough to contain the NULL-terminated format name for the queue identified by *hQueue*, the server MUST take the following actions:

- Copy the NULL-terminated format name into the lpwcsFormatName buffer.
- Set *pdwLength* to the length (in Unicode characters) of the format name, including the NULL-terminator.
- Return MQ_OK (0x00000000).

If the provided buffer is too small to contain the complete format name for the queue identified by *hQueue* (including the NULL-terminator), the server MUST take the following actions:

- If the buffer length (indicated by *pdwLength*) is greater than 0x00000000, and if *lpwcsFormatName* is non-NULL, copy the format name to the *lpwcsFormatName* buffer, truncated to fit the length indicated by the input value for *pdwLength*. The string MUST be NULL-terminated post-truncation.
- Set *pdwLength* to the length of the untruncated format name, including the NULL-terminator.
- Take no further action and return MQ_ERROR_FORMATNAME_BUFFER_TOO_SMALL (0xc00e001f).

If input parameter values violate constraints specified above, the server MUST take no further action and return MQ_ERROR_INVALID_PARAMETER (0xc00e0006).

If any other error occurs, the server MUST return a failure HRESULT,<59> and the client MUST treat all other failure HRESULTs identically. Additionally, if any other failure HRESULT is returned, the client MUST disregard all out-parameter values.

MQ_OK (0x00000000)
MQ_ERROR_INVALID_PARAMETER (0xc00e0006)

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

This method is called only by dependent clients. Servers SHOULD enforce limitations and security measures appropriate for dependent clients.<60>

This method is invoked at the dynamically assigned endpoint returned by the **R_QMGetRTQMServerPort** method when IP_HANDSHAKE (0x00000000) or IPX_HANDSHAKE (0x00000002) is the interface specified by the *fIP* parameter.

3.1.4.22 rpc_ACPurgeQueue (Opnum 27)

The **rpc_ACPurgeQueue** method is called by a dependent client to purge an opened queue.

```
HRESULT rpc_ACPurgeQueue(
  [in] RPC_QUEUE_HANDLE hQueue
);
```

hQueue: MUST be an **RPC_QUEUE_HANDLE** obtained from the *phQueue* parameter of the **rpc_QMOpenQueueInternal** method. Prior to this method being invoked, the queue MUST NOT have been deleted, and the queue handle MUST NOT have been closed.

Return Values: On success, this method MUST return MQ_OK (0x00000000); otherwise, the server MUST return a failure **HRESULT**,<61> and the client MUST treat all failure HRESULTs identically.

MQ_OK (0x00000000)
MQ_ERROR_INVALID_PARAMETER (0xc00e0006)

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

This method is called only by dependent clients. Servers SHOULD enforce limitations and security measures appropriate for dependent clients.<62>

This method is invoked at the dynamically assigned endpoint returned by the **R_QMGetRTQMServerPort** method when IP_HANDSHAKE (0x00000000) or IPX_HANDSHAKE (0x00000002) is the interface specified by the *fIP* parameter.

When processing this call, the server MUST:

- Determine if input parameter values violate constraints specified above. If an invalid parameter is detected, the server MUST take no further action and return MQ_ERROR_INVALID_PARAMETER (0xc00e0006).
- Look up the **RPC_QUEUE_HANDLE** in the QueueContextHandle table.
- If not found, return a failure HRESULT.
- Remove all messages from the message list associated with the queue, except messages which are locked or participating in a transactional operation.

3.1.4.23 R_QMQueryQMRegistryInternal (Opnum 28)

A dependent client calls the **R_QMQueryQMRegistryInternal** method to retrieve various string values from the supporting server.

```
HRESULT R_QMQueryQMRegistryInternal(
  [in] handle_t hBind,
  [in] DWORD dwQueryType,
```

```
  [out, string] WCHAR** lplpMQISServer
);
```

hBind: MUST be set to an RPC binding handle as described in [MS-RPCE] section 2.

dwQueryType: Specifies the type and format of the data to return to the caller via the *lplpMQISServer* parameter. MUST be one of the values in the following table:

Value	Meaning
0x00000000	A comma-delimited list of MQIS server names configured on the supporting server. If no MQIS servers are configured, the server MUST return an empty string. The string length MUST NOT exceed 1500 characters, including the NULL-terminator, and the server MUST omit items from the list to remain below this limit. The list of MQIS server names is maintained by the **DirectoryServiceServerArray** element defined by the Message Queuing (MSMQ) Directory Service Discovery Protocol ([MS-MQSD] section 2.2.3). <63>
0x00000001	The server's default time-to-reach-queue message property value, expressed in seconds, converted to a string.<64><65>
0x00000002	The GUID which represents the entire MSMQ forest.<66> See below for the curly braced GUID string representation to use. The string must use the "braceless" format.
0x00000003	A string representation of the supporting server version. This value is supported by Windows 2000 and higher. Windows NT servers MUST return a failure HRESULT.
0x00000004	The GUID for the server queue manager. The curly braced GUID string representation must uses a "braceless" format given below. This value is only supported by Windows 2000 and higher. Windows NT servers MUST return a failure HRESULT.<67>

The format for the comma-delimited list of MQIS server names (0x00000000) is given by the following augmented BNF:

```
list = [list ","] computer-name
computer-name = 1*15digit
digit = num-digit / uppercase-alpha-digit / lowercase-alpha-digit
                / special-digit
num-digit = %x30-39
uppercase-alpha-digit = %x41-5Alowercase-alpha-digit = %x61-7A
special-digit = "!" / "@" / "#" / "$" / "%" / "^" / "&" / "'"
               / ")" / "(" / "." / "-" / "_" / "{" / "}" / "~"
```

The GUID string for the MSMQ forest (0x00000002) uses the "braceless" format depicted in the following augmented BNF:

```
list = [list ","] computer-name
computer-name = 1*15digit
```

```
digit = num-digit / uppercase-alpha-digit / lowercase-alpha-digit
        / special-digit
num-digit = %x30-39
uppercase-alpha-digit = %x41-5Alowercase-alpha-digit = %x61-7A
special-digit = "!" / "@" / "#" / "$" / "%" / "^" / "&" / "'"
                / ")" / "(" / "." / "-" / "_" / "{" / "}" / "~"
```

The string format used for the supporting server version (0x00000003), depicted in augmented BNF, is as follows:

```
version = version-part "." version-part "." version-part
version-part = 1*4num-digit
num-digit = %x30-39
```

The GUID for the server queue manager (0x00000004) uses the following "braceless" format, depicted in augmented BNF:

```
braceless-guid = dword-part "-" word-part "-" word-part "-"
                 2byte-part "-" 6byte-part
dword-part = 2word-part
word-part = 2byte-part
byte-part = 2hex-digit
hex-digit = %x30-39 / %x41-46 / %x61-66
```

lplpMQISServer: On success, the server returns the string indicated by dwQueryType through this parameter. The server MAY set this parameter to NULL in the event of an error.

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

If input parameter values violate constraints specified above, the server MUST take no further action and return MQ_ERROR_INVALID_PARAMETER (0xc00e0006).

If any other error occurs, the server MUST return a failure **HRESULT**, and the client MUST treat all other failure HRESULTs identically. Additionally, if any other failure HRESULT is returned, the client MUST disregard all out-parameter values.

MQ_OK (0x00000000)
MQ_ERROR_INVALID_PARAMETER (0xc00e0006)

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

This method is called only by dependent clients. Servers SHOULD enforce limitations and security measures appropriate for dependent clients.<68>

This method is invoked at the dynamically assigned endpoint returned by the **R_QMGetRTQMServerPort** method when IP_HANDSHAKE (0x00000000) or IPX_HANDSHAKE (0x00000002) is the interface specified by the *fIP* parameter.

3.1.4.24 R_QMGetRTQMServerPort (Opnum 31)

The **R_QMGetRTQMServerPort** method returns an RPC port number, as specified in [MS-RPCE], for the requested combination of interface and protocol. The returned RPC port number MAY be used for all qmcomm and qmcomm2 methods.

```
DWORD R_QMGetRTQMServerPort(
  [in] handle_t hBind,
  [in] DWORD fIP
);
```

hBind: MUST be set to an RPC binding handle as described in [MS-RPCE] section 2.

fIP: Specifies the interface for which a port value is to be returned. One of the following values MUST be specified; otherwise, this method MUST return 0x00000000 to indicate failure.

Value	Meaning
IP_HANDSHAKE 0x00000000	Requests that the server return the RPC port number for the qmcomm and qmcomm2 interfaces bound to TCP/IP.
IP_READ 0x00000001	Requests that the server return the RPC port number for the qm2qm interface, as specified in [MS-MQQP], bound to TCP/IP.
IPX_HANDSHAKE 0x00000002	Requests that the server return the RPC port number for the qmcomm and qmcomm2 interfaces bound to SPX. Note that this value is not supported by Windows XP, Windows Server 2003, and Windows Vista and Windows Server 2008.<69>
IPX_READ 0x00000003	Requests that the server return the RPC port number for the qm2qm interface, as specified in [MS-MQQP], bound to SPX. Note that this value is not supported by Windows XP, Windows Server 2003, and Windows Vista and Windows Server 2008.<70>

Return Values: On success, this method returns a non-zero IP port value for the RPC interface specified by the *fIP* parameter. If an invalid value is specified for *fIP*, or if the requested interface is otherwise unavailable, or if any other error is encountered, this method MUST return 0x00000000.

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

As specified in section 3.1.3, this protocol configures a fixed listening endpoint at an RPC port number which may vary. For the interface and protocol specified by the *fIP* parameter, this method returns the RPC port number determined at server initialization time.

Security consideration: Servers MUST NOT enforce security limitations for this method, since clients MAY call this method before configuring RPC binding security. See section 5.1 for details.

3.1.5 Message Processing Events and Sequencing Rules for qmcomm2

The following methods comprise the Message Queuing (MSMQ): Queue Manager Client Protocol version 2 (qmcomm2) interface:

Methods in RPC Opnum Order

Method	Description
QMSendMessageInternalEx	Sends a message to the specified queue. Opnum: 0
rpc_ACSendMessageEx	Sends a message to the specified queue. Opnum: 1
rpc_ACReceiveMessageEx	Receives a message from the specified queue. Opnum: 2
rpc_ACCreateCursorEx	Creates a cursor for accessing the specified queue. Opnum: 3

3.1.5.1 QMSendMessageInternalEx (Opnum 0)

A dependent client invokes **QMSendMessageInternalEx** if the server returns STATUS_RETRY (0xc000022d) from a prior call to **rpc_ACSendMessageEx**. Implementations of this protocol SHOULD NOT return STATUS_RETRY from **rpc_ACSendMessageEx**, rendering this method unnecessary. Such implementations MUST take no action when **QMSendMessageInternalEx** is invoked, and return MQ_ERROR_ILLEGAL_OPERATION (0xc00e0064).

```
HRESULT QMSendMessageInternalEx(
  [in] handle_t hBind,
  [in] struct QUEUE_FORMAT* pQueueFormat,
  [in] struct CACTransferBufferV2* pCacTB,
  [in, out, unique] OBJECTID* pMessageID
);
```

hBind: MUST be set to an RPC binding handle as described in [MS-RPCE] section 2.

pQueueFormat: MUST be a pointer to a **QUEUE_FORMAT** structure, as specified in [MS-MQMQ] section 2.2.7 which identifies an existing queue to be opened. MUST NOT be NULL, and MUST conform to the format-name syntax rules defined in [MS-MQMQ]. The queue identified by *pQueueFormat* MUST be local to the supporting server, and MUST be successfully openable via a call to **rpc_QMOpenQueueInternal** with a *dwDesiredAccess* level of MQ_SEND_ACCESS (0x00000002).

pCacTB: A CACTransferBufferV2 structure pointer as described in section 2.2.2.3. See the identical parameter in section 3.1.5.2 for details on this parameter.

pMessageID: An OBJECTID as defined in [MS-MQMQ] section 2.2.8. See the identical parameter in section 3.1.5.2 for details on this parameter.

Return Values: On success, this method MUST return MQ_OK (0x00000000); otherwise, the server MUST return a failure **HRESULT**,<71> and the client MUST treat all failure HRESULTs identically. Additionally, if a failure HRESULT is returned, the client MUST disregard all out-parameter values.

MQ_OK (0x00000000)
MQ_ERROR_INVALID_PARAMETER (0xc00e0006)
MQ_ERROR_ILLEGAL_OPERATION (0xc00e0064)

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

This method is called only by dependent clients. Servers SHOULD enforce limitations and security measures appropriate for dependent clients.<72>

This method is invoked at the dynamically assigned endpoint returned by the **R_QMGetRTQMServerPort** method when IP_HANDSHAKE (0x00000000) or IPX_HANDSHAKE (0x00000002) is the interface specified by the *fIP* parameter.

When processing this call, the server MUST:

- Determine if input parameter values violate constraints specified above. If an invalid parameter is detected, the server MUST take no further action and return MQ_ERROR_INVALID_PARAMETER (0xc00e0006).
- Open the queue identified by *pQueueFormat* by invoking **rpc_QMOpenQueueInternal** with *dwDesiredAccess* of MQ_SEND_ACCESS (0x00000002).
- If this process is successful:
  - With the queue handle obtained from **rpc_QMOpenQueueInternal**, invoke **rpc_ACSendMessageEx** as specified in section 3.1.5.2.
  - Return MQ_OK (0x00000000).
- Else:
  - Return a failure HRESULT.

3.1.5.2 rpc_ACSendMessageEx (Opnum 1)

A dependent client calls the **rpc_ACSendMessageEx** method to place a message into a message queue for delivery.

```
HRESULT rpc_ACSendMessageEx(
  [in] RPC_QUEUE_HANDLE hQueue,
  [in] struct CACTransferBufferV2* ptb,
  [in, out, unique] OBJECTID* pMessageID
);
```

hQueue: MUST be an **RPC_QUEUE_HANDLE** obtained from the *phQueue* parameter of the **rpc_QMOpenQueueInternal** method called with the *dwDesiredAccess* parameter set to MQ_SEND_ACCESS. Prior to this method being invoked, the queue MUST NOT have been deleted, and the queue handle MUST NOT have been closed.

ptb: MUST NOT be NULL. ptb points to a CACTransferBufferV2 structure as defined in section 2.2.2.3. Refer to section 2.2.2.3 for definitions of the CACTransferBufferV2 member fields. Constraints for the member fields are defined below. In the section below, "ptb.old" is used as shorthand to refer to the CACTransferBufferOld member of the CACTransferBufferV2 structure.

ptb.old.uTransferType MUST be CACTB_SEND (0x00000000).

ptb.old.Send.pAdminQueueFormat MAY be NULL, in which case no administration queue format name is associated with the message. If not NULL, ptb.old.Send.pAdminQueueFormat MUST point to a QUEUE_FORMAT structure as defined in [MS-MQMQ] section 2.2.7.

ptb.old.Send.pResponseQueueFormat MAY be NULL, in which case no response queue format name is associated with the message. If not NULL, ptb.old.Send. pResponseQueueFormat MUST point to a QUEUE_FORMAT structure as defined in [MS-MQMQ] section 2.2.7.

If the queue identified *hQueue* was opened using a direct format name as specified in [MS-MQMQ] section 2.1.2, ptb.old.pulPrivLevel MUST be NULL or, if not NULL, MUST point to the value MQMSG_PRIV_LEVEL_NONE (0x00000000). Encryption MUST NOT be requested for queues opened with direct format names.

If the queue identified by *hQueue* is not an outgoing queue (rather, it is a queue which is local to the supporting server), and ptb.bEncrypted is not 0x00, the server MAY return STATUS_RETRY (0xc000022d) and take no action.<73>

ptb.old.pPriority MAY be NULL; otherwise, the value MUST be from 0x00 to 0x07 inclusive. If the value is NULL, the server MUST substitute the default value of 0x03.

ptb.old.pTrace MAY be NULL, in which case the server MUST substitute the default value of 0x00.

If ptb.old.ulAbsoluteTimeToQueue is 0x00000000, the server MUST substitute the default value of 0xffffffff.

ptb.old.ppMessageID MAY be NULL. If not NULL, the server MUST ignore the in-value, and the server MUST return the same value for ptb.old.ppMessageID as for pMessageID.

ptb.old.ppConnectorType MAY be NULL. If NULL, then no connector type value is associated with the message.

ptb.old.pDelivery MAY be NULL, in which case the server MUST substitute the default value of 0x00. However, if ptb.old.pUow contains a nonzero value, the server MUST substitute the value 0x01 for ptb.old.pDelivery, since transactional messages are by definition stored as recoverable.

ptb.old.pAuditing MAY be NULL, in which case the server MUST substitute the default value of 0x00.

ptb.old.pClass MAY be NULL, in which case the server MUST substitute the default value of 0x0000. This field may be used by connector applications to produce acknowledgment messages. Typical applications will always specify MQMSG_CLASS_NORMAL (0x0000).

ptb.old.ppCorrelationID MAY be NULL, in which case the server MUST substitute the default value by filling the array of bytes with hexadecimal zeros (0x00).

ptb.old.pAcknowledge MAY be NULL, in which case the server MUST substitute the default value of 0x00.

ptb.old.pApplicationTag MAY be NULL, in which case the server MUST substitute the default value of 0x00000000.

ptb.old.ppTitle MAY be NULL, in which case the server MUST treat the value as an empty string and MUST ignore the value of ptb.old.ulTitleBufferSizeInWCHARs. If ptb.old.ppTitle is NOT NULL, the server MUST take the number of Unicode characters indicated by ptb.old.ulTitleBufferSizeInWCHARs. If ptb.old.ulTitleBufferSizeInWCHARs is greater than 255, the value MUST be truncated to 255. The server MUST NULL-terminate the resulting character array.

ptb.old.ppMsgExtension MAY be NULL, in which case no extension array is associated with the message and the server MUST ignore the value of ptb.old.ulMsgExtensionBufferInBytes. If ptb.old.*ppMsgExtension* is NOT NULL, the server MUST take the number of bytes indicated by ptb.old.ulMsgExtensionBufferInBytes. The buffer is an opaque array of bytes and NULL-termination is not required.

ptb.old.ppBody MAY be NULL, in which case no body array is associated with the message and the server MUST ignore the values of ptb.old.ulBodyBufferSizeInBytes and ptb.old.ulAllocBodyBufferInBytes. If ptb.old.ppBody is NOT NULL, the server MUST take the number of bytes indicated by ptb.old.ulBodyBufferSizeInBytes, and allocate body storage for the number of bytes indicated by ptb.old.ulAllocBodyBufferInBytes. The message body is an opaque array of bytes and NULL-termination is not required.

ptb.old.pulPrivLevel MAY be NULL, in which case the server MUST substitute the default value of 0x00000000.

ptb.old.pulHashAlg MAY be NULL, in which case theserver SHOULD substitute the default value of MQMSG_CALG_SHA1 (0x00008004).<74> ptb.old.pulEncryptAlg MAY be NULL, in which case the server SHOULD substitute the default value of MQMSG_CALG_RC4 (0x00006801).<75>

ptb.old.pulBodyType MAY be NULL, in which case the server MUST substitute the default value of 0x00000000.

ptb.old.ppSenderCert MAY be NULL if ptb.old.ulSenderCertLen is 0x00000000, in which case an X509 certificate for the sender is not associated with the message.

ptb.old.pulSenderIDType MUST NOT be NULL if ptb.old.uSenderIDLen is nonzero.

ptb.old.pSenderID MAY be NULL if ptb.old.uSenderIDLen is zero and ptb.old.pulSenderIDType is MQMSG_SENDERID_TYPE_NONE (0x00000000), in which case a SID is not associated with the message.

ptb.old.ppSymmKeys MAY be NULL if ptb.old.ulSymmKeysSize is zero (0x00000000), in which case an encrypted symmetric key is not associated with the message. Otherwise, ptb.old.ppSymKeys MUST contain the symmetric key used to encrypt the message body. The symmetric key MUST be encrypted with the public key of the recipient QM. The manner by which the public key for the recipient QM is obtained is beyond the scope of this network protocol.

If ptb.old.ulSignatureSize is 0x00000000: no digital signature is associated with the message.

Else, if ptb.old.ulSignatureSize is not 0x00000000:

- If ptb.old.fDefaultProvider is 0x00000000, ptb.old.ppwcsProvName MUST NOT be NULL. Note that ptb.old.ulProvNameLen is used only to affect RPC marshaling of the ptb.old.ppwcsProvName buffer. The server MUST otherwise ignore ptb.old.ulProvNameLen and treat ptb.old.ppwcsProvName as a NULL-terminated string.
- If ptb.old.fDefaultProvider is not 0x00000000, ptb.old.pulProvType MUST NOT be NULL, and MUST contain PROV_RSA_FULL (0x00000001).
- If ptb.old.pulAuthProvNameLenProp is NULL:
  - If not NULL, the ptb.old.ppSignature buffer contains a simple array of bytes containing the **MSMQ 1.0 digital signature**. The byte length of the buffer is indicated by ptb.old.ulSignatureSize.

- Else, if ptb.old.pulAuthProvNameLenProp is NOT NULL:
  - If not NULL, the ptb.old.ppSignature buffer contains two distinct byte array parts. The first part MUST be ignored by the server. The second part contains an **MSMQ 2.0 digital signature**.
  - The byte length of the first part is indicated by subtracting the length of the second part from ptb.old.ulSignatureSize. (Thus length([first part]) + length([second part]) = ptb.old.ulSignatureSize.)
  - The byte length of the second part is indicated by subtracting ptb.old.ulProvNameLen from ptb.old.pulAuthProvNameLenProp.
  - The second part begins immediately after the first.

The following fields MUST be ignored by the server:

- ptb.old.Receive
- ptb.old.CreateCursor
- ptb.old.pSentTime
- ptb.old.pArrivedTime
- ptb.old.pBodySize
- ptb.old.pulTitleBufferSizeInWCHARs
- ptb.old.pulRelativeTimeToQueue
- ptb.old.pulRelativeTimeToLive
- ptb.old.pulSenderIDLenProp
- ptb.old.ulAuthLevel
- ptb.old.pAuthenticated
- ptb.old.bAuthenticated
- ptb.old.pulSenderCertLenProp
- ptb.old.pulSymmKeysSizeProp
- ptb.old.pulSignatureSizeProp
- ptb.old.pulAuthProvNameLenProp<76>
- ptb.old.ppSrcQMID
- ptb.old.pMsgExtensionSize
- ptb.old.pulVersion
- ptb.pbFirstInXact
- ptb.pbLastInXact

- ptb.ppXactID

pMessageID: An OBJECTID as defined in [MS-MQMQ] section 2. This value MAY be NULL. If not NULL, the server MUST return a message identifier for the new message if this method succeeds. The message identifier returned via this parameter MUST be identical to the value returned for ptb.old.ppMessageID.

Return Values: On success, this method MUST return MQ_OK (0x00000000); otherwise, the server MUST return a failure **HRESULT**,<77> and the client MUST treat all failure HRESULTs identically. Additionally, if a failure HRESULT is returned, the client MUST disregard all out-parameter values.

MQ_OK (0x00000000)
MQ_ERROR_INVALID_PARAMETER (0xc00e0006)

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

This method is called only by dependent clients. Servers SHOULD enforce limitations and security measures appropriate for dependent clients.<78>

This method is invoked at the dynamically assigned endpoint returned by the **R_QMGetRTQMServerPort** method when IP_HANDSHAKE (0x00000000) or IPX_HANDSHAKE (0x00000002) is the interface specified by the *fIP* parameter.

Security Considerations: The caller may request that the server perform security related operations such as signing and encrypting the message. These operations are requested by setting members of the ptb.CACTransferBufferOld structure.

When processing this call, the server MUST:

- Determine if input parameter values violate constraints specified above. If an invalid parameter is detected, the server MUST take no further action and return MQ_ERROR_INVALID_PARAMETER (0xc00e0006).
- Look up the hQueue **RPC_QUEUE_HANDLE** in the QueueContextHandle table.
- If not found:
  - Return a failure HRESULT.
- If ptb.old.pUow is non-NULL:
  - Using ptb.old.pUow as the transaction identifier key, look up the transaction in the Transaction Table.
  - If not found:
    - Return a failure HRESULT. The client did not enlist the transaction before attempting to perform a transactional send operation.
- Verify that the properties of the message satisfy constraints defined by the target queue properties. Storage constraints, such as quota limitations, should be verified. If the message cannot be placed in the queue, the server MUST return a failure HRESULT.
- Generate initial values for read-only messages properties or properties for which the client did not specify initial values. These include:

- Creating a message identifier for the new message, and placing the value in the **ptb.old.ppMessageID** field. The message identifier consists of the server GUID and an unsigned sequential number from an increasing sequence which MUST be preserved across service interruptions.
- Replacing NULL parameters with appropriate default values as described above.

- If the message body is encrypted (ptb.old.bEncrypted is not 0x00, and *ppSymmKeys* is not NULL, and ppSymmKeysSize is not 0x00000000), and the destination queue is located on the supporting server, the message body MUST be decrypted prior to adding the message to the queue. Using the algorithm indicated by ptb.old.pulEncryptAlg, decrypt the symmetric key contained in ppSymmKeysSize with the server's private key. Finally, use the decrypted symmetric key to decrypt the message body.
- If ptb.old.pUow is non-NULL: (if the message is being sent in the context of a transaction)
  - Create a new "send message" record in the transactional operation list for the associated transaction identified by ptb.old.pUow. The new record MUST contain the implementation-specific information necessary to place the message in the target queue when the transaction is committed. After successfully creating the new entry in the transactional operation list, return S_OK (0x00000000).
- Otherwise, if ptb.old.pUow is NULL, add the message to the target queue message list.
- If ptb.old.pDelivery is 0x01, the server SHOULD durably commit the queue storage prior to returning S_OK. This ensures the message has been stored durably prior to indicating success to the client.
- Return S_OK.

3.1.5.3 rpc_ACReceiveMessageEx (Opnum 2)

A dependent client calls **rpc_ACReceiveMessageEx** to peek or receive a message from a message queue.

```
HRESULT rpc_ACReceiveMessageEx(
  [in] handle_t hBind,
  [in] DWORD hQMContext,
  [in, out] struct CACTransferBufferV2* ptb
);
```

hBind: MUST be set to an RPC binding handle as described in [MS-RPCE] section 2.

hQMContext: A queue context value obtained from the pdwQMContext parameter of **rpc_QMOpenQueueInternal**. The queue MUST have been opened with MQ_PEEK_ACCESS or MQ_RECEIVE_ACCESS as the dwDesiredAccess parameter when **rpc_QMOpenQueueInternal** was called. Prior to this method being invoked, the queue MUST NOT have been deleted, and the queue handle for the open queue MUST NOT have been closed.

ptb: MUST NOT be NULL. ptb points to a CACTransferBufferV2 structure as defined in section 2.2.2.3. See section 2.2.2.3 for definitions of the CACTransferBufferV2 member fields. Constraints for the member fields are defined below. In the section below, "ptb.old" is used as shorthand referring to the CACTransferBufferOld member of the CACTransferBufferV2 structure.

ptb.old.uTransferType MUST be CACTB_RECEIVE (0x00000001).

ptb.old.Receive.Action MUST contain one of the following values: 0x00000000 (MQ_ACTION_RECEIVE), 0x80000000 (MQ_ACTION_PEEK_CURRENT) or 0x80000001 (MQ_ACTION_PEEK_NEXT).

ptb.old.Receive.Cursor MAY be 0x00000000, in which case no cursor is associated with the receive operation. Otherwise, ptb.old.Receive.Cursor MUST contain a cursor handle obtained from the pcc.hCursor parameter of **rpc_ACCreateCursorEx**. The cursor MUST have been created using the queue handle associated with the queue context value provided for the *hQMContext* parameter, and the cursor MUST NOT have been closed prior to this call.

ptb.old.Receive.ulResponseFormatNameLen is used only for RPC marshaling of the ppResponseFormatName buffer and SHOULD be otherwise ignored by the server. If ptb.old.Receive.ppResponseFormatName is NULL, this value MUST be 0x00000000.

On input, ptb.old.Receive.pulResponseFormatNameLenProp indicates the Unicode character length of the buffer contained in ppResponseFormatName. On output, the server MUST set this value to indicate the full length of the response queue format name associated with the message being retrieved.

ptb.old.Receive.ppResponseFormatName MAY be NULL; otherwise, on successful retrieval of a message and prior to filling the buffer, the server MUST verify that the pulResponseFormatNameLenProp field indicates that the buffer is large enough to contain the response queue format name for the retrieved message.

ptb.old.Receive.ulAdminFormatNameLen is used only for RPC marshaling of the ppAdminFormatName buffer and SHOULD be otherwise ignored by the server. If ptb.old.Receive.ppAdminFormatName is NULL, this value MUST be 0x00000000.

On input, ptb.old.Receive.pulAdminFormatNameLenProp indicates the Unicode character length of the buffer contained in ppAdminFormatName. On output, the server MUST set this value to indicate the full length of the administration queue format name associated with the message being retrieved.

ptb.old.Receive.ppAdminFormatName MAY be NULL; otherwise, on successful retrieval of a message and prior to filling the buffer, the server MUST verify that the pulAdminFormatNameLenProp field indicates that the buffer is large enough to contain the administration queue format name for the retrieved message.

ptb.old.Receive.ulDestFormatNameLen is used only for RPC marshaling of the ppDestFormatName buffer and SHOULD be otherwise ignored by the server. If ptb.old.Receive.ppDestFormatName is NULL, this value MUST be 0x00000000.

On input, ptb.old.Receive.pulDestFormatNameLenProp indicates the Unicode character length of the buffer contained in ppDestFormatName. On output, the server MUST set this value to indicate the full length of the destination queue format name associated with the message being retrieved.

ptb.old.Receive.ppDestFormatName MAY be NULL; otherwise, on successful retrieval of a message and prior to filling the buffer, the server MUST verify that the pulDestFormatNameLenProp field indicates that the buffer is large enough to contain the destination queue format name for the retrieved message.

ptb.old.Receive.ulOrderingFormatNameLen is used only for RPC marshaling of the ppOrderingFormatName buffer and SHOULD be otherwise ignored by the server. If ptb.old.Receive.ppAdminFormatName is NULL, this value MUST be 0x00000000.

On input, ptb.old.Receive.pulOrderingFormatNameLenProp indicates the Unicode character length of the buffer contained in ppOrderingFormatName. On output, the server MUST set this value to indicate the full length of the order queue format name associated with the message being retrieved.

ptb.old.Receive.ppOrderingFormatName MAY be NULL; otherwise, on successful retrieval of a message and prior to filling the buffer, the server MUST verify that the pulOrderingFormatNameLenProp field indicates that the buffer is large enough to contain the order queue format name for the retrieved message.

ptb.old.ppBody MAY be NULL; otherwise, on successful retrieval of a message, prior to filling the buffer, the server MUST verify that the ulBodyBufferSizeInBytes field indicates that the buffer is large enough to contain the message body for the retrieved message. The byte length of the complete body for the retrieved message MUST be returned in the pBodySize field, if it is not NULL.

ptb.old.ulBodyBufferSizeInBytes MUST be 0x00000000 if ptb.old.ppBody is NULL.

ptb.old.ulAllocBodyBufferInBytes is used only for RPC marshaling of the ppBody buffer and SHOULD be otherwise ignored by the server. If ppBody is NULL, this value MUST be 0x00000000.

ptb.old.pBodySize MAY be NULL; otherwise, on successful retrieval of a message, the server MUST set this value to the byte length of the message body.

ptb.old.ulTitleBufferSizeInWCHARs is used only for RPC marshaling of the ptb.old.ppTitle buffer and otherwise SHOULD be ignored by the server. If ptb.old.ppTitle is NULL, this value MUST be 0x00000000.

On input, ptb.old..pulTitleBufferSizeInWCHARs indicates the Unicode character length of the buffer contained in ppTitle. On output, the server MUST set this value to indicate the full length of the message label associated with the message being retrieved.

ptb.old.ppTitle MAY be NULL; otherwise, on successful retrieval of a message, prior to filling the buffer, the server MUST verify that the pulTitleBufferSizeInWCHARs field indicates that the buffer is large enough to contain the message label for the retrieved message.

ptb.old.ppMsgExtension MAY be NULL; otherwise, on successful retrieval of a message, prior to filling the buffer, the server MUST verify that the ptb.old.ulMsgExtensionBufferInBytes field indicates that the buffer is large enough to contain the message extension array for the retrieved message.

ptb.old.ulMsgExtensionBufferInBytes MUST be 0x00000000 if ptb.old.ppMsgExtension is NULL.

ptb.old.pMsgExtensionSize MAY be NULL; otherwise, the server MUST return the full length of the retrieved message extension array in ptb.old.pMsgExtensionSize.

ptb.old.pUow MAY be NULL, in which case the Receive operation is not associated with a transaction. Otherwise, pUow MUST contain a 16-byte transaction identifier which has been enlisted by a prior call to **R_QMEnlistTransaction** or **R_QMEnlistInternalTransaction**.

ptb.old.ppSenderID MAY be NULL; otherwise, on successful retrieval of a message and prior to filling the buffer, the server MUST verify that the ptb.old.uSenderIDLen field indicates that the buffer is large enough to contain the sender ID buffer for the retrieved message.

ptb.old.pulSenderIDLenProp MAY be NULL; otherwise, the server MUST return the full length of the sender ID buffer for the retrieved message in ptb.old.pulSenderIDLenProp.

ptb.old.ppwcsProvName MAY be NULL; otherwise, prior to filling the buffer, the server MUST verify that the ptb.old.ulProvNameLen field indicates that the buffer is large enough to contain the NULL-terminated CSP name string. If the retrieved message does not include a CSP name buffer, the server MUST return 0x00000000 for ptb.old.pulAuthProvNameLenProp if the pulAuthProvNameLenProp pointer is not NULL.

ptb.old.pulAuthProvNameLenProp MAY be NULL; otherwise, the server MUST return the length of the CSP name buffer for the retrieved message in ptb.old.pulAuthProvNameLenProp, or 0x00000000 if the value was not included in the retrieved message.

ptb.old.ppSenderCert MAY be NULL; otherwise, prior to filling the buffer, the server MUST verify that the ptb.old.ulSenderCertLen field indicates that the buffer is large enough to contain the sender certificate buffer. If the retrieved message does not include a sender certificate, the server MUST return 0x00000000 for ptb.old.pulSenderCertLenProp if the pulSenderCertLenProp pointer is not NULL.

ptb.old.pulSenderCertLenProp MAY be NULL; otherwise, the server MUST return the length of the sender certificate buffer for the retrieved message in ptb.old.pulSenderCertLenProp, or 0x00000000 if the value is not included in the retrieved message.

ptb.old.ppSymmKeys MAY be NULL; otherwise, prior to filling the buffer, the server MUST verify that the ptb.old.ulSymmKeysSize field indicates that the buffer is large enough to contain the symmetric key buffer. If the retrieved message does not include a symmetric key, the server MUST return 0x00000000 for ptb.old.pulSymmKeysSizeProp if the pulSymmKeysSizeProp pointer is not NULL.

ptb.old.pulSymmKeysSizeProp MAY be NULL; otherwise, the server MUST return the length of the symmetric key buffer for the retrieved message in ptb.old.pulSymmKeysSizeProp, or 0x00000000 if the value is not included in the retrieved message.

ptb.old.ppSignature MAY be NULL; otherwise, prior to filling the buffer, the server MUST verify that the ptb.old.ulSignatureSize field indicates that the buffer is large enough to contain the signed hash buffer. If the retrieved message does not include a signed hash, the server MUST return 0x00000000 for ptb.old.pulSignatureSizeProp if the pulSignatureSizeProp pointer is not NULL.

ptb.old.pulSignatureSizeProp MAY be NULL; otherwise, the server MUST return the length of the signed hash buffer for the retrieved message in ptb.old.pulSignatureSizeProp, or 0x00000000 if the value is not included in the retrieved message.

The following fields MAY be NULL, in which case the server MUST ignore them. On successful retrieval of a message, the server MUST return the appropriate message property value into each non_NULL field. See section 2.2.2.2 for definitions of these fields:

- ptb.old.pClass
- ptb.old.ppMessageID
- ptb.old.ppCorrelationID
- ptb.old.pSentTime
- ptb.old.pArrivedTime
- ptb.old.pPriority
- ptb.old.pDelivery

- ptb.old.pAcknowledge
- ptb.old.pAuditing
- ptb.old.pApplicationTag
- ptb.old.pulRelativeTimeToQueue
- ptb.old.pulRelativeTimeToLive
- ptb.old.pTrace
- ptb.old.pulPrivLevel
- ptb.old.pAuthenticated
- ptb.old.pulHashAlg
- ptb.old.pulEncryptAlg
- ptb.old.pulProvType
- ptb.old.pulSenderIDType
- ptb.old.ppSrcQMID
- ptb.old.ppConnectorType
- ptb.old.pulBodyType
- ptb.old.pulVersion
- ptb.pbFirstInXact
- ptb.pbLastInXact
- ptb.ppXactID

The following fields MUST be ignored by the server:

- ptb.old.Send
- ptb.old.CreateCursor
- ptb.old.Receive.Asynchronous
- ptb.old.ulAbsoluteTimeToQueue
- ptb.old.ulRelativeTimeToLive
- ptb.old.ulAuthLevel
- ptb.old.bEncrypted
- ptb.old.bAuthenticated
- ptb.old.fDefaultProvider

Return Values: On success, this method MUST return MQ_OK (0x00000000); otherwise, the server MUST return a failure **HRESULT**,<79> and the client MUST treat all failure HRESULTs

identically. Additionally, if a failure HRESULT is returned, the client MUST disregard all out-parameter values.

MQ_OK (0x00000000)
MQ_ERROR_INVALID_PARAMETER (0xc00e0006)

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

This method is called only by dependent clients. Servers SHOULD enforce limitations and security measures appropriate for dependent clients.<80>

This method is invoked at the dynamically assigned endpoint returned by the **R_QMGetRTQMServerPort** method when IP_HANDSHAKE (0x00000000) or IPX_HANDSHAKE (0x00000002) is the interface specified by the *fIP* parameter.

When processing this call, the server MUST:

- Determine if input parameter values violate constraints specified above. If an invalid parameter is detected, the server MUST take no further action and return MQ_ERROR_INVALID_PARAMETER (0xc00e0006).
- Search the QueueContextHandle Table for an entry that has a *queue context value* that matches *hQMContext*.
  - If not found, return a failure HRESULT.
  - If found:
    - Confirm that the queue was not opened with dwDesiredAccess of MQ_SEND_ACCESS (0x00000002); otherwise, return a failure HRESULT.
    - If the queue identified by hQMContext was opened for peek access, confirm that ptb.old.Receive.Action is not MQ_ACTION_RECEIVE (0x00000000); otherwise, return a failure HRESULT.
    - If the queue is not local to the supporting server (the queue handle is a proxy for a remote queue), delegate the read operation to the qm2qm RPC interface, as defined in [MS-MQQP].
- If ptb.old.pUow is non-NULL:
  - Using ptb.old.pUow as the transaction identifier key, look up the transaction in the transaction table.
  - If not found:
    - Return a failure HRESULT. The client did not enlist the transaction before attempting to perform a transactional send operation.
  - If found:
    - Verify that ptb.old.Receive.Action is MQ_ACTION_RECEIVE (0x00000000). If not, return a failure HRESULT.

- If ptb.old.Receive.Cursor is not zero (0x00000000), the client has specified a cursor for the receive or peek operation. Look up the cursor in the cursor list for the queue identified by hQMContext. If not found, return a failure HRESULT.
- Locate a message to be retrieved in the message list for the queue identified by hQMContext:
  - The table below specifies the server behavior which results from the combination of the ptb.old.Receive.Action and ptb.old.Receive.Cursor parameters. If ptb.old.Receive.Cursor is zero (0x00000000), a cursor is not specified by the client ("No" in the table below).

Action	Cursor	Server behavior
MQ_ACTION_RECEIVE	No	Retrieve message at the front of the queue.
MQ_ACTION_PEEK	No	Retrieve message at the front of the queue.
MQ_ACTION_PEEK_NEXT	No	Invalid combination. Return a failure HRESULT.
MQ_ACTION_RECEIVE	Yes	Retrieve message at the cursor location.
MQ_ACTION_PEEK	Yes	Retrieve message at the cursor location.
MQ_ACTION_PEEK_NEXT	Yes	Retrieve next message succeeding the cursor location.

- If a message is not found:
  - If ptb.old.Receive.RequestTimeout is 0x00000000, return MQ_ERROR_MESSAGE_NOT_FOUND (0xc00e0008).
  - If ptb.old.Receive.RequestTimeout is not 0x00000000:
    - Add a new entry to the outstanding receive list containing the parameters of this call.
    - Create a timer which expires at current time + ptb.old.Receive.RequestTimeout.
    - If a message is still not available when the timer expires, return MQ_ERROR_IO_TIMEOUT (0xc00e001b).
- Validate that variable-length message property buffers in the **CACTransferBufferV2** structure provided by the client are large enough to contain the corresponding buffers in the retrieved message. If not, return a failure HRESULT.
- If ptb.old.pUow is nonzero (0x00000000):
  - Mark the retrieved message as locked.
  - Add a new entry to the transactional operation list associated with the entry in the transaction table identified by ptb.old.pUow.
- Else, if ptb.old.pUow is zero (0x00000000):
  - Fill the CACTransferBuffer2 structure with the contents of the retrieved message.
  - If ptb.old.Receive.Action is MQ_ACTION_RECEIVE (0x00000000), delete the retrieved message from the message list.
- If ptb.old.Receive.Cursor is nonzero (0x00000000), advance the state of the cursor as shown in section 3.1.1.9.

3.1.5.4 rpc_ACCreateCursorEx (Opnum 3)

A dependent client calls **rpc_ACCreateCursorEx** to create a cursor for use when peeking and receiving from a message queue.

```
HRESULT rpc_ACCreateCursorEx(
  [in] RPC_QUEUE_HANDLE hQueue,
  [in, out] struct CACCreateRemoteCursor* pcc
);
```

hQueue: MUST be an **RPC_QUEUE_HANDLE** acquired from the *phQueue* parameter of **rpc_QMOpenQueueInternal**. Prior to this method being invoked, the queue MUST NOT have been deleted, and the queue handle MUST NOT have been closed.

pcc: A pointer to a CACCreateRemoteCursor structure as defined in section 2.2.2.4. MUST NOT be NULL.

Return Values: On success, this method MUST return MQ_OK (0x00000000); otherwise, the server MUST return a failure **HRESULT**,<81> and the client MUST treat all failure HRESULTs identically. Additionally, if a failure HRESULT is returned, the client MUST disregard all out-parameter values.

MQ_OK (0x00000000)
MQ_ERROR_INVALID_PARAMETER (0xc00e0006)

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

This method is called only by dependent clients. Servers SHOULD enforce limitations and security measures appropriate for dependent clients.<82>

This method is invoked at the dynamically assigned endpoint returned by the **R_QMGetRTQMServerPort** method when IP_HANDSHAKE (0x00000000) or IPX_HANDSHAKE (0x00000002) is the interface specified by the *fIP* parameter.

When processing this call, the server SHOULD:<83>

- Determine if input parameter values violate constraints specified above. If an invalid parameter is detected, the server MUST take no further action and return MQ_ERROR_INVALID_PARAMETER (0xc00e0006).
- Look up the queue identified by *hQueue* in the queue table.
- If not found, return a failure HRESULT.
- If the queue identified by *hQueue* is local to the supporting server:
  - Add a new entry to the cursor table. The new cursor MUST be in the uninitialized state.
  - Return a handle for the new cursor via the pcc.*hCursor* parameter. The other members of pcc may be ignored.
- Else, if the queue identified by *hQueue* is not local to the supporting server:

- Invoke the **R_QMCreateRemoteCursor** method at the remote server, as described in [MS-MQRR].
- If the RemoteRead interface, as described in [MS-MQRR] is not accessible at the remote server, invoke the R_QMCreateRemoteCursor method at the remote server, as described in section 3.1.4.4.
- If neither of these means of creating the remote cursor is successful, return a failure HRESULT.

3.1.6 Timer Events

The server MUST maintain a timer associated with each **rpc_ACReceiveMessageEx** request. The length of time in milliseconds is specified by the client in the ptb.Receive.RequestTimeout parameter. If the server is unable to satisfy the request before the timer expires, the server MUST return MQ_ERROR_IO_TIMEOUT (0xc00e001b) to the client and remove the associated entry from the outstanding receive list.

3.1.7 Other Local Events

The following local events trigger operations on the server:

- Transaction Commit notification
- Transaction Abort notification
- Message Added to Queue notification

3.1.7.1 Transaction Commit Notification

This event is received when a transaction coordinator notifies the server of the outcome of a transaction in which the server has been enlisted. The event notification MUST specify the transaction identifier for the transaction being committed. For more information, see [MS-DTCO] section 3.5.4.5.

While processing this event, the server MUST:

- Look up the transaction in the transaction table.
- If found:
  - For each entry in the associated transactional operation list:
    - If the entry is a "Send Message" operation:
      - Insert the message created during the call to rpc_ACSendMessageEx into the associated queue.
    - If the entry is a "Receive Message" operation:
      - Remove the message from the associated queue.
  - Remove the entry from the transaction table.
- Else:
  - Do nothing.

3.1.7.2 Transaction Abort Notification

This event is received when a transaction coordinator notifies the server of the outcome of a transaction in which the server has been enlisted. The event notification MUST specify the transaction identifier for the transaction being aborted. For more information, see [MS-DTCO] section 3.5.4.5.

While processing this event, the server MUST:

- Look up the transaction in the transaction table.
- If found:
  - For each entry in the associated transactional operation list:
    - Unlock the message associated with the operation.
  - Remove the entry from the transaction table.
- Else:
  - Do nothing.

3.1.7.3 Message Added to Queue Notification

This event is received when a message is added to a local queue.

While processing this event, the server MUST:

- For each item in the outstanding receive list:
  - If the new message was added to the same queue that the outstanding receive is blocked against:
    - Satisfy the outstanding receive, as described in section 3.1.5.3.
    - Remove the outstanding receive item from the list.
    - Discontinue iterating the outstanding receive list and take no further action.

3.2 Qmcomm and Qmcomm2 Client Details

3.2.1 Abstract Data Model

Clients MUST maintain the following data elements:

- QueueContextHandle
- CursorContextValue

3.2.1.1 QueueContextHandle

An RPC context handle representing an opened queue. QueueContextHandles are acquired by opening a queue. See section 3.2.4.7 for information about opening a queue.

3.2.1.2 CursorContextValue

A **DWORD** value representing an opened cursor. A CursorContextValue MUST be associated with a QueueContextHandle, and is unique in the context of its associated QueueContextHandle. As such, a cursor created for a particular QueueContextHandle MUST NOT be used for operations against different QueueContextHandles. A CursorContextValue is acquired by creating a cursor. See section 3.2.4.8 for information about creating a cursor.

3.2.2 Timers

No protocol timers are required beyond those used internally by RPC to implement resiliency to network outages. For more information, see [MS-RPCE].

3.2.3 Initialization

The client MUST create an RPC connection to the remote computer, using the details specified in section 2.1.

3.2.4 Message Processing Events and Sequencing Rules

The operation of the protocol is initiated and subsequently driven by the following higher-layer triggered events:

- An MSMQ application creates a local private queue.
- An MSMQ application deletes a local private queue.
- An MSMQ application updates the security configuration of a local private queue.
- An MSMQ application retrieves the security configuration of a local private queue.
- An MSMQ application updates the properties of a local private queue.
- An MSMQ application retrieves the properties of a local private queue.
- An MSMQ application opens a queue.
- An MSMQ application creates a cursor.
- An MSMQ application purges a queue.
- An MSMQ application sends a message.
- An MSMQ application peeks a message.
- An MSMQ application receives a message.
- An MSMQ application sends or receives a message in the context of an external transaction.
- An MSMQ application sends or receives messages in the context of an internal transaction.
- An MSMQ application peeks a message using a cursor.
- An MSMQ application requests a format name for a queue path name.
- An MSMQ application requests a format name for a queue context handle.
- An MSMQ application closes a queue handle.

- An MSMQ application closes a cursor handle.

Prior to performing any operations over this protocol, the client MUST first construct an RPC binding handle to the server, as specified in [C706] section 2.3. The client MAY call the **R_QMGetRTQMServerPort** method using the RPC handle described above. This method returns an RPC port number with which subsequent method calls to this interface MAY be invoked. The client MAY construct a new RPC binding handle using the RPC port number acquired from **R_QMGetRTQMServerPort**, and use the new binding handle for subsequent method invocations.

3.2.4.1 Creating a Local Private Queue

The MSMQ application MUST supply a queue name, and MAY supply a SECURITY_DESCRIPTOR and queue properties for the new queue. Creating a new local private queue consists of the following operations:

- The client MUST call **R_QMCreateObjectInternal**, supplying the following parameter values:
  - dwObjectType MUST 0x00000001.
  - lpwcsPathName MUST contain the NULL-terminated queue name string.
  - An initial SECURITY_DESCRIPTOR MAY be specified for the new queue using the pSecurityDescriptor and SDSize parameters as specified in section 3.1.4.5.
  - Initial property values MAY be supplied for the new queue using the cp, aProp, and apVar parameters as specified in section 3.1.4.5. To not specify any initial property values for the new queue, yet meet the requirement of specifying at least 1 property value, the client MAY supply the queue property PROPID_Q_PATHNAME using the same value specified for lpwcsPathName. MSMQ queue property values are defined in [MS-MQMQ] section 2.3.1.
- To receive or send messages to the new queue, the client application MUST first open the queue, as specified in section 3.2.4.7. Opening a queue requires a format name, which is either constructed by the MSMQ application, or acquired from the server, as specified in section 3.2.4.13.

3.2.4.2 Deleting a Local Private Queue

The MSMQ application MUST supply a format name for the local private queue to be deleted.

- The given format name MUST be of the "private" or "direct" variety, as specified in [MS-MQMQ].
- The client MUST call **R_QMDeleteObject**, supplying the following parameter values:
  - A pointer to an **OBJECT_FORMAT** structure containing the format name of the queue to be deleted, as specified in section 3.1.4.8.

3.2.4.3 Updating Local Private Queue Security

The MSMQ application MUST supply a format name for a local private queue for which the security configuration is to be updated, a new SECURITY_DESCRIPTOR for the queue, and a SECURITY_DESCRIPTOR value indicating which portions of the SECURITY_DESCRIPTOR are to be applied to the queue. SECURITY_DESCRIPTOR is specified in [MS-DTYP] section **2.4.6** and SECURITY_INFORMATION in [MS-MQMQ] section 2.2.3.

- The given format name MUST be of the "private" or "direct" variety, as specified in [MS-MQMQ].
- The given SECURITY_DESCRIPTOR MUST be in self-relative form.

- The client MUST call **R_QMSetObjectSecurityInternal**, supplying the following parameter values:
  - A pointer to an **OBJECT_FORMAT** structure containing the format name of the queue, as specified in section 3.1.4.6.
  - The SecurityInformation, SDSize, and pSecurityDescriptor parameters MUST be supplied as specified in section 3.1.4.6.

3.2.4.4 Retrieving Local Private Queue Security

The MSMQ application MUST supply a format name for a local private queue, and a SECURITY_INFORMATION value indicating which portions of the security configuration to retrieve. The client MAY provide a buffer into which the server returns a SECURITY_DESCRIPTOR. SECURITY_DESCRIPTOR is specified in [MS-DTYP] section **2.4.6** and SECURITY_INFORMATION in [MS-MQMQ] section 2.2.3.

- The given format name MUST be of the "private" or "direct" variety, as specified in [MS-MQMQ] section 2.1.2.
- The client MUST call **R_QMGetObjectSecurityInternal**, specifying the following parameter values:
  - A pointer to an **OBJECT_FORMAT** structure containing the format name of the queue, as specified in section 3.1.4.7.
  - pSecurityDescriptor MAY point to an array of bytes, in which case nLength MUST specify the byte length of the buffer. Alternatively, pSecurityDescriptor MAY be NULL, in which case nLength MUST be NULL.
  - lpnLengthNeeded MUST point to a DWORD that receives the actual byte length of the requested SECURITY_DESCRIPTOR.
- If the server returns MQ_OK (0x00000000), the buffer pointed to by pSecurityDescriptor contains the requested SECURITY_DESCRIPTOR. The length of the SECURITY_DESCRIPTOR is pointed to by lpnLengthNeeded.
- If the server returns MQ_ERROR_SECURITY_DESCRIPTOR_TOO_SMALL (0xc00e0023), lpnLengthNeeded points to a DWORD containing the byte length required to contain the requested SECURITY_DESCRIPTOR. A subsequent call to **R_QMGetObjectSecurityInternal** using a buffer of the byte length indicated by lpnLengthNeeded MAY succeed.

3.2.4.5 Updating Local Private Queue Properties

The MSMQ application MUST supply a format name for a local private queue for which property values are to be updated, and one or more new queue property values for the indicated queue. MSMQ queue property values are defined in [MS-MQMQ] section 2.3.1.

- The given format name MUST be of the "private" or "direct" variety, as specified in [MS-MQMQ] section 2.1.2.
- The client MUST call **R_QMSetObjectProperties**, supplying the following parameter values:
  - A pointer to an **OBJECT_FORMAT** structure containing the format name of the queue, as specified in section 3.1.4.10.

- Updated property values for the queue are provided using the cp, aProp, and apVar parameters as described in section 3.1.4.10.

3.2.4.6 Retrieving Local Private Queue Properties

The MSMQ application MUST supply a format name for a local private queue from which to retrieve property values, and a set of property identifiers for which values are to be retrieved. Additionally, the client MUST provide a set of **PROPVARIANT**s into which the server will place the requested property values. MSMQ queue property values and the PROPVARIANT structure are defined in [MS-MQMQ] section 2.2.13.2.

- The given format name MUST be of the "private" or "direct" variety, as specified in [MS-MQMQ] section 2.1.2.
- The client MUST call **R_QMGetObjectProperties**, supplying the following parameter values:
  - A pointer to an **OBJECT_FORMAT** structure containing the format name of the queue, as specified in section 3.1.4.9.
  - cp MUST contain the number of properties to be retrieved.
  - aProp MUST contain an array of queue property identifiers requested by the MSMQ application. The array MUST contain cp elements.
  - apVar MUST contain an array of PROPVARIANT structures to be populated by the server. The array MUST contain cp elements.

3.2.4.7 Opening a Queue

The MSMQ application MUST supply a format name for a queue to be opened. Additionally, the application MUST indicate the desired access mode and exclusivity level for the queue.

- The client MUST call **rpc_QMOpenQueueInternal**, supplying the following parameter values:
  - pQueueFormat MUST contain the format name of the queue to be opened.
  - dwDesiredAccess and dwShareMode MUST contain the access mode and exclusivity level.
  - hRemoteQueue MUST be 0x00000000.
  - lplpRemoteQueueName MUST be NULL.
  - Other parameters as specified in section 3.1.4.17.
- If MQ_OK (0x00000000) is returned, and lplpRemoteQueueName is still NULL, phQueue contains a QueueContextHandle. The client MUST record this context handle for subsequent operations against the queue.
- If MQ_OK (0x00000000) is returned, and lplpRemoteQueueName is non-NULL, the client MUST ignore the returned context handle value and take the following steps:
  - The server sets lplpRemoteQueueName to a queue path for a remote queue. The queue path contains the name of a remote server.
  - Bind to the remote server using the RPC binding procedure as specified in 3.2.4.
  - At the remote server, call **R_QMOpenRemoteQueue** supplying the following parameter values:

- pQueueFormat, dwDesiredAccess, and dwShareMode MUST be the same as specified for the call to **rpc_QMOpenQueueInternal**.
- Other parameters as specified in section 3.1.4.2.
- Record the out-parameter values for pphContext and pdwContext.

- At the original server, invoke **rpc_QMOpenQueueInternal** once more, specifying the same parameter values as the first call, except:
  - hRemoteQueue MUST be set to the pphContext value returned from **R_QMOpenRemoteQueue**.
  - dwpRemoteContext MUST be set to the pdwContext value returned from **R_QMOpenRemoteQueue**.
  - The client MUST record hQueue as a QueueContextHandle for use in subsequent operations.
- Call **R_QMCloseRemoteQueueContext** (as specified in [MS-MQQP]) to close the context handle acquired by the call to **R_QMOpenRemoteQueue**.

3.2.4.8 Creating a Cursor

The MSMQ application MUST supply a QueueContextHandle for which the cursor is to be associated.

- The client MUST call **rpc_ACCreateCursorEx**, supplying the following parameter values:
  - hQueue MUST be a QueueContextHandle acquired from **rpc_QMOpenQueueInternal**.
  - pcc MUST point to a **CACCreateRemoteCursor** structure.
- If MQ_OK (0x00000000) is returned, the client MUST record the CursorContextValue returned in pcc.hCursor. This CursorContextValue is specified in subsequent queue operations.
- Else if: MQ_INFORMATION_REMOTE_OPERATION (0x400e03e8) is returned, the client MUST take the following steps:
  - Record pcc.hCursor as a new CursorContextValue. This CursorContextValue is specified in subsequent queue operations, after successfully completing the following steps. The CursorContextValue SHOULD be closed by the client, even if a failure is encountered in the following steps.
  - Call **R_QMGetRemoteQueueName** supplying the value returned by **rpc_ACCreateCursorEx** in pcc.cli_pQMQueue for the pQueue parameter. A path for the remote queue is returned in lpRemoteQueueName parameter. The remote queue path contains the name of the remote server.
  - Bind to the remote server using the RPC binding procedure specified in section 3.2.4.
  - At the remote server, call **R_QMCreateRemoteCursor** supplying the following parameter values:
    - ptb1 MUST be NULL.
    - hQueue MUST contain the value returned by **rpc_ACCreateCursorEx** in pcc.srv_hACQueue.

- phCursor MUST point to a DWORD to accept a value from the server.

- At the original server, call **rpc_ACSetCursorProperties** supplying the following parameter values:

  - hProxy MUST contain the same QueueContextHandle originally passed to **rpc_ACCreateCursorEx**.

  - hCursor MUST contain the CursorContextValue returned by **rpc_ACCreateCursorEx** in pcc.hCursor.

  - hRemoteCursor MUST contain the remote cursor context value returned in the phCursor parameter after the call to **R_QMCreateRemoteCursor**.

3.2.4.9 Purging a Queue

The MSMQ application MUST supply a QueueContextHandle indicating the queue to be purged.

- The client MUST call **rpc_ACPurgeQueue** supplying the following parameter values:

  - hQueue MUST contain the QueueContextHandle of the queue to be purged. The QueueContextHandle MUST be acquired from **rpc_QMOpenQueueInternal**.

3.2.4.10 Sending a Message

The MSMQ application MUST supply a QueueCon QueueContextHandle textHandle indicating the queue to which the message is to be sent. The QueueContextHandle MUST be obtained from **rpc_QMOpenQueueInternal** specifying the value MQ_SEND_ACCESS (0x00000002) for dwDesiredAccess.

To perform the send operation in the context of a transaction, the client first must enlist the transaction by calling R_QMEnlistTransaction or R_QMEnlistInternalTransaction, as described in sections 3.1.4.13 and 3.1.4.14.

- The client MUST call **rpc_ACSendMessageEx** supplying the following parameter values:

  - hQueue MUST contain the QueueContextHandle for the target queue.

  - To receive the ID of the sent message, a pointer to a 20-byte buffer MUST be provided for the *pMessageID* parameter. Otherwise, pMessageID MUST be NULL.

  - A pointer to a **CACTransferBufferV2** structure MUST be passed to the ptb parameter. This structure contains the properties of the message to be sent.

    - ptb.old.uTransferType MUST be CACTB_SEND (0x00000000).

    - If the send operation is occurring in the context of a transaction, the **XACTUOW**transaction identifier must be provided in the ptb.old.pUow field. Otherwise, ptb.old.pUow MUST be NULL.

    - The remainder of the ptb2 member fields MUST be populated according to the limitations and definitions in section 3.1.5.2.

3.2.4.11 Peeking a Message

The MSMQ application MUST supply a QueueContextHandle indicating the queue from which to attempt to peek a message. The QueueContextHandle MUST be obtained from

rpc_QMOpenQueueInternal specifying the value MQ_PEEK_ACCESS (0x00000020) for dwDesiredAccess.

- The client MUST call **rpc_ACReceiveMessageEx** supplying the following parameter values:
  - The hQMContext parameter MUST contain a queue context value obtained from the pdwQMContext parameter of **rpc_QMOpenQueueInternal**.
  - A pointer to a **CACTransferBufferV2** structure MUST be passed to the ptb parameter. This structure will receive values of the retrieved message.
    - ptb.old.uTransferType MUST be CACTB_RECEIVE (0x00000001).
    - ptb.old.Receive.Action MUST be MQ_ACTION_PEEK_CURRENT (0x80000000). MQ_ACTION_PEEK_NEXT (0x80000001) MAY be acceptable if a cursor is provided for the ptb.old.Receive.Cursor field. The behavior of these fields is described in section 3.1.5.3.
    - The remainder of the ptb member fields MUST be populated according to the limitations and definitions in section 3.1.5.3.

3.2.4.12 Receiving a Message

The MSMQ application MUST supply a QueueContextHandle indicating the queue from which to attempt to receive a message. The QueueContextHandle MUST be obtained from **rpc_QMOpenQueueInternal** specifying the value MQ_RECEIVE_ACCESS (0x00000001) for dwDesiredAccess.

- The client MUST call **rpc_ACReceiveMessageEx** supplying the following parameter values:
  - The hQMContext parameter MUST contain a queue context value obtained from the pdwQMContext parameter of rpc_QMOpenQueueInternal.
  - A pointer to a **CACTransferBufferV2** structure MUST be passed to the ptb parameter. This structure will receive values of the retrieved message.
    - ptb.old.uTransferType MUST be CACTB_RECEIVE (0x00000001).
    - ptb.old.Receive.Action MUST be MQ_ACTION_RECEIVE (0x00000000).
    - The remainder of the ptb member fields MUST be populated according to the limitations and definitions in section 3.1.5.3.

3.2.4.13 Retrieving a Format Name for a Queue Path Name

The MSMQ application MUST supply a queue path for which a format name is to be retrieved by the server.

- The client MUST call **R_QMObjectPathToObjectFormat**, supplying the following parameter values:
  - lpwcsPathName MUST contain a path name.
  - pObjectFormat MUST point to an QUEUE_FORMAT structure to be populated by the server. The QUEUE_FORMAT structure is specified in [MS-MQMQ] section 2.2.7.

3.2.4.14 Retrieving a Format Name for a Queue Context Handle

The MSMQ application MUST supply a QueueContextHandle for which a format name is to be retrieved, as well as a buffer into which the format name string is to be placed.

- The client MUST call **rpc_ACHandleToFormatName**, supplying the following parameter values:
  - hQueue MUST contain a QueueContextHandle.
  - lpwcsFormatName MUST point to Unicode character array into which the server will copy a format name. Alternatively, lpwcsFormatName MAY be NULL.
  - dwFormatNameRPCBufferLen and pdwLength MUST contain the length of the lpwcsFormatName buffer, in Unicode characters. If lpwcsFormatName is NULL, dwFormatNameRPCBufferLen and pdwLength MUST be 0x00000000.
  - If MQ_OK (0x00000000) is returned, lpwcsFormatName contains a NULL-terminated format name for the given queue name.
  - Else, if MQ_ERROR_FORMATNAME_BUFFER_TOO_SMALL (0xc00e001f) is returned, the client SHOULD ignore lpwcsFormatName and call the method again with a larger (or non-NULL) buffer. pdwLength contains the length of the buffer necessary to contain the entire format name including the NULL-terminator, in Unicode characters.

3.2.4.15 Closing a Queue Context Handle

The MSMQ application MUST supply a QueueContextHandle to be closed. The client MUST call **rpc_ACCloseHandle**, specifying the QueueContextHandle.

3.2.4.16 Closing a Cursor

The MSMQ application MUST supply a CursorContextValue to be closed, as well as the QueueContextHandle supplied when the cursor was created. The client MUST call **rpc_ACCloseCursor**, specifying the QueueContextHandle and CursorContextValue.

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 sections describe several operations as used in common scenarios to illustrate the function of the Message Queuing (MSMQ): Queue Manager Client Protocol.

4.1 Dependent Client Opening and Closing a Local Queue Example

The following sequence diagram illustrates a dependent client interacting with a supporting server to open a queue handle for a queue located at the supporting server.

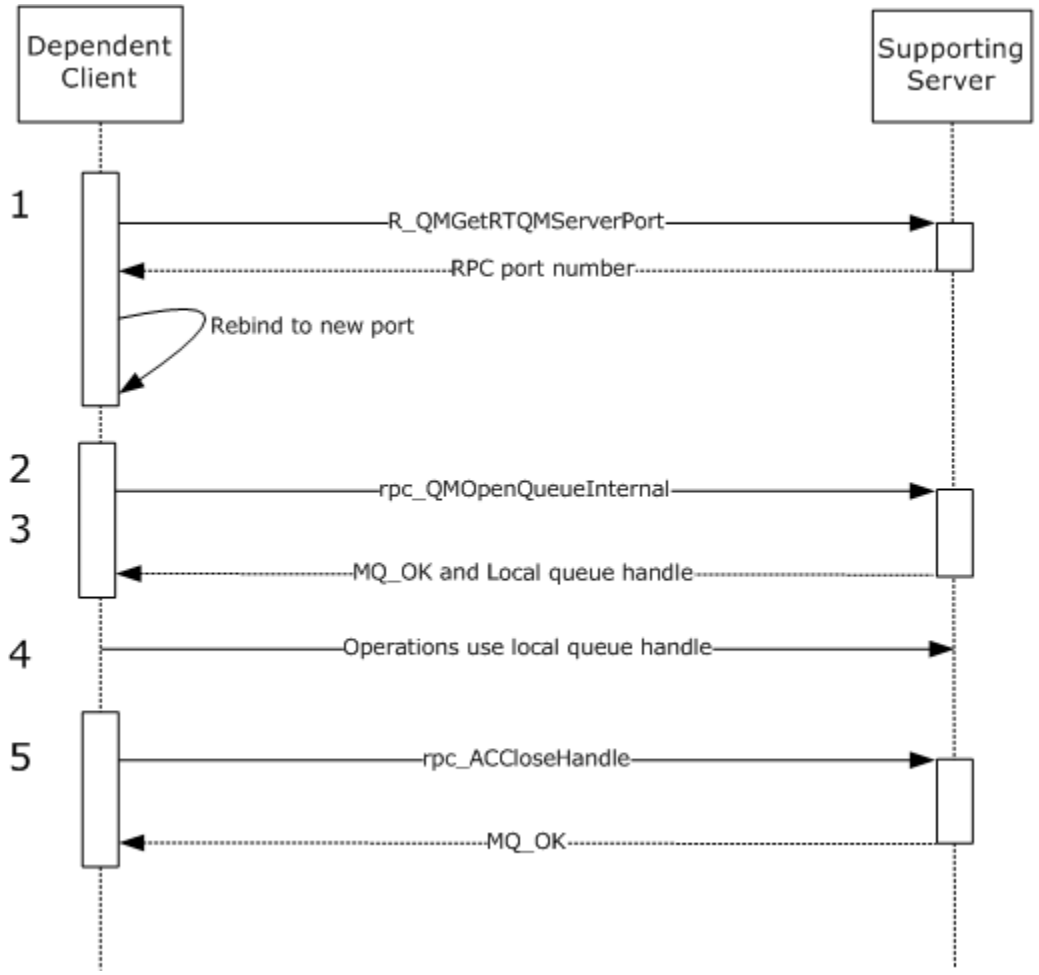

Figure 3: Queue opening

1. Dependent client begins the RPC session by invoking **R_QMGetRTQMServerPort** to query the RPC port number for subsequent method invocations.

2. Dependent client invokes **rpc_QMOpenQueueInternal**, specifying a format name identifying the queue to open.

3. Supporting server determines that the queue identified by the format name is located locally. A local queue handle is returned. lplpRemoteQueueName is NULL, to indicate that a remote queue open sequence (demonstrated in section 4.2) is not necessary.

4. Dependent client performs operations utilizing the local queue handle, such as send, receive, peek, or purge.

5. Dependent client closes the local queue handle when it is no longer required.

4.2 Dependent Client Opening and Closing a Remote Queue Example

The following sequence diagram illustrates a dependent client interacting with a supporting server to create a queue handle for a queue located at a remote queue manager.

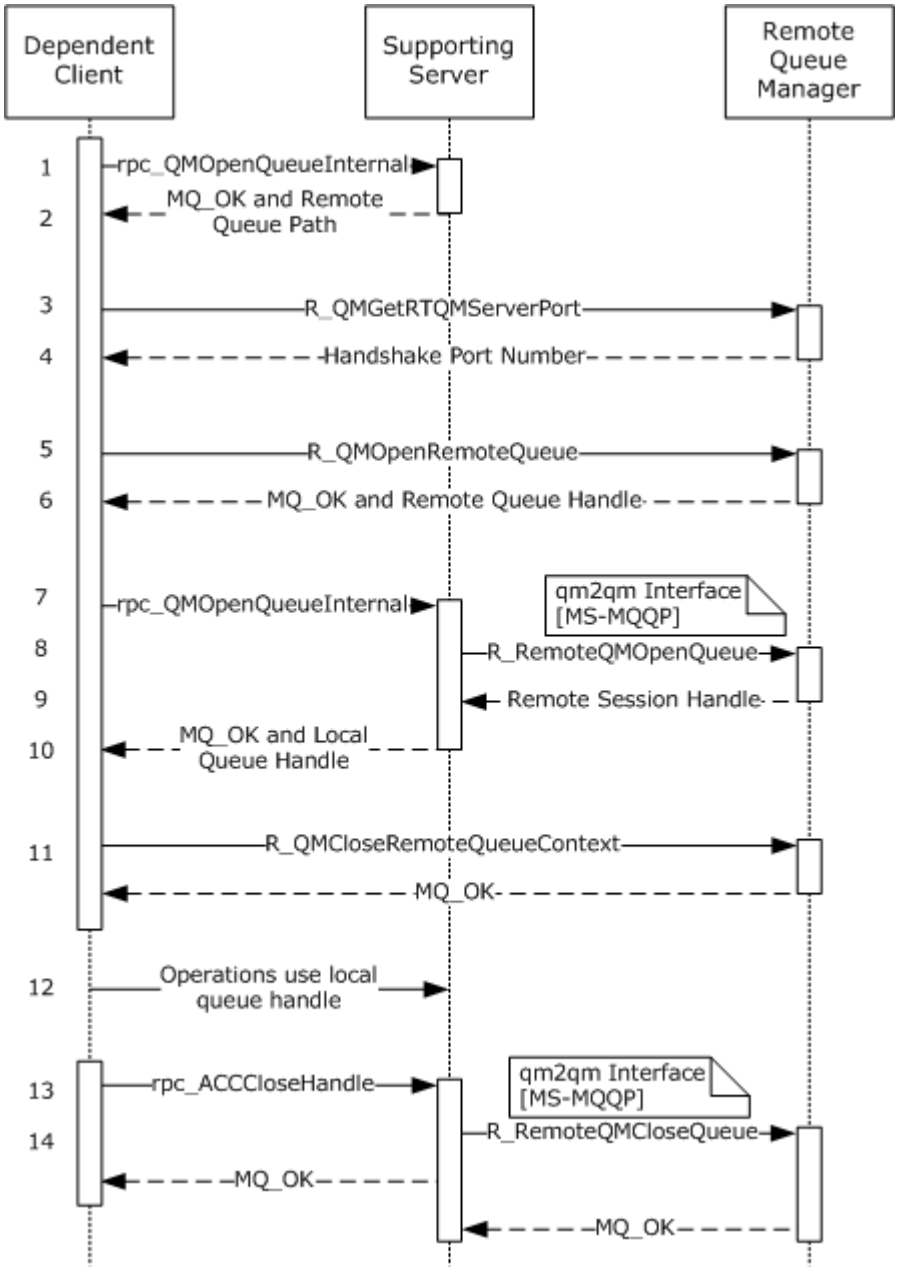

Figure 4: Creating a queue handle

1. Dependent client invokes **rpc_QMOpenQueueInternal**, providing a format name for a queue to open. NULL is specified for hRemoteQueue.

2. Supporting server determines that the queue identified by the format name is a remote queue. A path name for the remote queue is returned via lplpRemoteQueueName. All returned handles are NULL.

3. Dependent client uses the path name returned by **rpc_QMOpenQueueInternal** to determine the computer name of the remote queue manager, as specified in [MS-MQMQ] section 2.1.1. The dependent client then establishes an RPC connection with the remote queue manager and begins the session by invoking **R_QMGetRTQMServerPort**.

4. Remote queue manager returns the RPC port number requested by the dependent client.

5. Dependent client invokes **R_QMOpenRemoteQueue** at the remote queue manager, using the RPC port returned by **R_QMGetRTQMServerPort** and specifying the format name of the queue to be opened.

6. Remote queue manager verifies that the client has permission to open the queue for the requested operations. If the client has the necessary permission, a queue handle for the queue is returned.

7. Dependent client invokes **rpc_QMOpenQueueInternal** on the supporting server once again. For this invocation, the client provides the remote queue handle (and remote queue context value) returned from **R_QMOpenRemoteQueue**.

8. Supporting server binds to the remote queue manager and utilizes the qm2qm RPC protocol, as defined by [MS-MQQP], to create a remote-read session. The client passes the queue handle returned at step 6, which the server uses as validation that the client has permission to open the queue.

9. The qm2qm protocol exchange between the supporting server and the remote queue manager produces a remote-read session handle, as specified in [MS-MQQP].

10. Supporting server internally associates the qm2qm session handle with a new local queue handle, and returns the local queue handle to dependent client.

11. Dependent client, having successfully acquired a local queue handle, no longer requires the remote queue handle, and closes it via **R_QMCloseRemoteQueueContext**.

12. Dependent client utilizes the local queue handle to execute remote-read message operations via the qmcomm2 interface. Supporting server uses the remote-read session handle to contact the remote queue manager as necessary to carry out the message operations. This process is defined by [MS-MQQP].

13. Dependent client is finished utilizing the local queue handle, and closes it with a call to **rpc_ACCloseHandle**.

14. Supporting server closes the remote-read session handle (via the qm2qm protocol, as specified in [MS-MQQP]) which was associated with the local queue handle passed by dependent client at step 13. Note that supporting server invokes the qm2qm protocol in parallel, and does not block **rpc_ACCloseHandle** while the session handle is being closed.

4.3 Dependent Client Closing a Local Cursor Example

The following sequence diagram illustrates a dependent client interacting with a supporting server to create and close a cursor handle for a queue located at the supporting server.

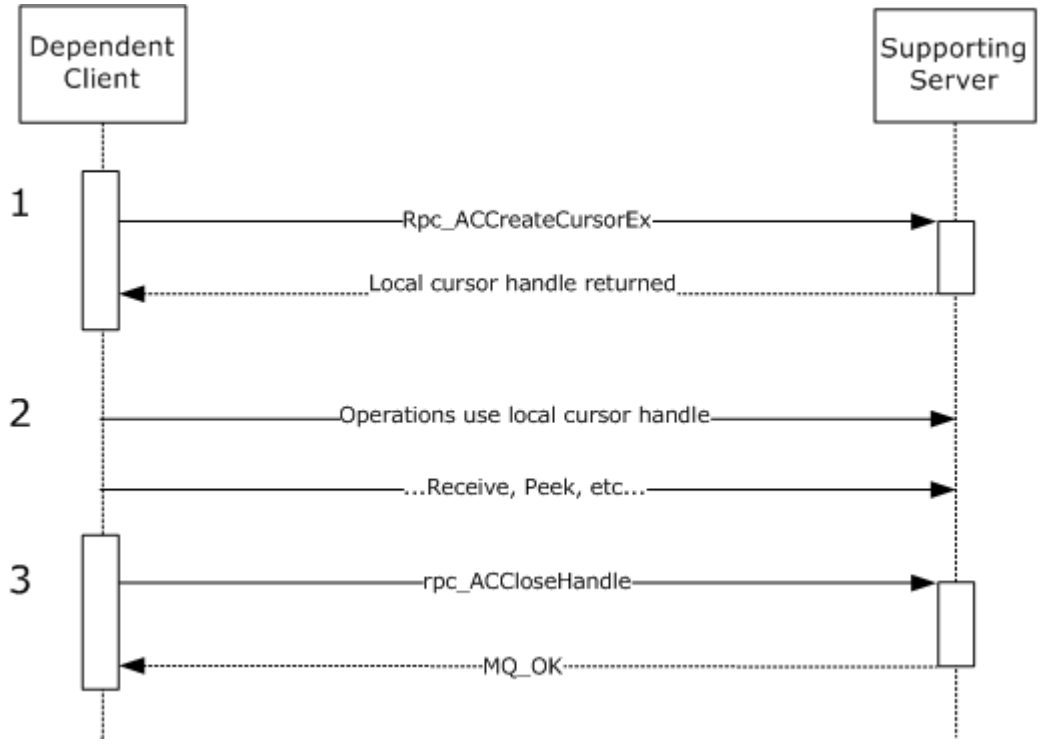

Figure 5: Creating and closing a local cursor

1. Dependent client creates a local cursor using the qmcomm2 interface.
2. Dependent client utilizes the local cursor handle value to perform messaging operations.
3. Dependent client closes the cursor handle via the **rpc_ACCloseCursor** method.

4.4 Dependent Client Creating and Closing a Remote Cursor Example

The following sequence diagram illustrates a dependent client interacting with a supporting server to create a cursor handle for a queue located at a remote queue manager.

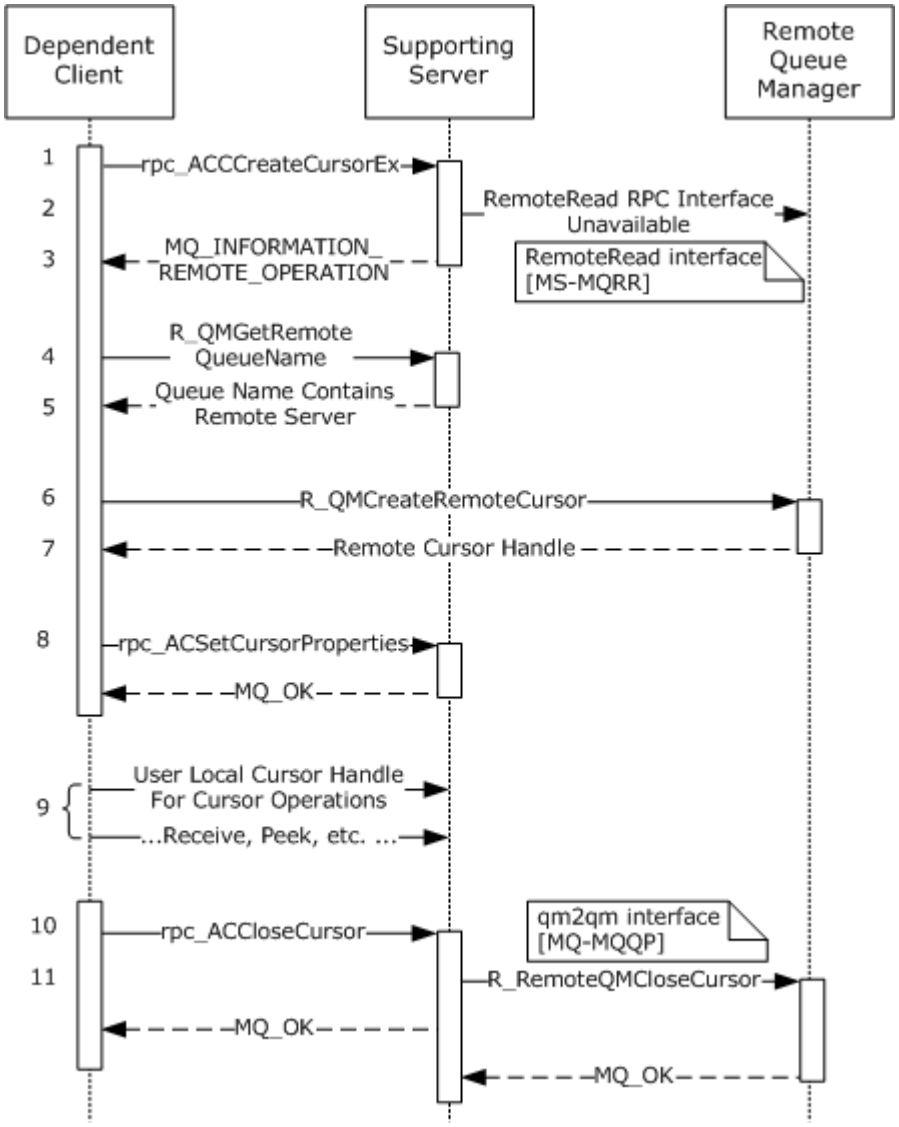

Figure 6: Creating and closing a remote cursor

1. Dependent client begins the process of creating a cursor by calling the **rpc_ACCreateCursorEx** (opnum 3) method of the qmcomm2 interface. A queue handle is specified which identifies the queue with which to associate the cursor.

2. Supporting server determines that the queue identified by the queue handle provided in step 1 is located at a remote queue manager. Supporting server MAY attempt to contact the remote queue manager via the RemoteRead protocol.<84> For this example, the remote queue manager does not support the RemoteRead interface, as specified in [MS-MQRR].

3. Supporting server, having determined that the queue is located remotely, and that it cannot create the remote cursor handle itself via the RemoteRead protocol, returns the following information to the client:

    1. A new local cursor handle value.

2. The local queue context value associated with the local queue context handle provided at step 1 (this value was determined when the queue was opened).

3. The remote queue context handle associated with the local queue context handle provided at step 1 (this handle was created when the queue was opened).

4. A special return code: MQ_INFORMATION_REMOTE_OPERATION.

4. Dependent client detects the special return code MQ_INFORMATION_REMOTE_OPERATION, which indicates that the cursor must be created at a remote queue manager. In order to bind to the remote queue manager, the dependent client must determine the computer name of the remote queue manager. The dependent client invokes **R_QMGetRemoteQueueName** using the local queue context value described at step 3b.

5. Supporting server returns the path name for the remote queue.

6. Dependent client determines the remote queue manager computer name using the path name returned at step 5. An RPC binding is established, and port number queried (via **R_QMGetRTQMServerPort**). The dependent client then invokes **R_QMCreateRemoteCursor** at the remote queue manager, specifying the remote queue context handle returned at step 3c.

7. Remote queue manager creates and returns a remote cursor handle value to the dependent client.

8. Dependent client invokes **rpc_ACSetCursorProperties**, specifying the original queue context handle from step 1, the local cursor handle from step 3a, and the remote cursor handle from step 7. The supporting server associates these values for future reference.

9. Dependent client may now utilize the local cursor handle value (returned at step 3a) to perform messaging operations via the qmcomm2 interface. The supporting server delegates the operations to the remote queue manager via the qm2qm interface, as specified in [MS-MQQP].

10. Dependent client closes the local cursor handle value.

11. Supporting server closes the remote cursor handle value created at step 7 via the qm2qm interface, as specified in [MS-MQQP]. Note that this is performed in parallel, and need not block the return from **rpc_ACCloseCursor**.

4.5 Dependent Client Internal Transaction Example

The following sequence diagram illustrates a dependent client interacting with a supporting server to enlist the supporting server's resource manager in an internal transaction, perform operations in the scope of the internal transaction, and finally commit the transaction.

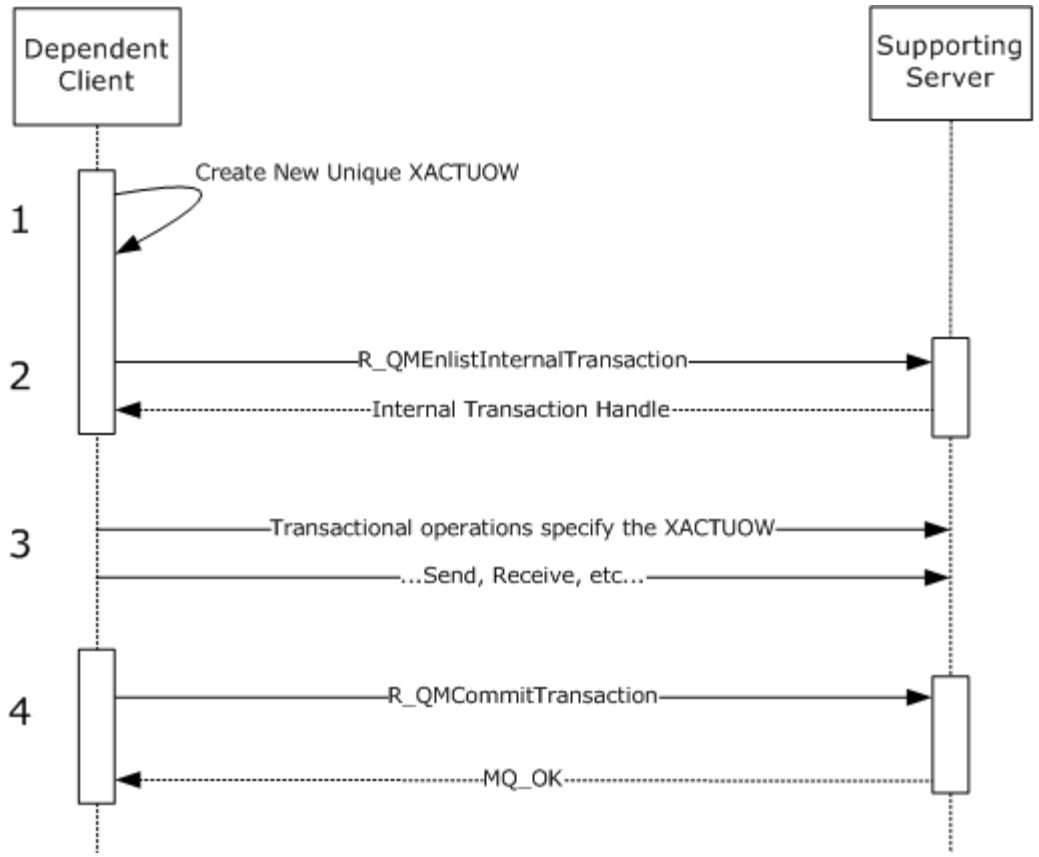

Figure 7: Using server resource manager for internal transaction

1. Prior to invoking **R_QMEnlistInternalTransaction**, the dependent client MUST create a new unique transactional unit of work identifier (**XACTUOW**).<85>

2. Dependent client invokes **R_QMEnlistInternalTransaction** to create an internal transaction handle for the **XACTUOW**.

3. Dependent client utilizes the **XACTUOW** identifier created at step 1 to perform operations in the scope of the transaction via the qmcomm2 interface.

4. Dependent client finally commits the transaction by calling **R_QMCommitTransaction**, specifying the internal transaction handle obtained at step 2.

5 Security

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

5.1 Security Considerations for Implementers

Clients MAY invoke methods of this interface at the "none" authentication level as defined by [MS-RPCE].<86> Server implementations SHOULD be designed with careful consideration given to the security implications of accepting method calls from unauthenticated clients. Server implementations SHOULD reject methods invoked by unauthenticated clients by returning RPC_S_ACCESS_DENIED (0x00000005).

The **R_QMGetRTQMServerPort** method is an exception to the above consideration, since clients MAY invoke **R_QMGetRTQMServerPort** prior to configuring security for the RPC binding. For this reason, server implementations MUST NOT restrict access to the **R_QMGetRTQMServerPort** method.

5.2 Index of Security Parameters

No security parameters are defined for this protocol.

6 Appendix A: Full IDL

For ease of implementation, the full IDL is provided below, where "ms-dtyp.idl" is the IDL found in [MS-DTYP] Appendix A and "ms-mqmq.idl" is the IDL found in [MS-MQMQ] Appendix A.

```
// Please refer to [MS-MQMQ] for definitions of the
// following types:
//   PROPVARIANT
//   MULTICAST_ID
//   OBJECTID
import "ms-mqmq.idl";

// Please refer to [MS-MQRR] for definitions of the
// following types:
//   QUEUE_FORMAT
//   DL_ID
import "ms-mqrr.idl";

// Please refer to [MS-DTYP] for definitions of the
// following types:
//   DWORD
//   GUID

typedef struct _XACTUOW
{
    unsigned char rgb[ 16 ];
} XACTUOW;

[
  uuid(fdb3a030-065f-11d1-bb9b-00a024ea5525),
  version(1.0),
  pointer_default(unique)
]
interface qmcomm
{

struct CACCreateRemoteCursor {
      DWORD hCursor;
      DWORD srv_hACQueue;
      DWORD cli_pQMQueue;
};
enum TRANSFER_TYPE {
    CACTB_SEND = 0,
    CACTB_RECEIVE,
    CACTB_CREATECURSOR,
};

struct CACTransferBufferV1 {
    [range(0,2)] DWORD uTransferType;
    [switch_is(uTransferType)]
    union {
        [case(CACTB_SEND)]
        struct {
            struct QUEUE_FORMAT* pAdminQueueFormat;
            struct QUEUE_FORMAT* pResponseQueueFormat;
        } Send;
```

```
        [case(CACTB_RECEIVE)]
        struct {
            DWORD RequestTimeout;
            DWORD Action;
            DWORD Asynchronous;
            DWORD Cursor;
            [range(0,1024)] DWORD   ulResponseFormatNameLen ;
            [size_is(,ulResponseFormatNameLen)]
              WCHAR** ppResponseFormatName;
            DWORD*  pulResponseFormatNameLenProp;
            [range(0,1024)] DWORD   ulAdminFormatNameLen ;
            [size_is(,ulAdminFormatNameLen)]
              WCHAR** ppAdminFormatName;
            DWORD*  pulAdminFormatNameLenProp;
            [range(0,1024)] DWORD   ulDestFormatNameLen;
            [size_is(,ulDestFormatNameLen)]
              WCHAR** ppDestFormatName;
            DWORD*  pulDestFormatNameLenProp;
            [range(0,1024)] DWORD   ulOrderingFormatNameLen;
            [size_is(,ulOrderingFormatNameLen)]
              WCHAR** ppOrderingFormatName;
            DWORD*  pulOrderingFormatNameLenProp;
        } Receive;
        [case(CACTB_CREATECURSOR)]
        struct CACCreateRemoteCursor CreateCursor;
    };
    unsigned short* pClass;
    OBJECTID** ppMessageID;
    [size_is(,20), length_is(,20)]
    unsigned char** ppCorrelationID;
    DWORD* pSentTime;
    DWORD* pArrivedTime;
    unsigned char* pPriority;
    unsigned char* pDelivery;
    unsigned char* pAcknowledge;
    unsigned char* pAuditing;
    DWORD* pApplicationTag;
    [size_is(,ulAllocBodyBufferInBytes),
     length_is(,ulBodyBufferSizeInBytes)]
    unsigned char** ppBody;
    DWORD ulBodyBufferSizeInBytes;
    DWORD ulAllocBodyBufferInBytes;
    DWORD* pBodySize;
    [size_is(,ulTitleBufferSizeInWCHARs),
     length_is(,ulTitleBufferSizeInWCHARs)]
    WCHAR** ppTitle;
    DWORD   ulTitleBufferSizeInWCHARs;
    DWORD*  pulTitleBufferSizeInWCHARs;
    DWORD ulAbsoluteTimeToQueue;
    DWORD* pulRelativeTimeToQueue;
    DWORD ulRelativeTimeToLive;
    DWORD* pulRelativeTimeToLive;
    unsigned char* pTrace;
    DWORD* pulSenderIDType;
    [size_is(,uSenderIDLen)]
    unsigned char** ppSenderID;
    DWORD* pulSenderIDLenProp;
    DWORD* pulPrivLevel;
```

```
    DWORD  ulAuthLevel;
    unsigned char* pAuthenticated;
    DWORD* pulHashAlg;
    DWORD* pulEncryptAlg;
    [size_is(,ulSenderCertLen)]
    unsigned char** ppSenderCert;
    DWORD ulSenderCertLen;
    DWORD* pulSenderCertLenProp;
    [size_is(,ulProvNameLen)] WCHAR** ppwcsProvName;
    DWORD   ulProvNameLen;
    DWORD*  pulAuthProvNameLenProp;
    DWORD*  pulProvType;
    long    fDefaultProvider;
    [size_is(,ulSymmKeysSize)] unsigned char** ppSymmKeys;
    DWORD   ulSymmKeysSize;
    DWORD*  pulSymmKeysSizeProp;
    unsigned char bEncrypted;
    unsigned char bAuthenticated;
    unsigned short uSenderIDLen;
    [size_is(,ulSignatureSize)] unsigned char** ppSignature;
    DWORD   ulSignatureSize;
    DWORD*  pulSignatureSizeProp;
    GUID** ppSrcQMID;
    XACTUOW* pUow;
    [size_is(,ulMsgExtensionBufferInBytes),
     length_is(,ulMsgExtensionBufferInBytes)]
    unsigned char** ppMsgExtension;
    DWORD ulMsgExtensionBufferInBytes;
    DWORD* pMsgExtensionSize;
    GUID** ppConnectorType;
    DWORD* pulBodyType;
    DWORD* pulVersion;
}; //  CACTransferBufferV1

struct CACTransferBufferV2 {
    struct CACTransferBufferV1 old;
    unsigned char * pbFirstInXact;
    unsigned char * pbLastInXact;
    OBJECTID** ppXactID;
}; // CACTransferBufferV2

struct OBJECT_FORMAT {
    [range(1,2)] DWORD ObjType;
        [switch_is(ObjType)] union
        {
            [case(1)]
                struct QUEUE_FORMAT* pQueueFormat;
        };
};

typedef [context_handle] void* PCTX_OPENREMOTE_HANDLE_TYPE;
typedef [context_handle] void* RPC_QUEUE_HANDLE;
typedef [context_handle] void* RPC_INT_XACT_HANDLE;

// opnum 0
void
Opnum0NotUsedOnWire (void);
```

```
// opnum 1
HRESULT
R_QMGetRemoteQueueName(
    [in]  handle_t                hBind,
    [in]  DWORD                   pQueue,
    [in, out, ptr, string]        WCHAR** lplpRemoteQueueName
    );

// opnum 2
HRESULT
R_QMOpenRemoteQueue(
    [in] handle_t                       hBind,
    [out] PCTX_OPENREMOTE_HANDLE_TYPE  *pphContext,
    [out] DWORD                        *pdwContext,
    [in, unique] struct QUEUE_FORMAT   *pQueueFormat,
    [in]  DWORD                         dwCallingProcessID,
    [in]  DWORD                         dwDesiredAccess,
    [in]  DWORD                         dwShareMode,
    [in]  GUID*                         pLicGuid,
    [in]  DWORD                         dwMQS,
    [out] DWORD                         *dwpQueue,
    [out] DWORD                         *phQueue
    );

// opnum 3
void
R_QMCloseRemoteQueueContext(
    [in, out] PCTX_OPENREMOTE_HANDLE_TYPE  *pphContext
    );

// opnum 4
HRESULT
R_QMCreateRemoteCursor(
    [in]  handle_t                      hBind,
    [in]  struct CACTransferBufferV1 * ptb1,
    [in]  DWORD                         hQueue,
    [out] DWORD *                       phCursor
    );

// opnum 5
void
Opnum5NotUsedOnWire (void);

// opnum 6
HRESULT
R_QMCreateObjectInternal(
    [in] handle_t                   hBind,
    [in] DWORD                      dwObjectType,
    [in, string] const WCHAR*       lpwcsPathName,
    [in, range(0, 524288)] DWORD    SDSize,
    [in, unique, size_is (SDSize)]
        unsigned char               *pSecurityDescriptor,
    [in, range(1, 128)] DWORD       cp,
    [in, size_is (cp)]  DWORD       aProp[],
    [in, size_is (cp)]  PROPVARIANT apVar[]
    );

// opnum 7
```

```
HRESULT
R_QMSetObjectSecurityInternal(
    [in] handle_t                   hBind,
    [in] struct OBJECT_FORMAT*      pObjectFormat,
    [in] DWORD                      SecurityInformation,
    [in, range(0, 524288)] DWORD    SDSize,
    [in, unique, size_is (SDSize)]
        unsigned char               *pSecurityDescriptor);


// opnum 8
HRESULT
R_QMGetObjectSecurityInternal(
    [in] handle_t                           hBind,
    [in] struct OBJECT_FORMAT*              pObjectFormat,
    [in] DWORD                              RequestedInformation,
    [out, size_is (nLength)]  unsigned char *pSecurityDescriptor,
    [in, range(0, 524288)] DWORD            nLength,
    [out]  DWORD*                           lpnLengthNeeded
    );

// opnum 9
HRESULT
R_QMDeleteObject(
     [in] handle_t hBind,
     [in] struct OBJECT_FORMAT*  pObjectFormat
     );

// opnum 10
HRESULT
R_QMGetObjectProperties(
    [in] handle_t                       hBind,
    [in] struct OBJECT_FORMAT*          pObjectFormat,
    [in, range(1, 128)] DWORD           cp,
    [in, size_is (cp )]  DWORD          aProp[],
    [in, out, size_is(cp )]  PROPVARIANT apVar[]
    );


// opnum 11
HRESULT
R_QMSetObjectProperties(
    [in] handle_t                           hBind,
    [in] struct OBJECT_FORMAT*              pObjectFormat,
    [in, range(1, 128)] DWORD               cp,
    [in, unique, size_is (cp )]  DWORD      aProp[],
    [in, unique, size_is(cp )]  PROPVARIANT  apVar[]
    );

// opnum 12
HRESULT
R_QMObjectPathToObjectFormat(
    [in] handle_t hBind,
    [in, string] const WCHAR* lpwcsPathName,
    [in, out] struct OBJECT_FORMAT *pObjectFormat
    );

// opnum 13
```

```
void
Opnum13NotUsedOnWire (void);

// opnum 14
HRESULT
R_QMGetTmWhereabouts(
    [in]  handle_t                      hBind,
    [in, range(0, 131072)]  DWORD       cbBufSize,
    [out, size_is (cbBufSize)]          unsigned char* pbWhereabouts,
    [out] DWORD                         *pcbWhereabouts
    );

// opnum 15
HRESULT
R_QMEnlistTransaction(
    [in] handle_t                       hBind,
    [in] XACTUOW*                       pUow,
    [in, range(0, 131072)] DWORD        cbCookie,
    [in, size_is (cbCookie)]  unsigned char*    pbCookie
    );

// opnum 16
HRESULT
R_QMEnlistInternalTransaction(
    [in]  handle_t hBind,
    [in]  XACTUOW* pUow,
    [out] RPC_INT_XACT_HANDLE* phIntXact
    );

// opnum 17
HRESULT
R_QMCommitTransaction(
    [in, out] RPC_INT_XACT_HANDLE* phIntXact
    );

// opnum 18
HRESULT
R_QMAbortTransaction(
    [in, out] RPC_INT_XACT_HANDLE* phIntXact
    );

// opnum 19
HRESULT
rpc_QMOpenQueueInternal(
    [in]  handle_t      hBind,
    [in]  struct QUEUE_FORMAT* pQueueFormat,
    [in]  DWORD         dwDesiredAccess,
    [in]  DWORD         dwShareMode,
    [in]  DWORD         hRemoteQueue,
    [in, out, ptr, string] WCHAR**  lplpRemoteQueueName,
    [in]  DWORD*            dwpQueue,
    [in]  GUID*             pLicGuid,
    [in, string] WCHAR*     lpClientName,
    [out] DWORD*            pdwQMContext,
    [out] RPC_QUEUE_HANDLE* phQueue,
    [in]  DWORD             dwRemoteProtocol,
    [in]  DWORD             dwpRemoteContext
    );
```

```
// opnum 20
HRESULT
rpc_ACCloseHandle(
    [in, out] RPC_QUEUE_HANDLE* phQueue
    );

// opnum 21
void
Opnum21NotUsedOnWire (void);

// opnum 22
HRESULT
rpc_ACCloseCursor(
    [in] RPC_QUEUE_HANDLE hQueue,
    [in] DWORD hCursor
    );

// opnum 23
HRESULT
rpc_ACSetCursorProperties(
    [in] RPC_QUEUE_HANDLE hProxy,
    [in] DWORD hCursor,
    [in] DWORD hRemoteCursor
    );


// opnum 24
void
Opnum24NotUsedOnWire (void);

// opnum 25
void
Opnum25NotUsedOnWire(void);


// opnum 26
HRESULT
rpc_ACHandleToFormatName(
    [in] RPC_QUEUE_HANDLE hQueue,
    [in, range(0, 524288)] DWORD dwFormatNameRPCBufferLen,
    [in, out, unique,
     size_is(dwFormatNameRPCBufferLen),
     length_is(dwFormatNameRPCBufferLen)] WCHAR* lpwcsFormatName,
    [in, out] DWORD* pdwLength
    );


// opnum 27
HRESULT
rpc_ACPurgeQueue(
    [in] RPC_QUEUE_HANDLE hQueue
    );

// opnum 28
HRESULT
R_QMQueryQMRegistryInternal(
    [in]  handle_t         hBind,
```

```
    [in]  DWORD            dwQueryType,
    [out, string] WCHAR**  lplpMQISServer
    );

// opnum 29
void
Opnum29NotUsedOnWire (void);

// opnum 30
void
Opnum30NotUsedOnWire (void);

// opnum 31
DWORD
R_QMGetRTQMServerPort(
      [in] handle_t hBind,
      [in] DWORD fIP
);

// opnum 32
void
Opnum32NotUsedOnWire (void);

// opnum 33
void
Opnum33NotUsedOnWire (void);

// opnum 34
void
Opnum34NotUsedOnWire(void);

}  // interface qmcomm

[
  uuid(76d12b80-3467-11d3-91ff-0090272f9ea3),
  version(1.0),
  pointer_default(unique)
]
interface qmcomm2
{

// opnum 0
HRESULT
QMSendMessageInternalEx(
    [in] handle_t                    hBind,
    [in] struct QUEUE_FORMAT* pQueueFormat,
    [in] struct CACTransferBufferV2 * ptb,
    [in, out, unique] OBJECTID *     pMessageID
    );


// opnum 1
HRESULT
rpc_ACSendMessageEx(
    [in] RPC_QUEUE_HANDLE            hQueue,
    [in] struct CACTransferBufferV2 * ptb,
    [in, out, unique] OBJECTID *     pMessageID
    );
```

```
// opnum 2
HRESULT
rpc_ACReceiveMessageEx(
    [in] handle_t                           hBind,
    [in] DWORD                              hQMContext,
    [in, out] struct CACTransferBufferV2 * ptb
    );

// opnum 3
HRESULT
rpc_ACCreateCursorEx(
    [in] RPC_QUEUE_HANDLE                     hQueue,
    [in, out] struct CACCreateRemoteCursor * pcc
    );

}  // interface qmcomm2
```

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 2.1: The ncacn_spx protocol sequence is only supported by Windows NT and Windows 2000, and MAY only be supported if TCP/IP is unavailable. Support for IPX and the ncacn_spx protocol sequence is deprecated on Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008. The ncacn_ip_tcp protocol sequence SHOULD be supported when TCP/IP is available.

<2> Section 2.1: The ncacn_spx protocol sequence is only supported by Windows NT and Windows 2000, and MAY only be supported if TCP/IP is unavailable. Support for IPX and the ncacn_spx protocol sequence is deprecated on Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008. The ncacn_ip_tcp protocol sequence SHOULD be supported when TCP/IP is available.

<3> Section 2.2.2.1: All Windows clients produce new **XACTUOW** values by calling the Windows RPC function UuidCreate.

<4> Section 2.2.2.2: The only value supported by Windows is PROV_RSA_FULL (0x00000001).

<5> Section 2.2.2.2: With MSMQ version 2 and higher the **ppSignature** member contains an MSMQ version 1 signature followed by an MSMQ version 2 signature.

<6> Section 2.2.2.2: The only value supported by Windows is FALCON_PACKET_VERSION (0x00000010).

<7> Section 3.1.1.3: All Windows Servers return separate unique values for the **RPC_QUEUE_HANDLE** and the Queue Context Value, but this is not required for interoperability. An implementation of this protocol is permitted to return identical values for each parameter without compromising interoperability with Windows clients.

<8> Section 3.1.4: Windows 2000 and Windows Server 2003 use target level 5.0. Windows NT disables strict NDR data consistency checks.

<9> Section 3.1.4: Local-only methods MUST NOT be invoked by remote clients. The following methods are local-only:

Opnum	Description
0	Only used locally by Windows, never remotely.
5	Deprecated and not defined. It is never used.
13	Only used locally by Windows, never remotely.
21	Deprecated and not defined. It is never used.
24	Deprecated and not defined. It is never used.
25	Deprecated and not defined. It is never used.
29	Only used locally by Windows, never remotely.
30	Only used locally by Windows, never remotely.
32	Only used locally by Windows, never remotely.
33	Only used locally by Windows, never remotely.
34	Only used locally by Windows, never remotely.

<10> Section 3.1.4.1: For Windows NT and Windows 2000 servers, the method **rpc_ACCreateCursorEx** returns MQ_INFORMATION_REMOTE_OPERATION (0x400e03e8) to the client to indicate that a different queue manager is required to create the cursor. Upon receiving this return code, a client MAY proceed with cursor creation by calling **R_QMGetRemoteQueueName** to determine which queue manager to contact. For Windows Server 2003 and Windows Server 2008, this process was revised such that **rpc_ACCreateCursorEx** contacts the remote queue on behalf of the client, eliminating the need for **R_QMGetRemoteQueueName** to exist. Therefore, if invoked, **R_QMGetRemoteQueueName** takes no action and immediately raises the exception MQ_ERROR_ILLEGAL_OPERATION (0xc00e0064).

<11> Section 3.1.4.1: For Windows NT and Windows 2000 servers, the method **rpc_ACCreateCursorEx** returns MQ_INFORMATION_REMOTE_OPERATION (0x400e03e8) to the client to indicate that a different queue manager is required to create the cursor. Upon receiving this return code, a client MAY proceed with cursor creation by calling **R_QMGetRemoteQueueName** to determine which queue manager to contact. For Windows Server 2003 and Windows Server 2008, this process was revised such that **rpc_ACCreateCursorEx** contacts the remote queue on behalf of the client, eliminating the need for **R_QMGetRemoteQueueName** to exist. Therefore, if invoked, **R_QMGetRemoteQueueName** takes no action and immediately raises the exception MQ_ERROR_ILLEGAL_OPERATION (0xc00e0064).

<12> Section 3.1.4.1: For Windows NT and Windows 2000 servers, the method rpc_ACCreateCursorEx returns MQ_INFORMATION_REMOTE_OPERATION (0x400e03e8) to the client to indicate that a different queue manager is required to create the cursor. Upon receiving this return code, a client MAY proceed with cursor creation by calling R_QMGetRemoteQueueName to determine which queue manager to contact. For Windows Server 2003 and Windows Server 2008, this process was revised such that **rpc_ACCreateCursorEx** contacts the remote queue on behalf of the client, eliminating the need for **R_QMGetRemoteQueueName** to exist. Therefore, if invoked, **R_QMGetRemoteQueueName** takes no action and immediately raises the exception MQ_ERROR_ILLEGAL_OPERATION (0xc00e0064).

<13> Section 3.1.4.1: For Windows NT and Windows 2000 servers, the method **rpc_ACCreateCursorEx** returns MQ_INFORMATION_REMOTE_OPERATION (0x400e03e8) to the client to indicate that a different queue manager is required to create the cursor. Upon receiving this

return code, a client MAY proceed with cursor creation by calling **R_QMGetRemoteQueueName** to determine which queue manager to contact. For Windows Server 2003 and Windows Server 2008, this process was revised such that **rpc_ACCreateCursorEx** contacts the remote queue on behalf of the client, eliminating the need for **R_QMGetRemoteQueueName** to exist. Therefore, if invoked, **R_QMGetRemoteQueueName** takes no action and immediately raises the exception MQ_ERROR_ILLEGAL_OPERATION (0xc00e0064).

<14> Section 3.1.4.1: Workstation editions of Windows operating systems do not act as supporting servers for dependent clients, and return MQ_ERROR_WKS_CANT_SERVE_CLIENT (0xC00E0066) when dependent-client-only methods are received. This applies to Windows NT Workstation, Windows 2000 Workstation, Windows XP, and Windows Vista.

By default, Windows Server 2008 and Windows Server 2003 reject calls from dependent clients, although an administrator can enable support. By default, Windows 2000 Servers accept calls from dependent clients, although an administrator can disable support. Windows NT Servers do not restrict calls from dependent clients. If a dependent-client-specific method is invoked against a Windows Server that is configured to reject calls from dependent clients, the server takes no action and returns RPC_S_ACCESS_DENIED (0x00000005).

The following methods are called only by dependent clients, and are therefore wholly subject to the above restrictions:

Opnum	Name
6	**R_QMCreateObjectInternal**
7	**R_QMSetObjectSecurityInternal**
8	**R_QMGetObjectSecurityInternal**
9	**R_QMDeleteObject**
10	**R_QMGetObjectProperties**
11	**R_QMSetObjectProperties**
12	**R_QMObjectPathToObjectFormat**
14	**R_QMGetTmWhereabouts**
15	**R_QMEnlistTransaction**
16	**R_QMEnlistInternalTransaction**
17	**R_QMCommitTransaction**
18	**R_QMAbortTransaction**
19	**rpc_QMOpenQueueInternal**
20	**rpc_ACCloseHandle**
22	**rpc_ACCloseCursor**
23	**rpc_ACSetCursorProperties**
26	**rpc_ACHandleToFormatName**

Opnum	Name
27	**rpc_ACPurgeQueue**
28	**R_QMQueryQMRegistryInternal**

Additionally, all methods of qmcomm2 are subject to the above restrictions.

<15> Section 3.1.4.2: A Windows client passes its Windows process ID, as returned by the Windows SDK function GetCurrentProcessId. Servers MUST ignore the value of this parameter. Therefore, clients MAY pass 0x00000000.

<16> Section 3.1.4.2: Clients MUST consistently identify themselves to the server using a GUID generated at install time, and never subsequently modified.

<17> Section 3.1.4.2: These parameters are used to implement client access licensing restrictions. Such restrictions are only enforced by Windows NT, Windows 2000, and Windows Server 2003 servers. The parameters are ignored by Windows Vista and Windows Server 2008.

<18> Section 3.1.4.2: Windows NT, Windows 2000, and Windows Server 2003 servers MAY limit the number of unique callers. If the limit is exceeded, the server MAY take no action and return MQ_ERROR_DEPEND_WKS_LICENSE_OVERFLOW (0xc00e0067).

<19> Section 3.1.4.2: These parameters are used to implement client access licensing restrictions. Such restrictions are only enforced by Windows NT, Windows 2000, and Windows Server 2003 servers. The parameters are ignored by Windows Vista and Windows Server 2008.

<20> Section 3.1.4.2: These parameters are used to implement client access licensing restrictions. Such restrictions are only enforced by Windows NT, Windows 2000, and Windows Server 2003 servers. The parameters are ignored by Windows Vista and Windows Server 2008.

<21> Section 3.1.4.4: Clients for all versions of Windows pass a non-NULL pointer to a zeroed-out **CACTransferBufferV1** structure when calling **R_QMCreateRemoteCursor**. The server MUST ignore the **CACTransferBufferV1** pointer.

<22> Section 3.1.4.5: Windows applications typically invoke **R_QMCreateObjectInternal** indirectly via the Windows API function MQCreateQueue. The Windows API documentation for MQCreateQueue includes the following error codes. For descriptions of the following error code name/value pairs, see [MS-MQMQ] section 2.7 and [MSDN-MQEIC], in that order.

Name	Value
MQ_ERROR_ACCESS_DENIED	0xc00e0025
MQ_ERROR_ILLEGAL_PROPERTY_VALUE	x0c00e0018
MQ_ERROR_ILLEGAL_QUEUE_PATHNAME	0xc00e0014
MQ_ERROR_ILLEGAL_SECURITY_DESCRIPTOR	0xc00e0021
MQ_ERROR_INSUFFICIENT_PROPERTIES	0xc00e003f
MQ_ERROR_INVALID_OWNER	0xc00e0044
MQ_ERROR_INVALID_PARAMETER	0xc00e0006
MQ_ERROR_NO_DS	0xc00e0013

Name	Value
MQ_ERROR_PROPERTY	x0c00e0002
MQ_ERROR_PROPERTY_NOTALLOWED	0xc00e003e
MQ_ERROR_QUEUE_EXISTS	0xc00e0005
MQ_ERROR_SERVICE_NOT_AVAILABLE	0xc00e000b
MQ_ERROR_WRITE_NOT_ALLOWED	0xc00e0065
MQ_INFORMATION_FORMATNAME_BUFFER_TOO_SMALL	0x400e0009
MQ_INFORMATION_PROPERTY	0x400e0001
LDAP_BUSY	0x8007200e

<23> Section 3.1.4.5: Workstation editions of Windows operating systems do not act as supporting servers for dependent clients, and return MQ_ERROR_WKS_CANT_SERVE_CLIENT (0xC00E0066) when dependent-client-only methods are received. This applies to Windows NT Workstation, Windows 2000 Workstation, Windows XP, and Windows Vista.

By default, Windows Server 2008 and Windows Server 2003 reject calls from dependent clients, although an administrator can enable support. By default, Windows 2000 Servers accept calls from dependent clients, although an administrator can disable support. Windows NT Servers do not restrict calls from dependent clients. If a dependent-client-specific method is invoked against a Windows Server that is configured to reject calls from dependent clients, the server takes no action and returns RPC_S_ACCESS_DENIED (0x00000005).

The following methods are called only by dependent clients, and are therefore wholly subject to the above restrictions:

Opnum	Name
6	**R_QMCreateObjectInternal**
7	**R_QMSetObjectSecurityInternal**
8	**R_QMGetObjectSecurityInternal**
9	**R_QMDeleteObject**
10	**R_QMGetObjectProperties**
11	**R_QMSetObjectProperties**
12	**R_QMObjectPathToObjectFormat**
14	**R_QMGetTmWhereabouts**
15	**R_QMEnlistTransaction**
16	**R_QMEnlistInternalTransaction**
17	**R_QMCommitTransaction**
18	**R_QMAbortTransaction**

Opnum	Name
19	**rpc_QMOpenQueueInternal**
20	**rpc_ACCloseHandle**
22	**rpc_ACCloseCursor**
23	**rpc_ACSetCursorProperties**
26	**rpc_ACHandleToFormatName**
27	**rpc_ACPurgeQueue**
28	**R_QMQueryQMRegistryInternal**

Additionally, all methods of qmcomm2 are subject to the above restrictions.

Name	Value
RPC_S_ACCESS_DENIED	0x00000005
MQ_ERROR_WKS_CANT_SERVE_CLIENT	0xC00E0066

<24> Section 3.1.4.6: Windows applications typically invoke **R_QMSetObjectSecurityInternal** indirectly via the Windows API function MQSetQueueSecurity. The Windows API documentation for MQSetQueueSecurity includes the following error codes. For descriptions of the following error code name/value pairs, see [MS-MQMQ] section 2.7 and [MSDN-MQEIC], in that order.

Name	Value
MQ_ERROR_ACCESS_DENIED	0xc00e0025
MQ_ERROR_ILLEGAL_FORMATNAME	0xc00e001e
MQ_ERROR_INVALID_PARAMETER	0xc00e0006
MQ_ERROR_NO_DS	0xc00e0013
MQ_ERROR_PRIVILEGE_NOT_HELD	0xc00e0026
MQ_ERROR_SERVICE_NOT_AVAILABLE	0xc00e000b
MQ_ERROR_UNSUPPORTED_FORMATNAME_OPERATION	0xc00e0020
MQ_INFORMATION_OWNER_IGNORED	0x400e000b
LDAP_BUSY	0x8007200e

<25> Section 3.1.4.6: Workstation editions of Windows operating systems do not act as supporting servers for dependent clients, and return MQ_ERROR_WKS_CANT_SERVE_CLIENT (0xC00E0066) when dependent-client-only methods are received. This applies to Windows NT Workstation, Windows 2000 Workstation, Windows XP, and Windows Vista.

By default, Windows Server 2008 and Windows Server 2003 reject calls from dependent clients, although an administrator can enable support. By default, Windows 2000 Servers accept calls from dependent clients, although an administrator can disable support. Windows NT Servers do not

restrict calls from dependent clients. If a dependent-client-specific method is invoked against a Windows Server that is configured to reject calls from dependent clients, the server takes no action and returns RPC_S_ACCESS_DENIED (0x00000005).

The following methods are called only by dependent clients, and are therefore wholly subject to the above restrictions:

Opnum	Name
6	R_QMCreateObjectInternal
7	R_QMSetObjectSecurityInternal
8	R_QMGetObjectSecurityInternal
9	R_QMDeleteObject
10	R_QMGetObjectProperties
11	R_QMSetObjectProperties
12	R_QMObjectPathToObjectFormat
14	R_QMGetTmWhereabouts
15	R_QMEnlistTransaction
16	R_QMEnlistInternalTransaction
17	R_QMCommitTransaction
18	R_QMAbortTransaction
19	rpc_QMOpenQueueInternal
20	rpc_ACCloseHandle
22	rpc_ACCloseCursor
23	rpc_ACSetCursorProperties
26	rpc_ACHandleToFormatName
27	rpc_ACPurgeQueue
28	R_QMQueryQMRegistryInternal

Additionally, all methods of qmcomm2 are subject to the above restrictions.

Name	Value
RPC_S_ACCESS_DENIED	0x00000005
MQ_ERROR_WKS_CANT_SERVE_CLIENT	0xC00E0066

<26> Section 3.1.4.7: Windows applications typically invoke R_QMGetObjectSecurityInternal indirectly via the Windows API function MQGetQueueSecurity. The Windows API documentation for

MQGetQueueSecurity includes the following error codes. For descriptions of the following error code name/value pairs, see [MS-MQMQ] section 2.7 and [MSDN-MQEIC], in that order.

Name	Value
MQ_ERROR_ACCESS_DENIED	0xc00e0025
MQ_ERROR_ILLEGAL_FORMATNAME	0xc00e001e
MQ_ERROR_NO_DS	0xc00e0013
MQ_ERROR_PRIVILEGE_NOT_HELD	0xc00e0026
MQ_ERROR_SECURITY_DESCRIPTOR_TOO_SMALL	0xc00e0023
MQ_ERROR_UNSUPPORTED_FORMATNAME_OPERATION	0xc00e0020
LDAP_BUSY	0x8007200e

<27> Section 3.1.4.7: Workstation editions of Windows operating systems do not act as supporting servers for dependent clients, and return MQ_ERROR_WKS_CANT_SERVE_CLIENT (0xC00E0066) when dependent-client-only methods are received. This applies to Windows NT Workstation, Windows 2000 Workstation, Windows XP, and Windows Vista.

By default, Windows Server 2008 and Windows Server 2003 reject calls from dependent clients, although an administrator can enable support. By default, Windows 2000 Servers accept calls from dependent clients, although an administrator can disable support. Windows NT Servers do not restrict calls from dependent clients. If a dependent-client-specific method is invoked against a Windows Server that is configured to reject calls from dependent clients, the server takes no action and returns RPC_S_ACCESS_DENIED (0x00000005).

The following methods are called only by dependent clients, and are therefore wholly subject to the above restrictions:

Opnum	Name
6	**R_QMCreateObjectInternal**
7	**R_QMSetObjectSecurityInternal**
8	**R_QMGetObjectSecurityInternal**
9	**R_QMDeleteObject**
10	**R_QMGetObjectProperties**
11	**R_QMSetObjectProperties**
12	**R_QMObjectPathToObjectFormat**
14	**R_QMGetTmWhereabouts**
15	**R_QMEnlistTransaction**
16	**R_QMEnlistInternalTransaction**
17	**R_QMCommitTransaction**

Opnum	Name
18	R_QMAbortTransaction
19	rpc_QMOpenQueueInternal
20	rpc_ACCloseHandle
22	rpc_ACCloseCursor
23	rpc_ACSetCursorProperties
26	rpc_ACHandleToFormatName
27	rpc_ACPurgeQueue
28	R_QMQueryQMRegistryInternal

Additionally, all methods of qmcomm2 are subject to the above restrictions.

Name	Value
RPC_S_ACCESS_DENIED	0x00000005
MQ_ERROR_WKS_CANT_SERVE_CLIENT	0xC00E0066

<28> Section 3.1.4.8: Windows applications typically invoke R_QMDeleteObject indirectly via the Windows API function MQDeleteQueue. The Windows API documentation for MQDeleteQueue includes the following error codes. For descriptions of the following error code name/value pairs, see [MS-MQMQ] section 2.7 and [MSDN-MQEIC], in that order.

Name	Value
MQ_ERROR_ACCESS_DENIED	0xc00e0025
MQ_ERROR_ILLEGAL_FORMATNAME	0xc00e001e
MQ_ERROR_NO_DS	0xc00e0013
MQ_ERROR_SERVICE_NOT_AVAILABLE	0xc00e000b
MQ_ERROR_UNSUPPORTED_FORMATNAME_OPERATION	0xc00e0020
MQ_ERROR_WRITE_NOT_ALLOWED	0xc00e0065
LDAP_BUSY	0x8007200e

<29> Section 3.1.4.8: Workstation editions of Windows operating systems do not act as supporting servers for dependent clients, and return MQ_ERROR_WKS_CANT_SERVE_CLIENT (0xC00E0066) when dependent-client-only methods are received. This applies to Windows NT Workstation, Windows 2000 Workstation, Windows XP, and Windows Vista.

By default, Windows Server 2008 and Windows Server 2003 reject calls from dependent clients, although an administrator can enable support. By default, Windows 2000 Servers accept calls from dependent clients, although an administrator can disable support. Windows NT Servers do not restrict calls from dependent clients. If a dependent-client-specific method is invoked against a

Windows Server that is configured to reject calls from dependent clients, the server takes no action and returns RPC_S_ACCESS_DENIED (0x00000005).

The following methods are called only by dependent clients, and are therefore wholly subject to the above restrictions:

Opnum	Name
6	**R_QMCreateObjectInternal**
7	**R_QMSetObjectSecurityInternal**
8	**R_QMGetObjectSecurityInternal**
9	**R_QMDeleteObject**
10	**R_QMGetObjectProperties**
11	**R_QMSetObjectProperties**
12	**R_QMObjectPathToObjectFormat**
14	**R_QMGetTmWhereabouts**
15	**R_QMEnlistTransaction**
16	**R_QMEnlistInternalTransaction**
17	**R_QMCommitTransaction**
18	**R_QMAbortTransaction**
19	**rpc_QMOpenQueueInternal**
20	**rpc_ACCloseHandle**
22	**rpc_ACCloseCursor**
23	**rpc_ACSetCursorProperties**
26	**rpc_ACHandleToFormatName**
27	**rpc_ACPurgeQueue**
28	**R_QMQueryQMRegistryInternal**

Additionally, all methods of qmcomm2 are subject to the above restrictions.

Name	Value
RPC_S_ACCESS_DENIED	0x00000005
MQ_ERROR_WKS_CANT_SERVE_CLIENT	0xC00E0066

<30> Section 3.1.4.9: Windows applications typically invoke **R_QMGetObjectProperties** indirectly via the Windows API function MQGetQueueProperties. The Windows API documentation for MQGetQueueProperties includes the following error codes. For descriptions of the following error code name/value pairs, see [MS-MQMQ] section 2.7 and [MSDN-MQEIC], in that order.

Name	Value
MQ_ERROR_ACCESS_DENIED	0xc00e0025
MQ_ERROR_ILLEGAL_FORMATNAME	0xc00e001e
MQ_ERROR_ILLEGAL_PROPERTY_VT	0xc00e0019
MQ_ERROR_NO_DS	0xc00e0013
MQ_ERROR_SERVICE_NOT_AVAILABLE	0xc00e000b
MQ_ERROR_UNSUPPORTED_FORMATNAME_OPERATION	0xc00e0020
MQ_INFORMATION_DUPLICATE_PROPERTY	0x400e0005
MQ_INFORMATION_PROPERTY	0x400e0001
MQ_INFORMATION_UNSUPPORTED_PROPERTY	0x400e0004
LDAP_BUSY	0x8007200e

<31> Section 3.1.4.9: Workstation editions of Windows operating systems do not act as supporting servers for dependent clients, and return MQ_ERROR_WKS_CANT_SERVE_CLIENT (0xC00E0066) when dependent-client-only methods are received. This applies to Windows NT Workstation, Windows 2000 Workstation, Windows XP, and Windows Vista.

By default, Windows Server 2008 and Windows Server 2003 reject calls from dependent clients, although an administrator can enable support. By default, Windows 2000 Servers accept calls from dependent clients, although an administrator can disable support. Windows NT Servers do not restrict calls from dependent clients. If a dependent-client-specific method is invoked against a Windows Server that is configured to reject calls from dependent clients, the server takes no action and returns RPC_S_ACCESS_DENIED (0x00000005).

The following methods are called only by dependent clients, and are therefore wholly subject to the above restrictions:

Opnum	Name
6	**R_QMCreateObjectInternal**
7	**R_QMSetObjectSecurityInternal**
8	**R_QMGetObjectSecurityInternal**
9	**R_QMDeleteObject**
10	**R_QMGetObjectProperties**
11	**R_QMSetObjectProperties**
12	**R_QMObjectPathToObjectFormat**
14	**R_QMGetTmWhereabouts**
15	**R_QMEnlistTransaction**
16	**R_QMEnlistInternalTransaction**

Opnum	Name
17	**R_QMCommitTransaction**
18	**R_QMAbortTransaction**
19	**rpc_QMOpenQueueInternal**
20	**rpc_ACCloseHandle**
22	**rpc_ACCloseCursor**
23	**rpc_ACSetCursorProperties**
26	**rpc_ACHandleToFormatName**
27	**rpc_ACPurgeQueue**
28	**R_QMQueryQMRegistryInternal**

Additionally, all methods of qmcomm2 are subject to the above restrictions.

Name	Value
RPC_S_ACCESS_DENIED	0x00000005
MQ_ERROR_WKS_CANT_SERVE_CLIENT	0xC00E0066

<32> Section 3.1.4.10: Windows applications typically invoke **R_QMSetObjectProperties** indirectly via the Windows API function MQSetQueueProperties. The Windows API documentation for MQSetQueueProperties includes the following error codes. For descriptions of the following error code name/value pairs, see [MS-MQMQ] section 2.7 and [MSDN-MQEIC], in that order.

Name	Value
MQ_ERROR_ACCESS_DENIED	0xc00e0025
MQ_ERROR_ILLEGAL_FORMATNAME	0xc00e001e
MQ_ERROR_ILLEGAL_PROPERTY_VALUE	x0c00e0018
MQ_ERROR_INVALID_PARAMETER	0xc00e0006
MQ_ERROR_NO_DS	0xc00e0013
MQ_ERROR_PROPERTY	x0c00e0002
MQ_ERROR_SERVICE_NOT_AVAILABLE	0xc00e000b
MQ_ERROR_UNSUPPORTED_FORMATNAME_OPERATION	0xc00e0020
MQ_ERROR_WRITE_NOT_ALLOWED	0xc00e0065
MQ_INFORMATION_PROPERTY	0x400e0001
LDAP_BUSY	0x8007200e

<33> Section 3.1.4.10: Workstation editions of Windows operating systems do not act as supporting servers for dependent clients, and return MQ_ERROR_WKS_CANT_SERVE_CLIENT (0xC00E0066) when dependent-client-only methods are received. This applies to Windows NT Workstation, Windows 2000 Workstation, Windows XP, and Windows Vista.

By default, Windows Server 2008 and Windows Server 2003 reject calls from dependent clients, although an administrator can enable support. By default, Windows 2000 Servers accept calls from dependent clients, although an administrator can disable support. Windows NT Servers do not restrict calls from dependent clients. If a dependent-client-specific method is invoked against a Windows Server that is configured to reject calls from dependent clients, the server takes no action and returns RPC_S_ACCESS_DENIED (0x00000005).

The following methods are called only by dependent clients, and are therefore wholly subject to the above restrictions:

Opnum	Name
6	**R_QMCreateObjectInternal**
7	**R_QMSetObjectSecurityInternal**
8	**R_QMGetObjectSecurityInternal**
9	**R_QMDeleteObject**
10	**R_QMGetObjectProperties**
11	**R_QMSetObjectProperties**
12	**R_QMObjectPathToObjectFormat**
14	**R_QMGetTmWhereabouts**
15	**R_QMEnlistTransaction**
16	**R_QMEnlistInternalTransaction**
17	**R_QMCommitTransaction**
18	**R_QMAbortTransaction**
19	**rpc_QMOpenQueueInternal**
20	**rpc_ACCloseHandle**
22	**rpc_ACCloseCursor**
23	**rpc_ACSetCursorProperties**
26	**rpc_ACHandleToFormatName**
27	**rpc_ACPurgeQueue**
28	**R_QMQueryQMRegistryInternal**

Additionally, all methods of qmcomm2 are subject to the above restrictions.

Name	Value
RPC_S_ACCESS_DENIED	0x00000005
MQ_ERROR_WKS_CANT_SERVE_CLIENT	0xC00E0066

<34> Section 3.1.4.11: Windows applications typically invoke **R_QMObjectPathToObjectFormat** indirectly via the Windows API function MQPathNameToFormatName. The Windows API documentation for MQPathNameToFormatName includes the following error codes. For descriptions of the following error code name/value pairs, see [MS-MQMQ] section 2.7 and [MSDN-MQEIC], in that order.

Name	Value
MQ_ERROR_FORMATNAME_BUFFER_TOO_SMALL	0xc00e001f
MQ_ERROR_ILLEGAL_QUEUE_PATHNAME	0xc00e0014
MQ_ERROR_NO_DS	0xc00e0013
MQ_ERROR_QUEUE_NOT_FOUND	0xc00e0003
MQ_ERROR_SERVICE_NOT_AVAILABLE	0xc00e000b
LDAP_BUSY	0x8007200e

<35> Section 3.1.4.11: Workstation editions of Windows operating systems do not act as supporting servers for dependent clients, and return MQ_ERROR_WKS_CANT_SERVE_CLIENT (0xC00E0066) when dependent-client-only methods are received. This applies to Windows NT Workstation, Windows 2000 Workstation, Windows XP, and Windows Vista.

By default, Windows Server 2008 and Windows Server 2003 reject calls from dependent clients, although an administrator can enable support. By default, Windows 2000 Servers accept calls from dependent clients, although an administrator can disable support. Windows NT Servers do not restrict calls from dependent clients. If a dependent-client-specific method is invoked against a Windows Server that is configured to reject calls from dependent clients, the server takes no action and returns RPC_S_ACCESS_DENIED (0x00000005).

The following methods are called only by dependent clients, and are therefore wholly subject to the above restrictions:

Opnum	Name
6	**R_QMCreateObjectInternal**
7	**R_QMSetObjectSecurityInternal**
8	**R_QMGetObjectSecurityInternal**
9	**R_QMDeleteObject**
10	**R_QMGetObjectProperties**
11	**R_QMSetObjectProperties**
12	**R_QMObjectPathToObjectFormat**

Opnum	Name
14	R_QMGetTmWhereabouts
15	R_QMEnlistTransaction
16	R_QMEnlistInternalTransaction
17	R_QMCommitTransaction
18	R_QMAbortTransaction
19	rpc_QMOpenQueueInternal
20	rpc_ACCloseHandle
22	rpc_ACCloseCursor
23	rpc_ACSetCursorProperties
26	rpc_ACHandleToFormatName
27	rpc_ACPurgeQueue
28	R_QMQueryQMRegistryInternal

Additionally, all methods of qmcomm2 are subject to the above restrictions.

Name	Value
RPC_S_ACCESS_DENIED	0x00000005
MQ_ERROR_WKS_CANT_SERVE_CLIENT	0xC00E0066

<36> Section 3.1.4.12: Workstation editions of Windows operating systems do not act as supporting servers for dependent clients, and return MQ_ERROR_WKS_CANT_SERVE_CLIENT (0xC00E0066) when dependent-client-only methods are received. This applies to Windows NT Workstation, Windows 2000 Workstation, Windows XP, and Windows Vista.

By default, Windows Server 2008 and Windows Server 2003 reject calls from dependent clients, although an administrator can enable support. By default, Windows 2000 Servers accept calls from dependent clients, although an administrator can disable support. Windows NT Servers do not restrict calls from dependent clients. If a dependent-client-specific method is invoked against a Windows Server that is configured to reject calls from dependent clients, the server takes no action and returns RPC_S_ACCESS_DENIED (0x00000005).

The following methods are called only by dependent clients, and are therefore wholly subject to the above restrictions:

Opnum	Name
6	R_QMCreateObjectInternal
7	R_QMSetObjectSecurityInternal
8	R_QMGetObjectSecurityInternal

Opnum	Name
9	R_QMDeleteObject
10	R_QMGetObjectProperties
11	R_QMSetObjectProperties
12	R_QMObjectPathToObjectFormat
14	R_QMGetTmWhereabouts
15	R_QMEnlistTransaction
16	R_QMEnlistInternalTransaction
17	R_QMCommitTransaction
18	R_QMAbortTransaction
19	rpc_QMOpenQueueInternal
20	rpc_ACCloseHandle
22	rpc_ACCloseCursor
23	rpc_ACSetCursorProperties
26	rpc_ACHandleToFormatName
27	rpc_ACPurgeQueue
28	R_QMQueryQMRegistryInternal

Additionally, all methods of qmcomm2 are subject to the above restrictions.

<37> Section 3.1.4.13: Workstation editions of Windows operating systems do not act as supporting servers for dependent clients, and return MQ_ERROR_WKS_CANT_SERVE_CLIENT (0xC00E0066) when dependent-client-only methods are received. This applies to Windows NT Workstation, Windows 2000 Workstation, Windows XP, and Windows Vista.

By default, Windows Server 2008 and Windows Server 2003 reject calls from dependent clients, although an administrator can enable support. By default, Windows 2000 Servers accept calls from dependent clients, although an administrator can disable support. Windows NT Servers do not restrict calls from dependent clients. If a dependent-client-specific method is invoked against a Windows Server that is configured to reject calls from dependent clients, the server takes no action and returns RPC_S_ACCESS_DENIED (0x00000005).

The following methods are called only by dependent clients, and are therefore wholly subject to the above restrictions:

Opnum	Name
6	R_QMCreateObjectInternal
7	R_QMSetObjectSecurityInternal
8	R_QMGetObjectSecurityInternal

Opnum	Name
9	**R_QMDeleteObject**
10	**R_QMGetObjectProperties**
11	**R_QMSetObjectProperties**
12	**R_QMObjectPathToObjectFormat**
14	**R_QMGetTmWhereabouts**
15	**R_QMEnlistTransaction**
16	**R_QMEnlistInternalTransaction**
17	**R_QMCommitTransaction**
18	**R_QMAbortTransaction**
19	**rpc_QMOpenQueueInternal**
20	**rpc_ACCloseHandle**
22	**rpc_ACCloseCursor**
23	**rpc_ACSetCursorProperties**
26	**rpc_ACHandleToFormatName**
27	**rpc_ACPurgeQueue**
28	**R_QMQueryQMRegistryInternal**

Additionally, all methods of qmcomm2 are subject to the above restrictions.

<38> Section 3.1.4.14: All Windows clients produce new **XACTUOW** values by calling the Windows RPC function UuidCreate.

<39> Section 3.1.4.14: Workstation editions of Windows operating systems do not act as supporting servers for dependent clients, and return MQ_ERROR_WKS_CANT_SERVE_CLIENT (0xC00E0066) when dependent-client-only methods are received. This applies to Windows NT Workstation, Windows 2000 Workstation, Windows XP, and Windows Vista.

By default, Windows Server 2008 and Windows Server 2003 reject calls from dependent clients, although an administrator can enable support. By default, Windows 2000 Servers accept calls from dependent clients, although an administrator can disable support. Windows NT Servers do not restrict calls from dependent clients. If a dependent-client-specific method is invoked against a Windows Server that is configured to reject calls from dependent clients, the server takes no action and returns RPC_S_ACCESS_DENIED (0x00000005).

The following methods are called only by dependent clients, and are therefore wholly subject to the above restrictions:

Opnum	Name
6	**R_QMCreateObjectInternal**

Opnum	Name
7	R_QMSetObjectSecurityInternal
8	R_QMGetObjectSecurityInternal
9	R_QMDeleteObject
10	R_QMGetObjectProperties
11	R_QMSetObjectProperties
12	R_QMObjectPathToObjectFormat
14	R_QMGetTmWhereabouts
15	R_QMEnlistTransaction
16	R_QMEnlistInternalTransaction
17	R_QMCommitTransaction
18	R_QMAbortTransaction
19	rpc_QMOpenQueueInternal
20	rpc_ACCloseHandle
22	rpc_ACCloseCursor
23	rpc_ACSetCursorProperties
26	rpc_ACHandleToFormatName
27	rpc_ACPurgeQueue
28	R_QMQueryQMRegistryInternal

Additionally, all methods of qmcomm2 are subject to the above restrictions.

<40> Section 3.1.4.15: Workstation editions of Windows operating systems do not act as supporting servers for dependent clients, and return MQ_ERROR_WKS_CANT_SERVE_CLIENT (0xC00E0066) when dependent-client-only methods are received. This applies to Windows NT Workstation, Windows 2000 Workstation, Windows XP, and Windows Vista.

By default, Windows Server 2008 and Windows Server 2003 reject calls from dependent clients, although an administrator can enable support. By default, Windows 2000 Servers accept calls from dependent clients, although an administrator can disable support. Windows NT Servers do not restrict calls from dependent clients. If a dependent-client-specific method is invoked against a Windows Server that is configured to reject calls from dependent clients, the server takes no action and returns RPC_S_ACCESS_DENIED (0x00000005).

The following methods are called only by dependent clients, and are therefore wholly subject to the above restrictions:

Opnum	Name
6	R_QMCreateObjectInternal

Opnum	Name
7	R_QMSetObjectSecurityInternal
8	R_QMGetObjectSecurityInternal
9	R_QMDeleteObject
10	R_QMGetObjectProperties
11	R_QMSetObjectProperties
12	R_QMObjectPathToObjectFormat
14	R_QMGetTmWhereabouts
15	R_QMEnlistTransaction
16	R_QMEnlistInternalTransaction
17	R_QMCommitTransaction
18	R_QMAbortTransaction
19	rpc_QMOpenQueueInternal
20	rpc_ACCloseHandle
22	rpc_ACCloseCursor
23	rpc_ACSetCursorProperties
26	rpc_ACHandleToFormatName
27	rpc_ACPurgeQueue
28	R_QMQueryQMRegistryInternal

Additionally, all methods of qmcomm2 are subject to the above restrictions.

<41> Section 3.1.4.16: Workstation editions of Windows operating systems do not act as supporting servers for dependent clients, and return MQ_ERROR_WKS_CANT_SERVE_CLIENT (0xC00E0066) when dependent-client-only methods are received. This applies to Windows NT Workstation, Windows 2000 Workstation, Windows XP, and Windows Vista.

By default, Windows Server 2008 and Windows Server 2003 reject calls from dependent clients, although an administrator can enable support. By default, Windows 2000 Servers accept calls from dependent clients, although an administrator can disable support. Windows NT Servers do not restrict calls from dependent clients. If a dependent-client-specific method is invoked against a Windows Server that is configured to reject calls from dependent clients, the server takes no action and returns RPC_S_ACCESS_DENIED (0x00000005).

The following methods are called only by dependent clients, and are therefore wholly subject to the above restrictions:

Opnum	Name
6	R_QMCreateObjectInternal

Opnum	Name
7	R_QMSetObjectSecurityInternal
8	R_QMGetObjectSecurityInternal
9	R_QMDeleteObject
10	R_QMGetObjectProperties
11	R_QMSetObjectProperties
12	R_QMObjectPathToObjectFormat
14	R_QMGetTmWhereabouts
15	R_QMEnlistTransaction
16	R_QMEnlistInternalTransaction
17	R_QMCommitTransaction
18	R_QMAbortTransaction
19	rpc_QMOpenQueueInternal
20	rpc_ACCloseHandle
22	rpc_ACCloseCursor
23	rpc_ACSetCursorProperties
26	rpc_ACHandleToFormatName
27	rpc_ACPurgeQueue
28	R_QMQueryQMRegistryInternal

Additionally, all methods of qmcomm2 are subject to the above restrictions.

<42> Section 3.1.4.17: Clients MUST consistently identify themselves to the server using a GUID generated at install time, and never subsequently modified.

<43> Section 3.1.4.17: These parameters are used to implement client access licensing restrictions. Such restrictions are only enforced by Windows NT, Windows 2000, and Windows Server 2003 servers. The parameters are ignored by Windows Vista and Windows Server 2008.

<44> Section 3.1.4.17: Windows NT, Windows 2000, and Windows Server 2003 servers MAY limit the number of unique callers. If the limit is exceeded, the server MAY take no action and return MQ_ERROR_DEPEND_WKS_LICENSE_OVERFLOW (0xc00e0067).

<45> Section 3.1.4.17: Windowsclients obtain this string from the Windows SDK function GetComputerName.

<46> Section 3.1.4.17: dependent client access licensing restrictions are only enforced by Windows NT, Windows 2000, and Windows Server 2003 supporting servers.

<47> Section 3.1.4.17: All versions of Windows servers accept the value 0x00000000 to indicate that the TCP/IP protocol sequence SHOULD be used when connecting to a remote queue manager

for remote read.Windows NT and Windows 2000 servers accept the value 0x00000003 to indicate that the IPX/SPX protocol sequence SHOULD be used when connecting to a remote queue manager for remote read. Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 servers ignore the parameter.

<48> Section 3.1.4.17: Windows applications typically invoke **rpc_QMOpenQueueInternal** indirectly via the Windows API function MQOpenQueue. The Windows API documentation for MQOpenQueue includes the following error codes. For descriptions of the following error code name/value pairs, see [MS-MQMQ] section 2.7 and [MSDN-MQEIC], in that order.

Name	Value
MQ_ERROR_ACCESS_DENIED	0xc00e0025
MQ_ERROR_ILLEGAL_FORMATNAME	0xc00e001e
MQ_ERROR_INVALID_PARAMETER	0xc00e0006
MQ_ERROR_NO_DS	0xc00e0013
MQ_ERROR_QUEUE_NOT_FOUND	0xc00e0003
MQ_ERROR_REMOTE_MACHINE_NOT_AVAILABLE	0xc00e0069
MQ_ERROR_SERVICE_NOT_AVAILABLE	0xc00e000b
MQ_ERROR_SHARING_VIOLATION	0xc00e0009
MQ_ERROR_UNSUPPORTED_ACCESS_MODE	0xc00e0045
MQ_ERROR_UNSUPPORTED_FORMATNAME_OPERATION	0xc00e0020

<49> Section 3.1.4.17: Workstation editions of Windows operating systems do not act as supporting servers for dependent clients, and return MQ_ERROR_WKS_CANT_SERVE_CLIENT (0xC00E0066) when dependent-client-only methods are received. This applies to Windows NT Workstation, Windows 2000 Workstation, Windows XP, and Windows Vista.

By default, Windows Server 2008 and Windows Server 2003 reject calls from dependent clients, although an administrator can enable support. By default, Windows 2000 Servers accept calls from dependent clients, although an administrator can disable support. Windows NT Servers do not restrict calls from dependent clients. If a dependent-client-specific method is invoked against a Windows Server that is configured to reject calls from dependent clients, the server takes no action and returns RPC_S_ACCESS_DENIED (0x00000005).

The following methods are called only by dependent clients, and are therefore wholly subject to the above restrictions:

Opnum	Name
6	**R_QMCreateObjectInternal**
7	**R_QMSetObjectSecurityInternal**
8	**R_QMGetObjectSecurityInternal**
9	**R_QMDeleteObject**

Opnum	Name
10	**R_QMGetObjectProperties**
11	**R_QMSetObjectProperties**
12	**R_QMObjectPathToObjectFormat**
14	**R_QMGetTmWhereabouts**
15	**R_QMEnlistTransaction**
16	**R_QMEnlistInternalTransaction**
17	**R_QMCommitTransaction**
18	**R_QMAbortTransaction**
19	**rpc_QMOpenQueueInternal**
20	**rpc_ACCloseHandle**
22	**rpc_ACCloseCursor**
23	**rpc_ACSetCursorProperties**
26	**rpc_ACHandleToFormatName**
27	**rpc_ACPurgeQueue**
28	**R_QMQueryQMRegistryInternal**

Additionally, all methods of qmcomm2 are subject to the above restrictions.

Name	Value
RPC_S_ACCESS_DENIED	0x00000005
MQ_ERROR_WKS_CANT_SERVE_CLIENT	0xC00E0066

<50> Section 3.1.4.17: When communicating with an [MS-MQQP] server, all Windows Server implementations of [MS-MQMP] wait a maximum of five (5) minutes for RPC requests issued against an [MS-MQQP] server to respond. If five minutes elapse with no response, the blocked client request is explicitly cancelled via the "Cancel" RPC Service Primitive as described in [C706] section 7.2.3. The client's Cancel Timeout Timer ([MS-RPCE] section 3.2.2.2.2) is configured to zero (0) so that the client is immediately unblocked.

<51> Section 3.1.4.18: Windows applications typically invoke **rpc_ACCloseHandle** indirectly via the Windows API function MQCloseQueue. The Windows API documentation for MQCloseQueue includes the following error code. For descriptions of the following error code name/value pair, see [MS-MQMQ] section 2.7 and [MSDN-MQEIC], in that order.

Name	Value
MQ_ERROR_INVALID_HANDLE	0xc00e0007

<52> Section 3.1.4.18: Workstation editions of Windows operating systems do not act as supporting servers for dependent clients, and return MQ_ERROR_WKS_CANT_SERVE_CLIENT (0xC00E0066) when dependent-client-only methods are received. This applies to Windows NT Workstation, Windows 2000 Workstation, Windows XP, and Windows Vista.

By default, Windows Server 2008 and Windows Server 2003 reject calls from dependent clients, although an administrator can enable support. By default, Windows 2000 Servers accept calls from dependent clients, although an administrator can disable support. Windows NT Servers do not restrict calls from dependent clients. If a dependent-client-specific method is invoked against a Windows Server that is configured to reject calls from dependent clients, the server takes no action and returns RPC_S_ACCESS_DENIED (0x00000005).

The following methods are called only by dependent clients, and are therefore wholly subject to the above restrictions:

Opnum	Name
6	**R_QMCreateObjectInternal**
7	**R_QMSetObjectSecurityInternal**
8	**R_QMGetObjectSecurityInternal**
9	**R_QMDeleteObject**
10	**R_QMGetObjectProperties**
11	**R_QMSetObjectProperties**
12	**R_QMObjectPathToObjectFormat**
14	**R_QMGetTmWhereabouts**
15	**R_QMEnlistTransaction**
16	**R_QMEnlistInternalTransaction**
17	**R_QMCommitTransaction**
18	**R_QMAbortTransaction**
19	**rpc_QMOpenQueueInternal**
20	**rpc_ACCloseHandle**
22	**rpc_ACCloseCursor**
23	**rpc_ACSetCursorProperties**
26	**rpc_ACHandleToFormatName**
27	**rpc_ACPurgeQueue**
28	**R_QMQueryQMRegistryInternal**

Additionally, all methods of qmcomm2 are subject to the above restrictions.

Name	Value
RPC_S_ACCESS_DENIED	0x00000005
MQ_ERROR_WKS_CANT_SERVE_CLIENT	0xC00E0066

<53> Section 3.1.4.18: All Windows Server implementations invoke the [MS-MQQP] method **RemoteQMCloseQueue** using a parallel process, permitting immediate return of control to the client. Note that this introduces the possibility that **RemoteQMCloseQueue** could fail, and that the client would not be informed.

<54> Section 3.1.4.19: Windows applications typically invoke **rpc_ACCloseCursor** indirectly via the Windows API function MQCloseCursor. The Windows API documentation for MQCloseCursor includes the following error code. For descriptions of the following error code name/value pair, see [MS-MQMQ] section 2.7 and [MSDN-MQEIC], in that order.

Name	Value
MQ_ERROR_INVALID_HANDLE	0xc00e0007

<55> Section 3.1.4.19: Workstation editions of Windows operating systems do not act as supporting servers for dependent clients, and return MQ_ERROR_WKS_CANT_SERVE_CLIENT (0xC00E0066) when dependent-client-only methods are received. This applies to Windows NT Workstation, Windows 2000 Workstation, Windows XP, and Windows Vista.

By default, Windows Server 2008 and Windows Server 2003 reject calls from dependent clients, although an administrator can enable support. By default, Windows 2000 Servers accept calls from dependent clients, although an administrator can disable support. Windows NT Servers do not restrict calls from dependent clients. If a dependent-client-specific method is invoked against a Windows Server that is configured to reject calls from dependent clients, the server takes no action and returns RPC_S_ACCESS_DENIED (0x00000005).

The following methods are called only by dependent clients, and are therefore wholly subject to the above restrictions:

Opnum	Name
6	**R_QMCreateObjectInternal**
7	**R_QMSetObjectSecurityInternal**
8	**R_QMGetObjectSecurityInternal**
9	**R_QMDeleteObject**
10	**R_QMGetObjectProperties**
11	**R_QMSetObjectProperties**
12	**R_QMObjectPathToObjectFormat**
14	**R_QMGetTmWhereabouts**
15	**R_QMEnlistTransaction**
16	**R_QMEnlistInternalTransaction**

Opnum	Name
17	**R_QMCommitTransaction**
18	**R_QMAbortTransaction**
19	**rpc_QMOpenQueueInternal**
20	**rpc_ACCloseHandle**
22	**rpc_ACCloseCursor**
23	**rpc_ACSetCursorProperties**
26	**rpc_ACHandleToFormatName**
27	**rpc_ACPurgeQueue**
28	**R_QMQueryQMRegistryInternal**

Additionally, all methods of qmcomm2 are subject to the above restrictions.

Name	Value
RPC_S_ACCESS_DENIED	0x00000005
MQ_ERROR_WKS_CANT_SERVE_CLIENT	0xC00E0066

<56> Section 3.1.4.20: Due to revisions to the cursor creation process, the method **rpc_ACSetCursorProperties** is obsolete on Windows Server 2003 and Windows Vista and Windows Server 2008. If the server implementation does not support **rpc_ACSetCursorProperties**, it should take no action and return MQ_ERROR_ILLEGAL_OPERATION (0xc00e0064). Note that this differs from the behavior of other obsolete methods that raise MQ_ERROR_ILLEGAL_OPERATION as an RPC exception.

<57> Section 3.1.4.20: For Windows NT and Windows 2000servers, the method **rpc_ACCreateCursorEx** returns MQ_INFORMATION_REMOTE_OPERATION (0x400e03e8) to the client to indicate that a different queue manager is required to create the cursor. Upon receiving this return code, a client MAY proceed with cursor creation by calling **R_QMGetRemoteQueueName** to determine which queue manager to contact. For Windows Server 2003 and Windows Vista and Windows Server 2008, this process was revised such that **rpc_ACCreateCursorEx** contacts the remote queue on behalf of the client, eliminating the need for **R_QMGetRemoteQueueName** to exist. Therefore, if invoked, **R_QMGetRemoteQueueName** takes no action and immediately raises the exception MQ_ERROR_ILLEGAL_OPERATION (0xc00e0064).

<58> Section 3.1.4.20: Workstation editions of Windows operating systems do not act as supporting servers for dependent clients, and return MQ_ERROR_WKS_CANT_SERVE_CLIENT (0xC00E0066) when dependent-client-only methods are received. This applies to Windows NT Workstation, Windows 2000 Workstation, Windows XP, and Windows Vista.

By default, Windows Server 2008 and Windows Server 2003 reject calls from dependent clients, although an administrator can enable support. By default, Windows 2000 Servers accept calls from dependent clients, although an administrator can disable support. Windows NT Servers do not restrict calls from dependent clients. If a dependent-client-specific method is invoked against a Windows Server that is configured to reject calls from dependent clients, the server takes no action and returns RPC_S_ACCESS_DENIED (0x00000005).

The following methods are called only by dependent clients, and are therefore wholly subject to the above restrictions:

Opnum	Name
6	**R_QMCreateObjectInternal**
7	**R_QMSetObjectSecurityInternal**
8	**R_QMGetObjectSecurityInternal**
9	**R_QMDeleteObject**
10	**R_QMGetObjectProperties**
11	**R_QMSetObjectProperties**
12	**R_QMObjectPathToObjectFormat**
14	**R_QMGetTmWhereabouts**
15	**R_QMEnlistTransaction**
16	**R_QMEnlistInternalTransaction**
17	**R_QMCommitTransaction**
18	**R_QMAbortTransaction**
19	**rpc_QMOpenQueueInternal**
20	**rpc_ACCloseHandle**
22	**rpc_ACCloseCursor**
23	**rpc_ACSetCursorProperties**
26	**rpc_ACHandleToFormatName**
27	**rpc_ACPurgeQueue**
28	**R_QMQueryQMRegistryInternal**

Additionally, all methods of qmcomm2 are subject to the above restrictions.

<59> Section 3.1.4.21: Windows applications typically invoke **rpc_ACHandleToFormatName** indirectly via the Windows API function MQHandleToFormatName. The Windows API documentation for MQHandleToFormatName includes the following error codes. For descriptions of the following error code name/value pairs, see [MS-MQMQ] section 2.7 and [MSDN-MQEIC], in that order.

Name	Value
MQ_ERROR_FORMATNAME_BUFFER_TOO_SMALL	0xc00e001f
MQ_ERROR_INVALID_HANDLE	0xc00e0007
MQ_ERROR_SERVICE_NOT_AVAILABLE	0xc00e000b
MQ_ERROR_STALE_HANDLE	0xc00e0056

<60> Section 3.1.4.21: Workstation editions of Windows operating systems do not act as supporting servers for dependent clients, and return MQ_ERROR_WKS_CANT_SERVE_CLIENT (0xC00E0066) when dependent-client-only methods are received. This applies to Windows NT Workstation, Windows 2000 Workstation, Windows XP, and Windows Vista.

By default, Windows Server 2008 and Windows Server 2003 reject calls from dependent clients, although an administrator can enable support. By default, Windows 2000 Servers accept calls from dependent clients, although an administrator can disable support. Windows NT Servers do not restrict calls from dependent clients. If a dependent-client-specific method is invoked against a Windows Server that is configured to reject calls from dependent clients, the server takes no action and returns RPC_S_ACCESS_DENIED (0x00000005).

The following methods are called only by dependent clients, and are therefore wholly subject to the above restrictions:

Opnum	Name
6	**R_QMCreateObjectInternal**
7	**R_QMSetObjectSecurityInternal**
8	**R_QMGetObjectSecurityInternal**
9	**R_QMDeleteObject**
10	**R_QMGetObjectProperties**
11	**R_QMSetObjectProperties**
12	**R_QMObjectPathToObjectFormat**
14	**R_QMGetTmWhereabouts**
15	**R_QMEnlistTransaction**
16	**R_QMEnlistInternalTransaction**
17	**R_QMCommitTransaction**
18	**R_QMAbortTransaction**
19	**rpc_QMOpenQueueInternal**
20	**rpc_ACCloseHandle**
22	**rpc_ACCloseCursor**
23	**rpc_ACSetCursorProperties**
26	**rpc_ACHandleToFormatName**
27	**rpc_ACPurgeQueue**
28	**R_QMQueryQMRegistryInternal**

Additionally, all methods of qmcomm2 are subject to the above restrictions.

Name	Value
RPC_S_ACCESS_DENIED	0x00000005
MQ_ERROR_WKS_CANT_SERVE_CLIENT	0xC00E0066

<61> Section 3.1.4.22: Windows applications typically invoke **rpc_ACPurgeQueue** indirectly via the Windows API function MQPurgeQueue. The Windows API documentation for MQPurgeQueue includes the following error code. For descriptions of the following error code name/value pair, see [MS-MQMQ] section 2.7 and [MSDN-MQEIC], in that order.

Name	Value
MQ_ERROR_INVALID_HANDLE	0xc00e0007

<62> Section 3.1.4.22: Workstation editions of Windows operating systems do not act as supporting servers for dependent clients, and return MQ_ERROR_WKS_CANT_SERVE_CLIENT (0xC00E0066) when dependent-client-only methods are received. This applies to Windows NT Workstation, Windows 2000 Workstation, Windows XP, and Windows Vista.

By default, Windows Server 2008 and Windows Server 2003 reject calls from dependent clients, although an administrator can enable support. By default, Windows 2000 Servers accept calls from dependent clients, although an administrator can disable support. Windows NT Servers do not restrict calls from dependent clients. If a dependent-client-specific method is invoked against a Windows Server that is configured to reject calls from dependent clients, the server takes no action and returns RPC_S_ACCESS_DENIED (0x00000005).

The following methods are called only by dependent clients, and are therefore wholly subject to the above restrictions:

Opnum	Name
6	**R_QMCreateObjectInternal**
7	**R_QMSetObjectSecurityInternal**
8	**R_QMGetObjectSecurityInternal**
9	**R_QMDeleteObject**
10	**R_QMGetObjectProperties**
11	**R_QMSetObjectProperties**
12	**R_QMObjectPathToObjectFormat**
14	**R_QMGetTmWhereabouts**
15	**R_QMEnlistTransaction**
16	**R_QMEnlistInternalTransaction**
17	**R_QMCommitTransaction**
18	**R_QMAbortTransaction**
19	**rpc_QMOpenQueueInternal**

Opnum	Name
20	rpc_ACCloseHandle
22	rpc_ACCloseCursor
23	rpc_ACSetCursorProperties
26	rpc_ACHandleToFormatName
27	rpc_ACPurgeQueue
28	R_QMQueryQMRegistryInternal

Additionally, all methods of qmcomm2 are subject to the above restrictions.

Name	Value
RPC_S_ACCESS_DENIED	0x00000005
MQ_ERROR_WKS_CANT_SERVE_CLIENT	0xC00E0066

<63> Section 3.1.4.23: Windows servers retrieve the value from the registry string "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSMQ\Parameters\MachineCache\MQISServer".

<64> Section 3.1.4.23: For Windows NT and Windows 2000 Server, this value defaults to "7776000" (90 days). For Windows Server 2003 and Windows Server 2008, the default value is "345600" (4 days).

<65> Section 3.1.4.23: Windows servers store and retrieve these values from the registry.

<66> Section 3.1.4.23: Windows servers store and retrieve these values from the registry.

<67> Section 3.1.4.23: Windows servers store and retrieve these values from the registry.

<68> Section 3.1.4.23: Workstation editions of Windows operating systems do not act as supporting servers for dependent clients, and return MQ_ERROR_WKS_CANT_SERVE_CLIENT (0xC00E0066) when dependent-client-only methods are received. This applies to Windows NT Workstation, Windows 2000 Workstation, Windows XP, and Windows Vista.

By default, Windows Server 2008 and Windows Server 2003 reject calls from dependent clients, although an administrator can enable support. By default, Windows 2000 Servers accept calls from dependent clients, although an administrator can disable support. Windows NT Servers do not restrict calls from dependent clients. If a dependent-client-specific method is invoked against a Windows Server that is configured to reject calls from dependent clients, the server takes no action and returns RPC_S_ACCESS_DENIED (0x00000005).

The following methods are called only by dependent clients, and are therefore wholly subject to the above restrictions:

Opnum	Name
6	R_QMCreateObjectInternal
7	R_QMSetObjectSecurityInternal

Opnum	Name
8	R_QMGetObjectSecurityInternal
9	R_QMDeleteObject
10	R_QMGetObjectProperties
11	R_QMSetObjectProperties
12	R_QMObjectPathToObjectFormat
14	R_QMGetTmWhereabouts
15	R_QMEnlistTransaction
16	R_QMEnlistInternalTransaction
17	R_QMCommitTransaction
18	R_QMAbortTransaction
19	rpc_QMOpenQueueInternal
20	rpc_ACCloseHandle
22	rpc_ACCloseCursor
23	rpc_ACSetCursorProperties
26	rpc_ACHandleToFormatName
27	rpc_ACPurgeQueue
28	R_QMQueryQMRegistryInternal

Additionally, all methods of qmcomm2 are subject to the above restrictions.

Name	Value
RPC_S_ACCESS_DENIED	0x00000005
MQ_ERROR_WKS_CANT_SERVE_CLIENT	0xC00E0066

<69> Section 3.1.4.24: RPC over SPX is only supported by Windows NT and Windows 2000. This value is not supported by Windows XP, Windows Server 2003, and Windows Vista and Windows Server 2008, and the server MUST return 0x00000000 to indicate failure.

<70> Section 3.1.4.24: RPC over SPX is only supported by Windows NT and Windows 2000. This value is not supported by Windows XP, Windows Server 2003, and Windows Vista and Windows Server 2008, and the server MUST return 0x00000000 to indicate failure.

<71> Section 3.1.5.1: Windows applications typically invoke **QMSendMessageInternalEx** indirectly via the Windows API function MQSendMessage. The Windows API documentation for MQSendMessage includes the following error codes. For descriptions of the following error code name/value pairs, see [MS-MQMQ] section 2.7 and [MSDN-MQEIC], in that order.

Name	Value
MQ_ERROR_ACCESS_DENIED	0xc00e0025
MQ_ERROR_BAD_SECURITY_CONTEXT	0xc00e0035
MQ_ERROR_CERTIFICATE_NOT_PROVIDED	0xc00e006d
MQ_ERROR_CORRUPTED_INTERNAL_CERTIFICATE	0xc00e002d
MQ_ERROR_CORRUPTED_PERSONAL_CERT_STORE	0xc00e0031
MQ_ERROR_CORRUPTED_SECURITY_DATA	0xc00e0030
MQ_ERROR_COULD_NOT_GET_USER_SID	0xc00e0036
MQ_ERROR_DTC_CONNECT	0xc00e004c
MQ_ERROR_ILLEGAL_FORMATNAME	0xc00e001e
MQ_ERROR_INSUFFICIENT_RESOURCES	0xc00e0027
MQ_ERROR_INVALID_CERTIFICATE	0xc00e002c
MQ_ERROR_INVALID_HANDLE	0xc00e0007
MQ_ERROR_INVALID_PARAMETER	0xc00e0006
MQ_ERROR_MESSAGE_STORAGE_FAILED	0xc00e002a
MQ_ERROR_NO_INTERNAL_USER_CERT	0xc00e002f
MQ_ERROR_PROPERTY	x0c00e0002
MQ_ERROR_SERVICE_NOT_AVAILABLE	0xc00e000b
MQ_ERROR_STALE_HANDLE	0xc00e0056
MQ_ERROR_TRANSACTION_USAGE	0xc00e0050
MQ_ERROR_TRANSACTION_ENLIST	0xc00e0058
MQ_ERROR_UNSUPPORTED_FORMATNAME_OPERATION	0xc00e0020
MQ_INFORMATION_PROPERTY	0x400e0001

<72> Section 3.1.5.1: Workstation editions of Windows operating systems do not act as supporting servers for dependent clients, and return MQ_ERROR_WKS_CANT_SERVE_CLIENT (0xC00E0066) when dependent-client-only methods are received. This applies to Windows NT Workstation, Windows 2000 Workstation, Windows XP, and Windows Vista.

By default, Windows Server 2008 and Windows Server 2003 reject calls from dependent clients, although an administrator can enable support. By default, Windows 2000 Servers accept calls from dependent clients, although an administrator can disable support. Windows NT Servers do not restrict calls from dependent clients. If a dependent-client-specific method is invoked against a Windows Server that is configured to reject calls from dependent clients, the server takes no action and returns RPC_S_ACCESS_DENIED (0x00000005).

The following methods are called only by dependent clients, and are therefore wholly subject to the above restrictions:

Opnum	Name
6	**R_QMCreateObjectInternal**
7	**R_QMSetObjectSecurityInternal**
8	**R_QMGetObjectSecurityInternal**
9	**R_QMDeleteObject**
10	**R_QMGetObjectProperties**
11	**R_QMSetObjectProperties**
12	**R_QMObjectPathToObjectFormat**
14	**R_QMGetTmWhereabouts**
15	**R_QMEnlistTransaction**
16	**R_QMEnlistInternalTransaction**
17	**R_QMCommitTransaction**
18	**R_QMAbortTransaction**
19	**rpc_QMOpenQueueInternal**
20	**rpc_ACCloseHandle**
22	**rpc_ACCloseCursor**
23	**rpc_ACSetCursorProperties**
26	**rpc_ACHandleToFormatName**
27	**rpc_ACPurgeQueue**
28	**R_QMQueryQMRegistryInternal**

Additionally, all methods of qmcomm2 are subject to the above restrictions.

Name	Value
RPC_S_ACCESS_DENIED	0x00000005
MQ_ERROR_WKS_CANT_SERVE_CLIENT	0xC00E0066

<73> Section 3.1.5.2: Message bodies MUST NOT be stored encrypted when messages reach their destination queues. Windows NT and Windows 2000 servers only perform message body decryption in the **QMSendMessageInternalEx** method. If a message with an encrypted body is sent directly to a target queue via the rpc_ACSendMessageEx method, Windows NT and Windows 2000 servers return STATUS_RETRY (0xc000022d) to indicate that the client MUST call **QMSendMessageInternalEx** instead.

<74> Section 3.1.5.2: The recommended default hash algorithm value used by Windows Vista and Windows Server 2008 is MQMSG_CALG_SHA1 (0x00008004). Windows NT, Windows 2000, and Windows Server 2003 servers use MQMSG_CALG_MD5 (0x00008003) as the default value.

<75> Section 3.1.5.2: The recommended default encryption algorithm value used by Windows Vista and Windows Server 2008 is MQMSG_CALG_RC4(0x00006801). Windows NT, Windows 2000, and Windows Server 2003 servers use MQMSG_CALG_RC2 (0x00006602) as the default value.

<76> Section 3.1.5.2: The ptb.old.pulAuthProvNameLenProp field is only ignored on input to send operations on Windows NT. The field has meaning, as specified in section 3.1.5.2, on Windows 2000, Windows XP, Windows Server 2003, and Windows Vista and Windows Server 2008.

<77> Section 3.1.5.2: Windows applications typically invoke **rpc_ACSendMessageEx** indirectly via the Windows API function MQSendMessage. The Windows API documentation for MQSendMessage includes the following error codes. For descriptions of the following error code name/value pairs, see [MS-MQMQ] section 2.7 and [MSDN-MQEIC], in that order.

Name	Value
MQ_ERROR_ACCESS_DENIED	0xc00e0025
MQ_ERROR_BAD_SECURITY_CONTEXT	0xc00e0035
MQ_ERROR_CERTIFICATE_NOT_PROVIDED	0xc00e006d
MQ_ERROR_CORRUPTED_INTERNAL_CERTIFICATE	0xc00e002d
MQ_ERROR_CORRUPTED_PERSONAL_CERT_STORE	0xc00e0031
MQ_ERROR_CORRUPTED_SECURITY_DATA	0xc00e0030
MQ_ERROR_COULD_NOT_GET_USER_SID	0xc00e0036
MQ_ERROR_DTC_CONNECT	0xc00e004c
MQ_ERROR_ILLEGAL_FORMATNAME	0xc00e001e
MQ_ERROR_INSUFFICIENT_RESOURCES	0xc00e0027
MQ_ERROR_INVALID_CERTIFICATE	0xc00e002c
MQ_ERROR_INVALID_HANDLE	0xc00e0007
MQ_ERROR_INVALID_PARAMETER	0xc00e0006
MQ_ERROR_MESSAGE_STORAGE_FAILED	0xc00e002a
MQ_ERROR_NO_INTERNAL_USER_CERT	0xc00e002f
MQ_ERROR_PROPERTY	x0c00e0002
MQ_ERROR_SERVICE_NOT_AVAILABLE	0xc00e000b
MQ_ERROR_STALE_HANDLE	0xc00e0056
MQ_ERROR_TRANSACTION_USAGE	0xc00e0050
MQ_ERROR_TRANSACTION_ENLIST	0xc00e0058

Name	Value
MQ_ERROR_UNSUPPORTED_FORMATNAME_OPERATION	0xc00e0020
MQ_INFORMATION_PROPERTY	0x400e0001

<78> Section 3.1.5.2: Workstation editions of Windows operating systems do not act as supporting servers for dependent clients, and return MQ_ERROR_WKS_CANT_SERVE_CLIENT (0xC00E0066) when dependent-client-only methods are received. This applies to Windows NT Workstation, Windows 2000 Workstation, Windows XP, and Windows Vista.

By default, Windows Server 2008 and Windows Server 2003 reject calls from dependent clients, although an administrator can enable support. By default, Windows 2000 Servers accept calls from dependent clients, although an administrator can disable support. Windows NT Servers do not restrict calls from dependent clients. If a dependent-client-specific method is invoked against a Windows Server that is configured to reject calls from dependent clients, the server takes no action and returns RPC_S_ACCESS_DENIED (0x00000005).

The following methods are called only by dependent clients, and are therefore wholly subject to the above restrictions:

Opnum	Name
6	**R_QMCreateObjectInternal**
7	**R_QMSetObjectSecurityInternal**
8	**R_QMGetObjectSecurityInternal**
9	**R_QMDeleteObject**
10	**R_QMGetObjectProperties**
11	**R_QMSetObjectProperties**
12	**R_QMObjectPathToObjectFormat**
14	**R_QMGetTmWhereabouts**
15	**R_QMEnlistTransaction**
16	**R_QMEnlistInternalTransaction**
17	**R_QMCommitTransaction**
18	**R_QMAbortTransaction**
19	**rpc_QMOpenQueueInternal**
20	**rpc_ACCloseHandle**
22	**rpc_ACCloseCursor**
23	**rpc_ACSetCursorProperties**
26	**rpc_ACHandleToFormatName**
27	**rpc_ACPurgeQueue**

Opnum	Name
28	**R_QMQueryQMRegistryInternal**

Additionally, all methods of qmcomm2 are subject to the above restrictions.

Name	Value
RPC_S_ACCESS_DENIED	0x00000005
MQ_ERROR_WKS_CANT_SERVE_CLIENT	0xC00E0066

<79> Section 3.1.5.3: Windows applications typically invoke **rpc_ACReceiveMessageEx** indirectly via the Windows API function MQReceiveMessage. The Windows API documentation for MQReceiveMessage includes the following error codes. For descriptions of the following error code name/value pairs, see [MS-MQMQ] section 2.7 and [MSDN-MQEIC], in that order.

Name	Value
MQ_ERROR_ACCESS_DENIED	0xc00e0025
MQ_ERROR_BUFFER_OVERFLOW	0xc00e001a
MQ_ERROR_DTC_CONNECT	0xc00e004c
MQ_ERROR_FORMATNAME_BUFFER_TOO_SMALL	0xc00e001f
MQ_ERROR_ILLEGAL_CURSOR_ACTION	0xc00e001c
MQ_ERROR_INSUFFICIENT_PROPERTIES	0xc00e003f
MQ_ERROR_INVALID_HANDLE	0xc00e0007
MQ_ERROR_IO_TIMEOUT	0xc00e001b
MQ_ERROR_LABEL_BUFFER_TOO_SMALL	0xc00e005e
MQ_ERROR_MESSAGE_ALREADY_RECEIVED	0xc00e001d
MQ_ERROR_OPERATION_CANCELLED	0xc00e0008
MQ_ERROR_PROV_NAME_BUFFER_TOO_SMALL	0xc00e0063
MQ_ERROR_PROPERTY	x0c00e0002
MQ_ERROR_QUEUE_DELETED	0xc00e005a
MQ_ERROR_SENDER_CERT_BUFFER_TOO_SMALL	0xc00e002b
MQ_ERROR_SENDERID_BUFFER_TOO_SMALL	0xc00e0022
MQ_ERROR_SERVICE_NOT_AVAILABLE	0xc00e000b
MQ_ERROR_SIGNATURE_BUFFER_TOO_SMALL	0xc00e0062
MQ_ERROR_STALE_HANDLE	0xc00e0056
MQ_ERROR_SYMM_KEY_BUFFER_TOO_SMALL	0xc00e0061

Name	Value
MQ_ERROR_TRANSACTION_USAGE	0xc00e0050
MQ_INFORMATION_OPERATION_PENDING	0x400e0006
MQ_INFORMATION_PROPERTY	0x400e0001

<80> Section 3.1.5.3: Workstation editions of Windows operating systems do not act as supporting servers for dependent clients, and return MQ_ERROR_WKS_CANT_SERVE_CLIENT (0xC00E0066) when dependent-client-only methods are received. This applies to Windows NT Workstation, Windows 2000 Workstation, Windows XP, and Windows Vista.

By default, Windows Server 2008 and Windows Server 2003 reject calls from dependent clients, although an administrator can enable support. By default, Windows 2000 Servers accept calls from dependent clients, although an administrator can disable support. Windows NT Servers do not restrict calls from dependent clients. If a dependent-client-specific method is invoked against a Windows Server that is configured to reject calls from dependent clients, the server takes no action and returns RPC_S_ACCESS_DENIED (0x00000005).

The following methods are called only by dependent clients, and are therefore wholly subject to the above restrictions:

Opnum	Name
6	**R_QMCreateObjectInternal**
7	**R_QMSetObjectSecurityInternal**
8	**R_QMGetObjectSecurityInternal**
9	**R_QMDeleteObject**
10	**R_QMGetObjectProperties**
11	**R_QMSetObjectProperties**
12	**R_QMObjectPathToObjectFormat**
14	**R_QMGetTmWhereabouts**
15	**R_QMEnlistTransaction**
16	**R_QMEnlistInternalTransaction**
17	**R_QMCommitTransaction**
18	**R_QMAbortTransaction**
19	**rpc_QMOpenQueueInternal**
20	**rpc_ACCloseHandle**
22	**rpc_ACCloseCursor**
23	**rpc_ACSetCursorProperties**
26	**rpc_ACHandleToFormatName**

Opnum	Name
27	**rpc_ACPurgeQueue**
28	**R_QMQueryQMRegistryInternal**

Additionally, all methods of qmcomm2 are subject to the above restrictions.

Name	Value
RPC_S_ACCESS_DENIED	0x00000005
MQ_ERROR_WKS_CANT_SERVE_CLIENT	0xC00E0066

<81> Section 3.1.5.4: Windows applications typically invoke **rpc_ACCreateCursorEx** indirectly via the Windows API function MQCreateCursor. The Windows API documentation for MQCreateCursor includes the following error codes. For descriptions of the following error code name/value pairs, see [MS-MQMQ] section 2.7 and [MSDN-MQEIC], in that order.

Name	Value
MQ_ERROR_INVALID_HANDLE	0xc00e0007
MQ_ERROR_INSUFFICIENT_RESOURCES	0xc00e0027
MQ_ERROR_REMOTE_MACHINE_NOT_AVAILABLE	0xc00e0069
MQ_ERROR_STALE_HANDLE	0xc00e0056

<82> Section 3.1.5.4: Workstation editions of Windows operating systems do not act as supporting servers for dependent clients, and return MQ_ERROR_WKS_CANT_SERVE_CLIENT (0xC00E0066) when dependent-client-only methods are received. This applies to Windows NT Workstation, Windows 2000 Workstation, Windows XP, and Windows Vista.

By default, Windows Server 2008 and Windows Server 2003 reject calls from dependent clients, although an administrator can enable support. By default, Windows 2000 Servers accept calls from dependent clients, although an administrator can disable support. Windows NT Servers do not restrict calls from dependent clients. If a dependent-client-specific method is invoked against a Windows Server that is configured to reject calls from dependent clients, the server takes no action and returns RPC_S_ACCESS_DENIED (0x00000005).

The following methods are called only by dependent clients, and are therefore wholly subject to the above restrictions:

Opnum	Name
6	**R_QMCreateObjectInternal**
7	**R_QMSetObjectSecurityInternal**
8	**R_QMGetObjectSecurityInternal**
9	**R_QMDeleteObject**
10	**R_QMGetObjectProperties**

Opnum	Name
11	**R_QMSetObjectProperties**
12	**R_QMObjectPathToObjectFormat**
14	**R_QMGetTmWhereabouts**
15	**R_QMEnlistTransaction**
16	**R_QMEnlistInternalTransaction**
17	**R_QMCommitTransaction**
18	**R_QMAbortTransaction**
19	**rpc_QMOpenQueueInternal**
20	**rpc_ACCloseHandle**
22	**rpc_ACCloseCursor**
23	**rpc_ACSetCursorProperties**
26	**rpc_ACHandleToFormatName**
27	**rpc_ACPurgeQueue**
28	**R_QMQueryQMRegistryInternal**

Additionally, all methods of qmcomm2 are subject to the above restrictions.

Name	Value
RPC_S_ACCESS_DENIED	0x00000005
MQ_ERROR_WKS_CANT_SERVE_CLIENT	0xC00E0066

<83> Section 3.1.5.4: The described behavior is for Windows Server 2003, Windows Vista, and Windows Server 2008. Windows NT, Windows 2000, Windows XP servers behave as follows: If the given queue handle represents a queue which is NOT local (remote) to the supporting server, the server MUST create a cursor object and return a handle to it via the CreateCursor.hCursor member of ptbl. Additionally, the server must also set the srv_hACQueue member of CreateCursor to the remote queue handle associated with the queue handle provided for the *hQueue* method parameter. The remote queue handle is originally acquired from the *pphContext* out-parameter of the **R_QMOpenRemoteQueue** method, which must be invoked at the appropriate remote queue manager server. Refer to sections 3.1.4.18 and 3.1.4.2 for details. The server MUST also set the **cli_pQMQueue** member of CreateCursor to the local queue context value associated with the queue handle provided for the *hQueue* method parameter. The local queue context value is originally acquired from the *pdwQMContext* out-parameter of the rpc_QMOpenQueueInternal method. Refer to section 3.1.4.17 for details on the rpc_QMOpenQueueInternal method.

The server then MUST return MQ_INFORMATION_REMOTE_OPERATION (0x400e03e8). This specific return code instructs the client that it MUST contact a remote queue manager to create a remote cursor handle via **R_QMCreateRemoteCursor**, and MUST associate the result with the local cursor handle via **rpc_ACSetCursorProperties**. The caller MUST complete these operations successfully prior to using the cursor handle returned by this method.

<84> Section 4.4: Windows NT and Windows 2000 servers predate the existence of the RemoteRead protocol, and therefore do not perform this step.

<85> Section 4.5: All Windowsclients produce new XACTUOW values by calling the Windows RPC function UuidCreate.

<86> Section 5.1: Workstation editions of Windows operating systems do not act as supporting servers for dependent clients, and return MQ_ERROR_WKS_CANT_SERVE_CLIENT (0xC00E0066) when dependent-client-only methods are received. This applies to Windows NT Workstation, Windows 2000 Workstation, Windows XP, and Windows Vista.

By default, Windows Server 2008 and Windows Server 2003 reject calls from dependent clients, although an administrator can enable support. By default, Windows 2000 Servers accept calls from dependent clients, although an administrator can disable support. Windows NT Servers do not restrict calls from dependent clients. If a dependent-client-specific method is invoked against a Windows Server that is configured to reject calls from dependent clients, the server takes no action and returns RPC_S_ACCESS_DENIED (0x00000005).

The following methods are called only by dependent clients, and are therefore wholly subject to the above restrictions:

Opnum	Name
6	**R_QMCreateObjectInternal**
7	**R_QMSetObjectSecurityInternal**
8	**R_QMGetObjectSecurityInternal**
9	**R_QMDeleteObject**
10	**R_QMGetObjectProperties**
11	**R_QMSetObjectProperties**
12	**R_QMObjectPathToObjectFormat**
14	**R_QMGetTmWhereabouts**
15	**R_QMEnlistTransaction**
16	**R_QMEnlistInternalTransaction**
17	**R_QMCommitTransaction**
18	**R_QMAbortTransaction**
19	**rpc_QMOpenQueueInternal**
20	**rpc_ACCloseHandle**
22	**rpc_ACCloseCursor**
23	**rpc_ACSetCursorProperties**
26	**rpc_ACHandleToFormatName**
27	**rpc_ACPurgeQueue**

Opnum	Name
28	R_QMQueryQMRegistryInternal

Additionally, all methods of qmcomm2 are subject to the above restrictions.

8 Index