[MC-IISIAQ]:
Internet Information Services (IIS) IAQ AdminRPC Protocol Specification

Revision Summary

Date	Revision History	Revision Class	Comments
08/10/2007	0.1	Major	Initial Availability
09/28/2007	0.2	Minor	Updated the technical content.
10/23/2007	0.3	Minor	Updated the technical content.
11/30/2007	0.3.1	Editorial	Revised and edited the technical content.

Date	Revision History	Revision Class	Comments
01/25/2008	0.3.2	Editorial	Revised and edited the technical content.

Table of Contents

1 Introduction

This document specifies the Internet Information Services (IIS) IAQ AdminRPC Protocol, for querying and managing a **Simple Mail Transfer Protocol (SMTP)** virtual **server** with advanced queuing.

1.1 Glossary

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

Client
Globally Unique Identifier (GUID)
Interface Definition Language (IDL)
Microsoft Interface Definition Language (MIDL)
Network Data Representation (NDR)
Opnum
Remote Procedure Call (RPC)
Simple Mail Transfer Protocol (SMTP)
Server
Unicode (UCS-2)
Universally Unique Identifier (UUID)

The following terms are specific to this document:

Cursor: A remembered position in a sequence or list, at which activity takes place.

DS: An Active Directory service or **server**.

Freeze: An administrative action to halt the operation of a **link**, **queue**, or message.

Link: A collection of **queues**, the messages within which have all been assigned to the same **next hop**.

Next Hop: A determination of the next relay action to be taken in routing the message.

Next Hop Server: The **server** targeted by the **next hop**.

Queue: A collection that holds all messages bound for the same delivery point.

Proxy Address: The unique designation of a mail recipient in the format required by the messaging protocol. Thus, a user may have one or more X.400 **proxy addresses**, one or more XMTP **proxy addresses**, and one or more Lotus **proxy addresses**.

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as specified 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-DCOM] Microsoft Corporation, "Distributed Component Object Model (DCOM) Remote Protocol Specification", March 2007.

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

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

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

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

1.2.2 Informative References

None.

1.3 Protocol Overview (Synopsis)

This protocol defines an interface for modeling and managing Simple Mail Transfer Protocol (SMTP) servers that implement advanced queuing.

The Internet Information Services (IIS) IAQ AdminRPC Protocol is designed to perform **queue** management operations. It is also used to administer messages in the queues and **links**. (A link is a group of queues that are destined for the same **next hop server**.)

Through the IIS IAQ AdminRPC Protocol, the user can select the virtual server, enumerate the queues, enumerate all messages in the queues, and perform such actions as **freeze**, unfreeze, delete, and retrieve the content stream of the messages.

The IIS IAQ AdminRPC Protocol can also be used to enumerate all the links in the virtual server, and to perform start and stop actions on all active outbound links. It is also used to get the current state of the links.

1.4 Relationship to Other Protocols

The IIS IAQ AdminRPC Protocol uses **RPC**, as specified in [MS-RPCE], for underlying network authentication, event notifications, data transfer, and state management. The IIS IAQ AdminRPC Protocol is built on top of the Distributed Component Object Model (DCOM) Remote Protocol ([MS-DCOM]).

1.5 Prerequisites/Preconditions

The IIS IAQ AdminRPC Protocol depends on the RPC protocol layer for transport. It assumes the presence of an API with underlying RPC functionality.

1.6 Applicability Statement

This protocol is most appropriate for managing an advanced SMTP server implemented according to the Microsoft Exchange model, with virtual servers, queues, links and messages.

1.7 Versioning and Capability Negotiation

This protocol contains versioning of some of the structures to enable independent evolution of **client** and server protocol engines. The underlying RPC may have versioning/negotiation, but that aspect must be transparent to the users of this interface.

1.8 Vendor-Extensible Fields

This protocol uses HRESULT values as defined in [MS-ERREF] section 2.1. Vendors can define their own HRESULT values, provided they set the C bit (0x20000000) for each vendor-defined value, indicating that the value is a customer code.

This protocol uses Win32 error codes. These values are taken from the Windows error number space defined in [MS-ERREF] section 2.2. Vendors SHOULD reuse those values with their indicated meanings. Choosing any other value runs the risk of a collision in the future.

1.9 Standards Assignments

No standards bodies have assigned a designation to this protocol.

2 Messages

The following sections specify how Internet Information Services (IIS) IAQ AdminRPC Protocol messages are transported and common IIS IAQ AdminRPC Protocol data types.

2.1 Transport

All IIS IAQ AdminRPC Protocol messages are transported via the Distributed Component Object Model (DCOM) Remote Protocol, as specified in [MS-DCOM].

2.2 Common Data Types

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

This protocol also uses the following types, as specified in [MS-DTYP].

Type	Reference
DWORD	As specified in [MS-DTYP] section **2.2.7**
GUID	As specified in [MS-DTYP] section **2.3.2**
HRESULT	As specified in [MS-DTYP] section **2.2.17**
LPCWSTR	As specified in [MS-DTYP] section **2.2.32**
LPWSTR	As specified in [MS-DTYP] section **2.2.35**
SYSTEMTIME	As specified in [MS-DTYP] section **2.3.6**
ULARGE_INTEGER	As specified in [MS-DTYP] section **2.3.8**
ULONG	As specified in [MS-DTYP] section **2.2.51**
WCHAR	As specified in [MS-DTYP] section **2.2.58**

2.2.1 QUEUE_ADMIN_VERSIONS

The **QUEUE_ADMIN_VERSIONS** enumeration specifies current and supported queue administrative versions.

```
typedef  enum tagQUEUE_ADMIN_VERSIONS
{
  CURRENT_QUEUE_ADMIN_VERSION = 4
} QUEUE_ADMIN_VERSIONS;
```

CURRENT_QUEUE_ADMIN_VERSION: The current queue administrative version. All structures should have this value in their **dwVersion** fields.

2.2.2 MESSAGE_FILTER

The **MESSAGE_FILTER** structure specifies criteria for the selection of messages based on the values in the filter.

```
typedef struct tagMESSAGE_FILTER {
  DWORD dwVersion;
  DWORD fFlags;
  [string] LPCWSTR szMessageId;
  [string] LPCWSTR szMessageSender;
  [string] LPCWSTR szMessageRecipient;
  DWORD dwLargerThanSize;
  SYSTEMTIME stOlderThan;
} MESSAGE_FILTER,
 *PMESSAGE_FILTER;
```

dwVersion: Version of the **MESSAGE_FILTER** structure. It MUST be CURRENT_QUEUE_ADMIN_VERSION.

fFlags: MUST be a valid selection of flags defined by the **MESSAGE_FILTER_FLAGS** enumeration, indicating which fields of the filter are specified.

szMessageId: MUST be a null-terminated **Unicode (UCS-2)** string specifying a Message ID.

szMessageSender: MUST be a null-terminated ASCII string specifying the message sender to be matched.

szMessageRecipient: MUST be a null-terminated ASCII string specifying the message recipient to be matched.

dwLargerThanSize: Integer specifying the minimum size for messages selected.

stOlderThan: **SYSTEMTIME** structure indicating the minimum age for messages selected.

2.2.3 MESSAGE_FILTER_FLAGS

The **MESSAGE_FILTER_FLAGS** enumeration specifies bitflags that indicate which fields are in use in the **MESSAGE_FILTER** structure. More than one flag may be used in a value.

```
typedef  enum tagMESSAGE_FILTER_FLAGS
{
  MF_MESSAGEID = 0x00000001,
  MF_SENDER = 0x00000002,
  MF_RECIPIENT = 0x00000004,
  MF_SIZE = 0x00000008,
  MF_TIME = 0x00000010,
  MF_FROZEN = 0x00000020,
  MF_FAILED = 0x00000100,
  MF_ALL = 0x40000000,
  MF_INVERTSENSE = 0x80000000
} MESSAGE_FILTER_FLAGS;
```

MF_MESSAGEID: The MESSAGE_FILTER.szMessageId field is specified.

MF_SENDER: The MESSAGE_FILTER.szMessageSender field is specified.

MF_RECIPIENT: The MESSAGE_FILTER.szMessageRecipient field is specified.

MF_SIZE: The MESSAGE_FILTER.dwLargerThanSize field is specified.

MF_TIME: The MESSAGE_FILTER.stOlderThan field is specified.

MF_FROZEN: The MESSAGE_FILTER.fFrozen field is specified.

MF_FAILED: Selects messages that have had at least one failed delivery attempt.

MF_ALL: Selects all messages.

MF_INVERTSENSE: If set, indicates the complement of the filter definition.

2.2.4 MESSAGE_ACTION

The **MESSAGE_ACTION** enumeration specifies possible administrative actions that may be applied to messages in a virtual server, link, or queue.

```
typedef  enum tagMESSAGE_ACTION
{
  MA_THAW_GLOBAL = 0x00000001,
  MA_COUNT = 0x00000002,
  MA_FREEZE_GLOBAL = 0x00000004,
  MA_DELETE = 0x00000008,
  MA_DELETE_SILENT = 0x00000010
} MESSAGE_ACTION;
```

MA_THAW_GLOBAL: Unfreeze the message in the virtual server, link, or queue.

MA_COUNT: Null operation; does not affect messages, but does return count.

MA_FREEZE_GLOBAL: Freeze the message in the virtual server, link, or queue.

MA_DELETE: Remove the message from the virtual server, link, or queue.

MA_DELETE_SILENT: Remove the message without generating a **Network Data Representation (NDR)**.

In certain operations, these flags may be combined.

2.2.5 MESSAGE_ENUM_FILTER_TYPE

The **MESSAGE_ENUM_FILTER_TYPE** enumeration specifies the type of filter requested. These are bitflags and can be OR'd together.

```
typedef  enum tagMESSAGE_ENUM_FILTER_TYPE
{
  MEF_FIRST_N_MESSAGES = 0x00000001,
  MEF_SENDER = 0x00000002,
  MEF_RECIPIENT = 0x00000004,
  MEF_LARGER_THAN = 0x00000008,
  MEF_OLDER_THAN = 0x00000010,
  MEF_FROZEN = 0x00000020,
  MEF_N_LARGEST_MESSAGES = 0x00000040,
  MEF_N_OLDEST_MESSAGES = 0x00000080,
  MEF_FAILED = 0x00000100,
  MEF_ALL = 0x40000000,
  MEF_INVERTSENSE = 0x80000000
```

```
} MESSAGE_ENUM_FILTER_TYPE;
```

MEF_FIRST_N_MESSAGES: Return the first MESSAGE_ENUM_FILTER.cMessages messages.

MEF_SENDER: The MESSAGE_ENUM_FILTER.szMessageSender field is specified.

MEF_RECIPIENT: The MESSAGE_ENUM_FILTER.szMessageRecipient field is specified.

MEF_LARGER_THAN: Return messages larger than the number of bytes given in the MESSAGE_ENUM_FILTER.cbSize field.

MEF_OLDER_THAN: Return messages older than the value given in the MESSAGE_ENUM_FILTER.stDate field.

MEF_FROZEN: Return messages that are frozen.

MEF_N_LARGEST_MESSAGES: Return the largest messages in the MESSAGE_ENUM_FILTER.cMessages field.

MEF_N_OLDEST_MESSAGES: Return the oldest messages in the MESSAGE_ENUM_FILTER.cMessages field.

MEF_FAILED: Return only messages that have had failed delivery attempts.

MEF_ALL: Select all messages.

MEF_INVERTSENSE: Invert the meaning of the filter.

2.2.6 MESSAGE_ENUM_FILTER

The **MESSAGE_ENUM_FILTER** structure defines criteria for enumerating messages.

```
typedef struct tagMESSAGE_ENUM_FILTER {
  DWORD dwVersion;
  DWORD mefType;
  DWORD cMessages;
  DWORD cbSize;
  DWORD cSkipMessages;
  SYSTEMTIME stDate;
  [string] LPCWSTR szMessageSender;
  [string] LPCWSTR szMessageRecipient;
} MESSAGE_ENUM_FILTER,
 *PMESSAGE_ENUM_FILTER;
```

dwVersion: The version of the filter. It MUST be CURRENT_QUEUE_ADMIN_VERSION.

mefType: The **MESSAGE_ENUM_FILTER_TYPE** flags for the filter.

cMessages: The number of messages to return.

cbSize: The size parameter of messages.

cSkipMessages: The number of messages at the front of the queue to skip. This is provided to allow "paged" queries to the server.

stDate: The date/time parameter of messages.

szMessageSender: Messages sent by this sender match.

szMessageRecipient: Messages sent to this recipient match.

2.2.7 LINK_INFO_FLAGS

The **LINK_INFO_FLAGS** enumeration specifies the state of the link.

```
typedef  enum tagLINK_INFO_FLAGS
{
  LI_ACTIVE = 0x00000001,
  LI_READY = 0x00000002,
  LI_RETRY = 0x00000004,
  LI_SCHEDULED = 0x00000008,
  LI_REMOTE = 0x00000010,
  LI_FROZEN = 0x00000020,
  LI_TYPE_REMOTE_DELIVERY = 0x00000100,
  LI_TYPE_LOCAL_DELIVERY = 0x00000200,
  LI_TYPE_PENDING_ROUTING = 0x00000400,
  LI_TYPE_PENDING_CAT = 0x00000800,
  LI_TYPE_CURRENTLY_UNREACHABLE = 0x00001000,
  LI_TYPE_DEFERRED_DELIVERY = 0x00002000,
  LI_TYPE_INTERNAL = 0x00004000,
  LI_TYPE_PENDING_SUBMIT = 0x00008000
} LINK_INFO_FLAGS;
```

LI_ACTIVE: Link has an active connection that is transferring mail.

LI_READY: Link is ready for a connection, but there are no connections.

LI_RETRY: Link is waiting for the retry interval to elapse.

LI_SCHEDULED: Link is waiting for the next scheduled time.

LI_REMOTE: Link is to be activated by remote server. A connection will not be made unless requested by a remote server.

LI_FROZEN: Link was frozen by administrative action.

LI_TYPE_REMOTE_DELIVERY: Messages on this link are being delivered remotely. This is the default type of link.

LI_TYPE_LOCAL_DELIVERY: Messages on this link are being delivered locally.

LI_TYPE_PENDING_ROUTING: Messages on this link have not been routed to their **next hop**.

LI_TYPE_PENDING_CAT: Messages on this link are pending message categorization.

LI_TYPE_CURRENTLY_UNREACHABLE: Messages on this link do not have an available route to their final destination. This is due to transient network or server errors. These messages will be retried when a route becomes available.

LI_TYPE_DEFERRED_DELIVERY: All messages in this link are delayed by originator request until a specified time.

LI_TYPE_INTERNAL: This link is an internal link not described by the above flags.

LI_TYPE_PENDING_SUBMIT: Messages in this link have not yet been through categorization. This is the default queue that all messages pass through prior to sorting.

2.2.8 LINK_ACTION

The **LINK_ACTION** enumeration specifies possible administrative action that may be applied to links in a virtual server.

```
typedef  enum tagLINK_ACTION
{
  LA_INTERNAL = 0x00000000,
  LA_KICK = 0x00000001,
  LA_FREEZE = 0x00000020,
  LA_THAW = 0x00000040
} LINK_ACTION;
```

LA_INTERNAL: For internal use only.

LA_KICK: Force a connection to be made for this link. This will even work for connections pending retry, or a scheduled connection.

LA_FREEZE: Prohibit the link from creating outbound connections.

LA_THAW: Remove the prohibition from creating outbound connections.

2.2.9 LINK_INFO

The **LINK_INFO** structure describes the state of the link in a virtual server.

```
typedef struct tagLINK_INFO {
  DWORD dwVersion;
  [string] LPWSTR szLinkName;
  DWORD cMessages;
  DWORD fStateFlags;
  SYSTEMTIME stNextScheduledConnection;
  SYSTEMTIME stOldestMessage;
  ULARGE_INTEGER cbLinkVolume;
  [string] LPWSTR szLinkDN;
  [string] LPWSTR szExtendedStateInfo;
  DWORD dwSupportedLinkActions;
} LINK_INFO,
 *PLINK_INFO;
```

dwVersion: Version of the **LINK_INFO** structure. It MUST be CURRENT_QUEUE_ADMIN_VERSION.

szLinkName: A unique null-terminated Unicode (UCS-2) string that is the name of the link, which might be the name of the next hop.

cMessages: The number of messages queued up in this link.

fStateFlags: MUST be values from the **LINK_INFO_FLAGS** enumeration indicating the link state.

stNextScheduledConnection: The time at which the next connection to the next hop will be attempted.

stOldestMessage: The time of the oldest message on this link.

cbLinkVolume: The total number of bytes in all messages in this link.

szLinkDN: A null-terminated Unicode (UCS-2) string containing the domain name associated with this link. It MAY be NULL, indicating the local domain.

szExtendedStateInfo: A null-terminated Unicode (UCS-2) string that provides additional information about the link state. For example, this member may contain an explanation when the link is in a retry state. It MAY be NULL.

dwSupportedLinkActions: An unsigned integer that gives a list of actions supported on this link. Possible values are defined in **LINK_ACTION**.

2.2.10 AQ_MESSAGE_FLAGS

The **AQ_MESSAGE_FLAGS** enumeration describes message properties.

```
typedef  enum tagAQ_MESSAGE_FLAGS
{
  MP_HIGH = 0x00000001,
  MP_NORMAL = 0x00000002,
  MP_LOW = 0x00000004,
  MP_MSG_FROZEN = 0x00000008,
  MP_MSG_RETRY = 0x00000010,
  MP_MSG_CONTENT_AVAILABLE = 0x00000020
} AQ_MESSAGE_FLAGS;
```

MP_HIGH: The message property High Priority has been requested by the originator.

MP_NORMAL: The message property Normal Priority has been requested by the originator.

MP_LOW: The message property Low Priority has been requested by the originator.

MP_MSG_FROZEN: The message has been frozen by the administrator.

MP_MSG_RETRY: Delivery of this message has been attempted and failed at least once.

MP_MSG_CONTENT_AVAILABLE: The content for this message can be accessed through the QAPI.

2.2.11 QUEUE_INFO

The **QUEUE_INFO** structure describes the state of the queue in a link for a virtual server.

```
typedef struct tagQUEUE_INFO {
  DWORD dwVersion;
  [string] LPWSTR szQueueName;
  [string] LPWSTR szLinkName;
  DWORD cMessages;
```

```
  ULARGE_INTEGER cbQueueVolume;
  DWORD dwMsgEnumFlagsSupported;
} QUEUE_INFO,
 *PQUEUE_INFO;
```

dwVersion: Version of the **QUEUE_INFO** structure. It MUST be CURRENT_QUEUE_ADMIN_VERSION.

szQueueName: A null-terminated Unicode (UCS-2) string that is the name of a queue.

szLinkName: A null-terminated Unicode (UCS-2) string that is the name of a link that is servicing this queue.

cMessages: Number of messages on this queue.

cbQueueVolume: Total number of bytes for all messages in the queue.

dwMsgEnumFlagsSupported: The types of message enumeration supported.

2.2.12 MESSAGE_INFO

The **MESSAGE_INFO** structure defines the details of a single mail message.

```
typedef struct tagMESSAGE_INFO {
  DWORD dwVersion;
  [string] LPWSTR szMessageId;
  [string] LPWSTR szSender;
  [string] LPWSTR szSubject;
  DWORD cRecipients;
  [string] LPWSTR szRecipients;
  DWORD cCCRecipients;
  [string] LPWSTR szCCRecipients;
  DWORD cBCCRecipients;
  [string] LPWSTR szBCCRecipients;
  DWORD fMsgFlags;
  DWORD cbMessageSize;
  SYSTEMTIME stSubmission;
  SYSTEMTIME stReceived;
  SYSTEMTIME stExpiry;
  DWORD cFailures;
  DWORD cEnvRecipients;
  DWORD cbEnvRecipients;
  [sizeis(cbEnvRecipients/sizeof(WCHAR))]
    WCHAR* mszEnvRecipients;
} MESSAGE_INFO,
 *PMESSAGE_INFO;
```

dwVersion: Version of the **MESSAGE_INFO** structure. It MUST be CURRENT_QUEUE_ADMIN_VERSION.

szMessageId: A null-terminated Unicode (UCS-2) string that is the message ID.

szSender: A null-terminated Unicode (UCS-2) string that is the sender address, from the "From:" header.

szSubject: A null-terminated Unicode (UCS-2) string that is the message subject.

cRecipients: The number of recipients named in the "To:" header.

szRecipients: A null-terminated Unicode (UCS-2) string that contains the recipient addresses from the "To:" header.

cCCRecipients: The number of recipients in the "CC:" header.

szCCRecipients: A null-terminated Unicode (UCS-2) string that contains the CC recipient addresses from the "CC:" header.

cBCCRecipients: The number of recipients in the "BCC:" header.

szBCCRecipients: A null-terminated Unicode (UCS-2) string that contains the BCC recipient addresses, from the "BCC:" header.

fMsgFlags: An **AQ_MESSAGE_FLAGS** enumeration describing the message properties.

cbMessageSize: The size of the message, in bytes.

stSubmission: The time the message was submitted.

stReceived: The time the message was received by this server.

stExpiry: The time the message will expire if not delivered to all recipients.

cFailures: The number of failed delivery attempts associated with the message.

cEnvRecipients: The number of envelope recipients. Envelope recipients are those to whom delivery will be attempted. Other copies of the message may reside in other queues or servers.

cbEnvRecipients: The size, in bytes, of the envelope recipients.

mszEnvRecipients: A Unicode (UCS-2) string containing the list of envelope recipients, separated by NULL characters. The buffer itself is terminated by an additional NULL character. Each recipient string will be formatted in the **proxy address** format of 'addr-type ":" address'. The addr-type should match the address type found in the **DS** (that is, SMTP). The address should be in its native format.

2.2.13 QUEUELINK_TYPE

The **QUEUELINK_TYPE** enumeration specifies the valid types for a **QUEUELINK_ID** structure.

```
typedef  enum tagQUEUELINK_TYPE
{
  QLT_QUEUE,
  QLT_LINK,
  QLT_NONE
} QUEUELINK_TYPE;
```

QLT_QUEUE: This value indicates that the structure is a queue.

QLT_LINK: This value indicates that the structure is a link.

QLT_NONE: This value indicates that the structure is neither a link nor a queue.

2.2.14 QUEUELINK_ID

The **QUEUELINK_ID** structure defines how a QUEUELINK structure is identified.

```
typedef struct tagQUEUELINK_ID {
  GUID uuid;
  [string] LPWSTR szName;
  DWORD dwId;
  QUEUELINK_TYPE qltType;
} QUEUELINK_ID;
```

uuid: A unique **GUID** assigned to the structure.

szName: A null-terminated Unicode (UCS-2) string that is a unique name assigned to the structure.

dwId: A unique integer assigned to the structure.

qltType: The **QUEUELINK_TYPE** appropriate to the structure.

3 Protocol Details

The following sections specify details of the IIS IAQ AdminRPC Protocol, including abstract data models, message processing events, and sequencing rules.

The client side of this protocol is not concerned with authentication, states, or events. All of these are handled by lower level providers.

3.1 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.

The following information model is presumed by this protocol:

The smallest atomic object is a message. Messages may be individually viewed, deleted, or otherwise manipulated. Creation and alteration of messages is not supported.

Messages bound for the same destination (as determined by the domain part of the recipient addresses) are gathered into "queues."

Two or more queues bound for ultimately different destinations may have the same next "hop" (relay destination). The server gathers into groups the queues with the same next hop in preparation for physical transfer. Each such group is called a "link."

Queues, links, and message transfers are managed by one or more virtual servers. The virtual servers are independent, named entities within the physical server. A single physical server contains one or more virtual servers.

Thus, the hierarchy of objects is accessible through a succession of interfaces as listed below. Each interface serves as a container and provides a method for listing and finding its contained objects.

- A physical server contains one or more virtual servers.
- A virtual server contains zero or more links.
- A link contains one or more queues.
- A queue contains zero or more messages.
- A message has content and other attributes.

3.1.2 Timers

There are no timers specific to this protocol.

3.1.3 Initialization

There is no protocol-specific initialization.

3.1.4 Message Processing Events and Sequencing Rules

3.1.4.1 IAQAdmin

The **IAQAdmin** interface provides one method for a client to access Advanced Queue admin objects.

This interface inherits from IUnknown, as specified in [MS-DCOM] section **3.2.1.5.8**. The version for this interface is 0.0.

The server MUST implement a DCOM Remote Protocol object class with the CLSID CLSID_AQAdmin (83866cad-740d-11d2-94e4-00c04fa379f1) using the **UUID** {ba7af300-7373-11d2-94e4-00c04fa379f1} for this interface. This interface includes the following methods beyond those of IUnknown.

Methods in RPC Opnum Order

Method	Description
GetVirtualServerAdminITF	Gets an administrative interface to an SMTP or message transfer agent (MTA) virtual server's queues. Opnum: 3

All methods MUST NOT throw exceptions.

3.1.4.1.1 GetVirtualServerAdminITF (Opnum 3)

The **GetVirtualServerAdminITF** method is called by a client to get an administrative interface to an SMTP or MTA virtual server's queues.

```
HRESULT GetVirtualServerAdminITF(
  [in] LPCWSTR wszComputer,
  [in] LPCWSTR wszVirtualServer,
  [out] IVSAQAdmin** ppivsaqadmin
);
```

wszComputer: A null-terminated Unicode (UCS-2) string that is the hostname of the physical computer where the virtual server is running. If this is NULL, the call is directed to a local physical computer.

wszVirtualServer: A null-terminated Unicode (UCS-2) numeric string identifying the requested virtual server. It MUST NOT be NULL.

ppivsaqadmin: The value assigned to the address of the requested virtual server advanced queue admin interface.

Return Values: This method MUST return S_OK (0x00000000) on success, and a failure result (as specified in [MS-ERREF], section 2) on failure. All failure results MUST be treated identically. The following table specifies failure results specific to this method.

Return value/code	Description
0x00000000 S_OK	Success.

Return value/code	Description
0x8007000E E_OUTOFMEMORY	Insufficient resources.
0x80000005 E_POINTER	Null pointer parameter.
0x80070057 E_INVALIDARG	Invalid parameter.
0x80004005 E_ACCESSDENIED	Logged-on principal is not authorized to view queues on the specified virtual server.
0x80071722 HRESULT_FROM_WIN32(RPC_S_SERVER_UNAVAILABLE)	Unable to connect to the specified virtual server.

When this method is invoked, the server MUST attempt to return a valid **IVSAQAdmin** interface pointer to the client, and return S_OK upon success. Otherwise, this method MUST return an appropriate error code.

3.1.4.2 IVSAQAdmin

The **IVSAQAdmin** interface provides methods to access the list of links on a virtual server.

This interface inherits from IUnknown, as specified in [MS-DCOM] section **3.2.1.5.8**. The UUID of the RPC interface for **IVSAQAdmin** is {e2ed3340-1e96-11d3-bfcc-00c04fa3490a}. The version for this interface is 0.0.

This interface includes the following methods beyond those of IUnknown.

Methods in RPC Opnum Order

Method	Description
GetLinkEnum	Gets an enumerator for all the links on this virtual server. Opnum: 3
StopAllLinks	Stops activities on the links of a virtual server. Opnum: 4
StartAllLinks	Starts all activities on the links of a virtual server. Opnum: 5
GetGlobalLinkState	Checks the global state of the links as set by StopAllLinks/StartAllLinks. Opnum: 6

All methods MUST NOT throw exceptions.

3.1.4.2.1 GetLinkEnum (Opnum 3)

The **GetLinkEnum** method is called by a client to get an enumerator to all the links within the virtual server.

```
HRESULT GetLinkEnum(
  [out] IEnumVSAQLinks** ppenum
);
```

ppenum: Upon success, this pointer MUST be the valid address of an **IEnumVSAQLinks** interface.

Return Values: This method MUST return S_OK (0x00000000) on success, and a failure result (as specified in [MS-ERREF], section 2) on failure. All failure results MUST be treated identically. The following table specifies failure results specific to this method.

Return value/code	Description
0x00000000 S_OK	Success.
0x8007000E E_OUTOFMEMORY	Insufficient resources.
0x80000005 E_POINTER	Null pointer parameter.
0x80070057 E_INVALIDARG	Invalid parameter.
0x80004005 E_ACCESSDENIED	Logged-on principal is not authorized to view queues on the specified virtual server.
0x80071722 HRESULT_FROM_WIN32(RPC_S_SERVER_UNAVAILABLE)	Unable to connect to the specified virtual server.

When this method is invoked, the server MUST attempt to return a valid **IEnumVSAQLinks** enumerator interface to the client. Upon success, the method returns S_OK. Otherwise, it returns the appropriate error code.

3.1.4.2.2 StopAllLinks (Opnum 4)

The **StopAllLinks** method is called by a client to stop all activities on the links of a virtual server.

```
HRESULT StopAllLinks();
```

This method has no parameters.

Return Values: This method MUST return S_OK (0x00000000) on success, and a failure result (as specified in [MS-ERREF], section 2) on failure. All failure results MUST be treated identically. The following table specifies failure results specific to this method.

Return value/code	Description
0x00000000 S_OK	Success.
0x80071722	Unable to connect to the specified

Return value/code	Description
HRESULT_FROM_WIN32(RPC_S_SERVER_UNAVAILABLE)	virtual server.

When this method is invoked, the server MUST attempt to stop all the links in the virtual server. Upon success, the method returns S_OK. Otherwise, it returns the appropriate error code.

3.1.4.2.3 StartAllLinks (Opnum 5)

The **StartAllLinks** method is called by a client to start all activities on the links of a virtual server.

```
HRESULT StartAllLinks();
```

This method has no parameters.

Return Values: This method MUST return S_OK (0x00000000) on success, and a failure result (as specified in [MS-ERREF], section 2) on failure. All failure results MUST be treated identically. The following table specifies failure results specific to this method.

Return value/code	Description
0x00000000 S_OK	Success.
0x80071722 HRESULT_FROM_WIN32(RPC_S_SERVER_UNAVAILABLE)	Unable to connect to the specified virtual server.

When this method is invoked, the server MUST attempt to start all the links in the virtual server. Upon success, the method returns S_OK. Otherwise, it returns the appropriate error code.

3.1.4.2.4 GetGlobalLinkState (Opnum 6)

The **GetGlobalLinkState** method is called by a client to check the global state of the links as set by **StopAllLinks** or **StartAllLinks**.

```
HRESULT GetGlobalLinkState();
```

This method has no parameters.

Return Values: This method MUST return S_OK (0x00000000) on success, and a failure result (as specified in [MS-ERREF], section 2) on failure. All failure results MUST be treated identically. The following table specifies failure results specific to this method.

Return value/code	Description
0x00000000 S_OK	Success.
0x00000001 S_FALSE	Links have been stopped (by a previous call to **StopAllLinks**).
0x80004001 E_NOTIMPL	Method not implemented.

Return value/code	Description
0x80071722 HRESULT_FROM_WIN32(RPC_S_SERVER_UNAVAILABLE)	Unable to connect to the specified virtual server.

When this method is invoked, the server MUST attempt to return the global state of the links, which has been set by the **StartAllLinks** and **StopAllLinks** functions. Upon success, the method returns S_OK. Otherwise, it returns the appropriate error code.

3.1.4.3 IEnumVSAQLinks

The **IEnumVSAQLinks** interface provides access to individual links or groups of links through a **cursor** acquired through the **IVSAQAdmin** interface.

This interface inherits from IUnknown, as specified in [MS-DCOM] section **3.2.1.5.8**. The UUID of the RPC interface for **IEnumVSAQLinks** is {ba7af300-7373-11d2-94e4-00c04fa379f1}. The version for this interface is 0.0.

This interface includes the following methods beyond those of IUnknown.

Methods in RPC Opnum Order

Method	Description
Next	Gets the requested number of links following the cursor position in the list of links. Opnum: 3
Skip	Moves the cursor through the list of **IVSAQLink** links. Opnum: 4
Reset	Resets the cursor to the first position in the list of **IVSAQLink** links. Opnum: 5
Clone	Creates a copy of the **IEnumVSAQLinks** interface. Opnum: 6

All methods MUST NOT throw exceptions.

3.1.4.3.1 Next (Opnum 3)

The **Next** method is called by a client to return the requested number of links following the cursor position in the list of links.

```
HRESULT Next(
  [in] ULONG cElt,
  [out] IVSAQLink** rgelt,
  [out] ULONG* pcFetched
);
```

cElt: The number of **IVSAQLink** references requested.

rgelt: The array in which the requested **IVSAQLink** references will be returned. The array MUST be at least as large as the number of objects requested.

pcFetched: The number of references actually returned.

Return Values: This method MUST return S_OK (0x00000000) on success, and a failure result (as specified in [MS-ERREF], section 2) on failure. All failure results MUST be treated identically. The following table specifies failure results specific to this method.

Return value/code	Description
0x00000000 S_OK	The requested number of references were retrieved.
0x00000001 S_FALSE	Fewer than the requested number of references were retrieved.

When this method is invoked, the server MUST attempt to return an array of valid **IVSAQLink** interface pointers less than or equal to the number of references requested, indicating the successful number. If less than the number requested, the server must indicate the difference with S_FALSE.

3.1.4.3.2 Skip (Opnum 4)

The **Skip** method is called by a client to move the cursor through the list of **IVSAQLink** links.

```
HRESULT Skip(
  [in] ULONG cElt
);
```

cElt: The number of **IVSAQLink** references through which the cursor is to be moved.

Return Values: This method MUST return S_OK (0x00000000) on success, and a failure result (as specified in [MS-ERREF], section 2) on failure. All failure results MUST be treated identically. The following table specifies failure results specific to this method.

Return value/code	Description
0x00000000 S_OK	The cursor was moved the requested number of positions.
0x00000001 S_FALSE	The list contained an insufficient number of links to move the cursor the requested number of positions. The cursor will be left at the last link in the list.

When this method is invoked, the server MUST attempt to move the cursor through the list the requested number of **IVSAQLink** references. If the list contains an insufficient number of links to accomplish the requested move, the server MUST set the cursor at the last link in the list.

3.1.4.3.3 Reset (Opnum 5)

The **Reset** method is called by a client to reset the cursor to the first position in the list of **IVSAQLink** links.

```
HRESULT Reset();
```

This method has no parameters.

Return Values: This method MUST return S_OK (0x00000000) on success, and a failure result (as specified in [MS-ERREF], section 2) on failure. All failure results MUST be treated identically. The following table specifies failure results specific to this method.

Return value/code	Description
0x00000000 S_OK	The cursor was reset to the beginning of the list.

When this method is invoked, the server MUST attempt to reset the cursor position at the beginning of the list.

3.1.4.3.4 Clone (Opnum 6)

The **Clone** method is called by a client to return a copy of the **IEnumVSAQLinks** interface.

```
HRESULT Clone(
  [out] IEnumVSAQLinks** ppenum
);
```

ppenum: The address of a reference to **IEnumVSAQLinks** into which will be loaded the reference to the cloned interface.

Return Values: This method MUST return S_OK (0x00000000) on success, and a failure result (as specified in [MS-ERREF], section 2) on failure. All failure results MUST be treated identically. The following table specifies failure results specific to this method.

Return value/code	Description
0x00000000 S_OK	Success.
0x8007000E E_OUTOFMEMORY	Insufficient resources.
0x80000005 E_POINTER	Null pointer parameter.

When this method is invoked, the server MUST attempt to create a copy of the **IEnumVSAQLinks** interface enumerator. On failure, the server MUST return the appropriate error code.

3.1.4.4 IVSAQLink

The **IVSAQLink** interface provides access to a single virtual server AQ Link. A link represents a connectable entity. Another way to think of a link is that it represents the next hop to which mail messages on this link will be delivered.

This interface inherits from IUnknown, as specified in [MS-DCOM]. The UUID of the RPC interface for the **IVSAQLink** is {3f962f94-1ecd-11d3-bfcc-00c04fa3490a}. The version for this interface is 0.0.

This interface includes the following methods beyond those of IUnknown.

Methods in RPC Opnum Order

Method	Description
GetInfo	Returns information about the link. Opnum: 3
SetLinkState	Changes the state of a link. Opnum: 4
GetQueueEnum	Provides an enumerator object for iterating through the queues associated with the link. Opnum: 5

All methods MUST NOT throw exceptions.

3.1.4.4.1 GetInfo (Opnum 3)

The **GetInfo** method is called by a client to get information about the link.

```
HRESULT GetInfo(
  [in, out] PLINK_INFO pli
);
```

pli: MUST be a non-null pointer to the **LINK_INFO** structure with valid values as specified in section 2.2.9.

Return Values: This method MUST return S_OK (0x00000000) on success, and a failure result (as specified in [MS-ERREF], section 2) on failure. All failure results MUST be treated identically. The following table specifies failure results specific to this method.

Return value/code	Description
0x00000000 S_OK	Success.
0x8007000E E_OUTOFMEMORY	Insufficient resources.
0x80000005 E_POINTER	Null pointer parameter.
0x80070057 E_INVALIDARG	Invalid parameter.
0x80071722 HRESULT_FROM_WIN32(RPC_S_SERVER_UNAVAILABLE)	Unable to connect to the specified virtual server.

When this method is invoked, the server MUST attempt to fill the **LINK_INFO** structure with the link information. On failure, the server MUST return the appropriate error code.

3.1.4.4.2 SetLinkState (Opnum 4)

The **SetLinkState** method is called by a client to change the state of the link.

```
HRESULT SetLinkState(
```

```
  [in] LINK_ACTION la
);
```

la: Specifies the action to apply to the link. It MUST be one of the values defined in **LINK_ACTION**.

Return Values: This method MUST return S_OK (0x00000000) on success, and a failure result (as specified in [MS-ERREF], section 2) on failure. All failure results MUST be treated identically. The following table specifies failure results specific to this method.

Return value/code	Description
0x00000000 S_OK	Success.
0x8007000E E_OUTOFMEMORY	Insufficient resources.
0x80000005 E_POINTER	Null pointer parameter.
0x80070057 E_INVALIDARG	Invalid parameter.
0x80071722 HRESULT_FROM_WIN32(RPC_S_SERVER_UNAVAILABLE)	Unable to connect to the specified virtual server.

When this method is invoked, the server MUST attempt to accomplish the action requested by the *la* parameter. On failure, the server MUST return the appropriate error code.

3.1.4.4.3 GetQueueEnum (Opnum 5)

The **GetQueueEnum** method is called by a client to get an enumerator object for iterating through the queues associated with the link.

```
HRESULT GetQueueEnum(
  [out] IEnumLinkQueues** ppenum
);
```

ppenum: MUST be the valid address of a **IEnumLinkQueues** pointer.

Return Values: This method MUST return S_OK (0x00000000) on success, and a failure result (as specified in [MS-ERREF], section 2) on failure. All failure results MUST be treated identically. The following table specifies failure results specific to this method.

Return value/code	Description
0x00000000 S_OK	Success.
0x8007000E E_OUTOFMEMORY	Insufficient resources.
0x80000005	Null pointer parameter.

Return value/code	Description
E_POINTER	
0x80070057 E_INVALIDARG	Invalid parameter.
0x80071722 HRESULT_FROM_WIN32(RPC_S_SERVER_UNAVAILABLE)	Unable to connect to the specified virtual server.

When this method is invoked, the server MUST attempt to return a valid **IEnumLinkQueues** enumerator interface. On failure, the server MUST return the appropriate error code.

3.1.4.5 IEnumLinkQueues

The **IEnumLinkQueues** interface provides access to individual links and groups of links.

This interface inherits from IUnknown, as specified in [MS-DCOM] section **3.2.1.5.8**. The UUID of the RPC interface for the **IEnumLinkQueues** is {ba7af303-7373-11d2-94e4-00c04fa379f1}. The version for this interface is 0.0.

This interface includes the following methods beyond those of IUnknown.

Methods in RPC Opnum Order

Method	Description
Next	Gets the sequentially consecutive specified number of **ILinkQueue** objects. Opnum: 3
Skip	Moves the cursor through the list of **ILinkQueue** references. Opnum: 4
Reset	Resets the cursor to the first position in the list of **ILinkQueue** references. Opnum: 5
Clone	Creates a copy of the **IEnumLinkQueues** interface. Opnum: 6

All methods MUST NOT throw exceptions.

3.1.4.5.1 Next (Opnum 3)

The **Next** method is called by a client to get the sequentially consecutive specified number of **ILinkQueue** objects.

```
HRESULT Next(
  [in] ULONG cElt,
  [out] ILinkQueue** rgelt,
  [out] ULONG* pcFetched
);
```

cElt: The number of **ILinkQueue** references requested.

rgelt: The array where the requested **ILinkQueue** references will be returned. The array MUST be at least as large as the number of references requested.

pcFetched: The number of references actually returned.

Return Values: This method MUST return S_OK (0x00000000) on success, and a failure result (as specified in [MS-ERREF], section 2) on failure. All failure results MUST be treated identically. The following table specifies failure results specific to this method.

Return value/code	Description
0x00000000 S_OK	The requested number of references were retrieved.
0x00000001 S_FALSE	Fewer than the requested number of references were retrieved.

When this method is invoked, the server MUST attempt to return an array of **ILinkQueue** interface pointers equal to or less than the number requested. Otherwise, the server will return the number of available interface pointers.

3.1.4.5.2 Skip (Opnum 4)

The **Skip** method is called by a client to move a cursor through the list of **ILinkQueue** references.

```
HRESULT Skip(
  [in] ULONG cElt
);
```

cElt: The number of **ILinkQueue** references through which the cursor is to be moved.

Return Values: This method MUST return S_OK (0x00000000) on success, and a failure result (as specified in [MS-ERREF], section 2) on failure. All failure results MUST be treated identically. The following table specifies failure results specific to this method.

Return value/code	Description
0x00000000 S_OK	The cursor was moved the requested number of positions.
0x00000001 S_FALSE	The list contained an insufficient number of objects. The cursor will be left at the last object in the list.

When this method is invoked, the server MUST attempt to move the cursor through the list the specified number of **ILinkQueue** references requested. If the available number of references in the list is less than the number requested, the server MUST set the cursor at the last in the list.

3.1.4.5.3 Reset (Opnum 5)

The **Reset** method is called by a client to reset the cursor to the first position in the list of **ILinkQueue** references.

```
HRESULT Reset();
```

This method has no parameters.

Return Values: This method MUST return S_OK (0x00000000) on success, and a failure result (as specified in [MS-ERREF], section 2) on failure. All failure results MUST be treated identically. The following table specifies failure results specific to this method.

Return value/code	Description
0x00000000 S_OK	The cursor was reset to the beginning of the list.

When this method is invoked, the server MUST attempt to reset the cursor to the first position in the list of **ILinkQueue** references.

3.1.4.5.4 Clone (Opnum 6)

The **Clone** method is called by a client to get a copy of the **IEnumLinkQueues** interface.

```
HRESULT Clone(
  [out] IEnumLinkQueues** ppenum
);
```

ppenum: The address of a reference to **IEnumLinkQueues** into which will be loaded the reference to the cloned interface.

Return Values: This method MUST return S_OK (0x00000000) on success, and a failure result (as specified in [MS-ERREF], section 2) on failure. All failure results MUST be treated identically. The following table specifies failure results specific to this method.

Return value/code	Description
0x00000000 S_OK	Success.
0x8007000E E_OUTOFMEMORY	Insufficient resources.
0x80000005 E_POINTER	Null pointer parameter.

When this method is invoked, the server MUST attempt to return the copy of the **IEnumLinkQueues** enumerator.

3.1.4.6 ILinkQueue

The **ILinkQueue** interface provides access to individual queues in the link. A queue contains a list of messages bound for the same destination. All messages in a queue have the same SMTP domain part.

This interface inherits from IUnknown, as specified in [MS-DCOM] section **3.2.1.5.8**. The UUID of the RPC interface for the **ILinkQueue** is {ff9a1bb6-1e96-11d3-bfcc-00c04fa3490a}. The version for this interface is 0.0.

This interface includes the following methods beyond those of IUnknown.

Methods in RPC Opnum Order

Method	Description
GetInfo	Returns information about the queue. Opnum: 3
GetMessageEnum	Provides an enumerator object for iterating through the messages associated with the queue. Opnum: 4

All methods MUST NOT throw exceptions.

3.1.4.6.1 GetInfo (Opnum 3)

The **GetInfo** method is called by a client to get information about the queue.

```
HRESULT GetInfo(
  [in, out] PQUEUE_INFO pqi
);
```

pqi: MUST be a non-null pointer to the **QUEUE_INFO** structure with valid values as specified in section 2.2.11.

Return Values: This method MUST return S_OK (0x00000000) on success, and a failure result (as specified in [MS-ERREF], section 2) on failure. All failure results MUST be treated identically. The following table specifies failure results specific to this method.

Return value/code	Description
0x00000000 S_OK	Success.
0x8007000E E_OUTOFMEMORY	Insufficient resources.
0x80000005 E_POINTER	Null pointer parameter.
0x80070057 E_INVALIDARG	Invalid parameter.
0x80071722 HRESULT_FROM_WIN32(RPC_S_SERVER_UNAVAILABLE)	Unable to connect to the specified virtual server.

When this method is invoked, the server MUST attempt to fill the user's **QUEUE_INFO** structure with valid values.

3.1.4.6.2 GetMessageEnum (Opnum 4)

The **GetMessageEnum** method is called by a client to get an enumerator object for iterating through the messages associated with the queue.

```
HRESULT GetMessageEnum(
  [in] PMESSAGE_ENUM_FILTER pFilter,
  [out] IAQEnumMessages** ppenum
);
```

pFilter: MUST be a valid filter specification as defined in **MESSAGE_ENUM_FILTER (section 2.2.6)**.

ppenum: MUST be the valid address of a **IAQEnumMessages** pointer.

Return Values: This method MUST return S_OK (0x00000000) on success, and a failure result (as specified in [MS-ERREF], section 2) on failure. All failure results MUST be treated identically. The following table specifies failure results specific to this method.

Return value/code	Description
0x00000000 S_OK	Success.
0x8007000E E_OUTOFMEMORY	Insufficient resources.
0x80000005 E_POINTER	Null pointer parameter.
0x80070057 E_INVALIDARG	Invalid parameter.
0x80071722 HRESULT_FROM_WIN32(RPC_S_SERVER_UNAVAILABLE)	Unable to connect to the specified virtual server.

When this method is invoked, the server MUST attempt to return a message enumerator interface **IAQEnumMessages**.

3.1.4.7 IAQEnumMessages

The **IAQEnumMessages** interface provides access to individual messages and groups of messages in a queue.

This interface inherits from IUnknown, as specified in [MS-DCOM] section **3.2.1.5.8**. The UUID of the RPC interface for **IAQEnumMessages** is {ba7af302-7373-11d2-94e4-00c04fa379f1}. The version for this interface is 0.0.

This interface includes the following methods beyond those of IUnknown.

Methods in RPC Opnum Order

Method	Description
Next	Gets the sequentially consecutive specified number of **IAQMessage** objects.

Method	Description
	Opnum: 3
Skip	Moves the cursor through the list of **IAQMessage** references. Opnum: 4
Reset	Resets the cursor to the first position in the list of **IAQMessage** references. Opnum: 5
Clone	Creates a copy of this enumerator object. Opnum: 6

All methods MUST NOT throw exceptions.

3.1.4.7.1 Next (Opnum 3)

The **Next** method is called by a client to get the sequentially consecutive specified number of **IAQMessage** objects.

```
HRESULT Next(
  [in] ULONG cElt,
  [out] IAQMessage** rgelt,
  [out] ULONG* pcFetched
);
```

cElt: The number of **IAQMessage** references requested.

rgelt: The array in which the requested **IAQMessage** references will be returned. The array MUST be at least as large as the number of references requested.

pcFetched: The number of references actually returned.

Return Values: This method MUST return S_OK (0x00000000) on success, and a failure result (as specified in [MS-ERREF], section 2) on failure. All failure results MUST be treated identically. The following table specifies failure results specific to this method.

Return value/code	Description
0x00000000 S_OK	The requested number of references were retrieved.
0x00000001 S_FALSE	Fewer than the requested number of references were retrieved.

When this method is invoked, the server MUST attempt to return an array of **IAQMessage** interface pointers less than or equal to the number requested. The returned array will have the specified number of interface pointers. Otherwise, the server will return the maximum number available with the error code S_FALSE.

3.1.4.7.2 Skip (Opnum 4)

The **Skip** method is called by a client to move the cursor through the list of **IAQMessage** references.

```
HRESULT Skip(
  [in] ULONG cElt
);
```

cElt: The number of **IAQMessage** references through which the cursor is to be moved.

Return Values: This method MUST return S_OK (0x00000000) on success, and a failure result (as specified in [MS-ERREF], section 2) on failure. All failure results MUST be treated identically. The following table specifies failure results specific to this method.

Return value/code	Description
0x00000000 S_OK	The cursor was moved the requested number of positions.
0x00000001 S_FALSE	The list contained an insufficient number of objects. The cursor will be positioned at the last reference in the list.

When this method is invoked, the server MUST attempt to move the cursor the specified number of **IAQMessage** references in the list. If the number requested is greater than the number available, the cursor MUST be moved to the end of the list, and S_FALSE returned to the user.

3.1.4.7.3 Reset (Opnum 5)

The **Reset** method is called by a client to reset to the first position in the list of **IAQMessage** references.

```
HRESULT Reset();
```

This method has no parameters.

Return Values: This method MUST return S_OK (0x00000000) on success, and a failure result (as specified in [MS-ERREF], section 2) on failure. All failure results MUST be treated identically. The following table specifies failure results specific to this method.

Return value/code	Description
0x00000000 S_OK	The cursor was reset to the beginning of the list.

When this method is invoked, the server MUST reset the cursor to the first position in the list.

3.1.4.7.4 Clone (Opnum 6)

The **Clone** method is called by a client to create a clone of the enumerator object.

```
HRESULT Clone(
  [out] IAQEnumMessages** ppenum
);
```

ppenum: The address of a reference to **IAQEnumMessages** into which will be loaded the reference to the cloned interface.

Return Values: This method MUST return S_OK (0x00000000) on success, and a failure result (as specified in [MS-ERREF], section 2) on failure. All failure results MUST be treated identically. The following table specifies failure results specific to this method.

Return value/code	Description
0x00000000 S_OK	Success.
0x8007000E E_OUTOFMEMORY	Insufficient resources.
0x80000005 E_POINTER	Null pointer parameter.

When this method is invoked, the server MUST attempt to return a copy of the **IAQEnumMessages** enumerator. Otherwise, the server MUST return the appropriate error code.

3.1.4.8 IAQMessage

The **IAQMessage** interface provides methods for a client to access a single message.

This interface inherits from IUnknown, as specified in [MS-DCOM] section **3.2.1.5.8**. The UUID of the RPC interface for **IAQMessage** is {ba7af305-7373-11d2-94e4-00c04fa379f1}. The version for this interface is 0.0.

This interface includes the following methods beyond those of IUnknown.

Methods in RPC Opnum Order

Method	Description
GetInfo	Gets information about the queue. Opnum: 3
GetContentStream	Gets a stream for the message content. Opnum: 4

All methods MUST NOT throw exceptions.

3.1.4.8.1 GetInfo (Opnum 3)

The **GetInfo** method is called by a client to get information about a message.

```
HRESULT GetInfo(
  [in, out] PMESSAGE_INFO pmi
);
```

pmi: Must be a non-null pointer to the **MESSAGE_INFO** structure with valid values as specified in section 2.2.12.

Return Values: This method MUST return S_OK (0x00000000) on success, and a failure result (as specified in [MS-ERREF] section 2) on failure. All failure results MUST be treated identically. The following table specifies failure results specific to this method.

Return value/code	Description
0x00000000 S_OK	Success.
0x80004001 E_NOTIMPL	Method not implemented.
0x8007000E E_OUTOFMEMORY	Insufficient resources.
0x80000005 E_POINTER	Null pointer parameter.
0x80070057 E_INVALIDARG	Invalid parameter.
0x80071722 HRESULT_FROM_WIN32(RPC_S_SERVER_UNAVAILABLE)	Unable to connect to the specified virtual server.

When this method is invoked, the server MUST attempt to return a valid pointer to the **MESSAGE_INFO** structure. Otherwise, the server MUST return the appropriate error code.

3.1.4.8.2 GetContentStream (Opnum 4)

The **GetContentStream** method is called by a client to get the content of a message.

```
HRESULT GetContentStream(
  [out] IStream** ppIStream,
  [out] LPWSTR* pwszContentType
);
```

ppIStream: MUST be the address of an actual **IStream** pointer.

pwszContentType: MUST be the address of an actual Unicode (UCS-2) string pointer that will return a string describing the content.

Return Values: This method MUST return S_OK (0x00000000) on success, and a failure result (as specified in [MS-ERREF] section 2) on failure. All failure results MUST be treated identically. The following table specifies failure results specific to this method.

Return value/code	Description
0x00000000 S_OK	Success.
0x80004001 E_NOTIMPL	Method not implemented.
0x8007000E E_OUTOFMEMORY	Insufficient resources.

Return value/code	Description
0x80000005 E_POINTER	Null pointer parameter.
0x80070057 E_INVALIDARG	Invalid parameter.
0x80071722 HRESULT_FROM_WIN32(RPC_S_SERVER_UNAVAILABLE)	Unable to connect to the specified virtual server.

When this method is invoked, the server MUST attempt to return a valid pointer to the **IStream** interface to retrieve message content.

3.1.4.9 IAQMessageAction

The **IAQMessageAction** interface is used to apply specified actions to messages on this virtual server that match the specified message filter criteria. The actions will only be applied to messages that belong to the object implementing this interface. For example, only messages that match the filter on a given queue will be affected if this method is called on an **ILinkQueue** object.

This interface inherits from IUnknown, as specified in [MS-DCOM] section **3.2.1.5.8**. The UUID of the RPC interface for **IAQMessageAction** is {1eb44a71-1e95-11d3-bfcc-00c04fa3490a}. The version for this interface is 0.0.

This interface includes the following methods beyond those of IUnknown.

Methods in RPC Opnum Order

Method	Description
ApplyActionToMessages	Apply the specified action to messages on this virtual server that match the specified message filter criteria. Opnum: 3
QuerySupportedActions	Describes which actions/filters of the **ApplyActionToMessages** method are supported. Opnum: 4

All methods MUST NOT throw exceptions.

3.1.4.9.1 ApplyActionToMessages (Opnum 3)

The **ApplyActionToMessages** method is called by a client to apply specified actions to messages on this virtual server that match the specified message filter criteria. The actions will only be applied to messages that belong to the object that implements this interface.

For example, only messages that match the filter on a given queue will be affected if this method is called on an **ILinkQueue** object.

```
HRESULT ApplyActionToMessages(
  [in] PMESSAGE_FILTER Filter,
  [in] MESSAGE_ACTION Action,
  [out] DWORD* pcMsgs
```

```
);
```

Filter: The criteria for selection of messages formatted according to the **MESSAGE_FILTER** structure definition. The argument MUST be a non-null pointer to such a structure. The filter MAY specify formatting that does not match existing messages.

Action: The action to be applied to the messages selected by the filter. The argument MUST be a value defined by the **MESSAGE_ACTION** enumeration.

pcMsgs: This argument MUST be a non-null pointer to a **DWORD**. On execution, the method returns via this argument the number of messages to which the action was applied.

Return Values: This method MUST return S_OK (0x00000000) on success, and a failure result (as specified in [MS-ERREF] section 2) on failure. All failure results MUST be treated identically. The following table specifies failure results specific to this method.

Return value/code	Description
0x00000000 S_OK	Success.
0x8007000E E_OUTOFMEMORY	Insufficient resources.
0x80000005 E_POINTER	Null pointer parameter.
0x80070057 E_INVALIDARG	Invalid parameter.
0x80004005 E_ACCESSDENIED	Logged-on principal is not authorized to view queues on the specified virtual server.
0x80071722 HRESULT_FROM_WIN32(RPC_S_SERVER_UNAVAILABLE)	Unable to connect to the specified virtual server.

When this method is invoked, the server MUST attempt to return a **DWORD** value indicating the number of messages to which the action was applied. Upon method or action failure, the server MUST return the appropriate error code.

3.1.4.9.2 QuerySupportedActions (Opnum 4)

The **QuerySupportedActions** method is called by a client to discover the list of supported actions and filters.

```
HRESULT QuerySupportedActions(
  [out] DWORD* pdwSupportedActions,
  [out] DWORD* pdwSupportedFilterFlags
);
```

pdwSupportedActions: MUST be a non-null pointer to a **DWORD** that returns a bitfield indicating the supported actions from values specified in **MESSAGE_ACTION**.

pdwSupportedFilterFlags: MUST be a non-null pointer to a **DWORD** that returns a bitfield indicating the supported filters from values specified in **MESSAGE_FILTER_FLAGS**.

Return Values: This method MUST return S_OK (0x00000000) on success, and a failure result (as specified in [MS-ERREF] section 2) on failure. All failure results MUST be treated identically. The following table specifies failure results specific to this method.

Return value/code	Description
0x00000000 S_OK	Success.
0x8007000E E_OUTOFMEMORY	Insufficient resources.
0x80000005 E_POINTER	Null pointer parameter.
0x80070057 E_INVALIDARG	Invalid parameter.
0x80004005 E_ACCESSDENIED	Logged-on principal is not authorized to view queues on the specified virtual server.
0x80071722 HRESULT_FROM_WIN32(RPC_S_SERVER_UNAVAILABLE)	Unable to connect to the specified virtual server.

When this method is invoked, the server MUST attempt to return a bitfield indicating the supported filters from values defined in **MESSAGE_FILTER_FLAGS**. On failure, the server MUST return the appropriate error code.

3.1.4.10 IUniqueId

The **IUniqueId** interface provides methods for a client to get the canonical name of a queue or link.

This interface inherits from IUnknown, as specified in [MS-DCOM] section **3.2.1.5.8**. The UUID of the RPC interface for **IUniqueId** is {EA4DFDF2-9E87-4c57-B845-123872C5649F}. The version for this interface is 0.0.

This interface includes the following methods beyond those of IUnknown.

Methods in RPC Opnum Order

Method	Description
GetUniqueId	Return **QUEUELINK_ID** for the queue or link. Opnum: 3

All methods MUST NOT throw exceptions.

3.1.4.10.1 GetUniqueId (Opnum 3)

The **GetUniqueId** method is called by a client to get a **QUEUELINK_ID** for this queue or link. This is the canonical representation of the queue. The QAPI guarantees that there is no more than one

queue or link with the same **QUEUELINK_ID** at a given point in time. The memory associated with the out parameter is guaranteed good until the underlying QAPI object is released.

```
HRESULT GetUniqueId(
  [out] QUEUELINK_ID** ppqlid
);
```

ppqlid: MUST be a valid address to the **QUEUELINK_ID** pointer.

Return Values: This method MUST return S_OK (0x00000000) on success, and a failure result (as specified in [MS-ERREF] section 2) on failure. All failure results MUST be treated identically. The following table specifies failure results specific to this method.

Return value/code	Description
0x00000000 S_OK	Success.
0x80000005 E_POINTER	Null pointer parameter.
0x80004001 E_NOTIMPL	Method not implemented.
0x80071722 HRESULT_FROM_WIN32(RPC_S_SERVER_UNAVAILABLE)	Unable to connect to the specified virtual server.

When this method is invoked, the server MUST attempt to return a valid pointer to a **QUEUELINK_ID** structure. Otherwise, the server MUST return the appropriate error code.

3.1.5 Timer Events

None.

3.1.6 Other Local Events

None.

3.2 Client Details

3.2.1 Abstract Data Model

None.

3.2.2 Timers

None.

3.2.3 Initialization

None.

3.2.4 Message Processing Events and Sequencing Rules

None.

3.2.5 Timer Events

None.

3.2.6 Other Local Events

None.

4 Protocol Examples

The hierarchy of objects is accessible through a succession of interfaces.

The **IAQAdmin** interface provides a method called **GetVirtualServerAdminITF** for a server to send the client an interface (**IVSAQAdmin**) to the SMTP virtual server advance queue.

The **IVSAQAdmin** interface provides a method called **GetLinkEnum** for a server to send an enumerator interface (**IEnumVSAQLinks**) that can be used to get all the links in this virtual server.

This interface also provides a method called **StartAllLinks** for a server to start all the links in the SMTP virtual server.

This interface also provides a method called **StopAllLinks** for a server to stop all the links in the SMTP virtual server.

This interface also provides a method called **GetGlobalLinkState** for a server to return the global state of the links.

The **IEnumVSAQLinks** interface that is returned provides a method to enumerate references to all the links in the virtual server. References to the links will be returned through the **IVSAQLink** interface.

The **IVSAQLink** interface provides a method called **GetInfo** to get information about each link.

This **IVSAQLink** interface also provides a method called **SetLinkState** to set the state of each link.

This **IVSAQLink** interface also provides a method called **GetQueueEnum** to retrieve an enumerator interface (**IEnumLinkQueues**) to queues in a link.

The **IEnumLinkQueues** interface provides methods to enumerate or retrieve queues in a link. The reference to the queue in this link will be returned through the **ILinkQueue** interface.

The **ILinkQueue** interface provides methods to get information about queues and the message enumerator interface (**IAQEnumMessages**).

The **IAQEnumMessages** interface provides methods to iterate over messages in a queue. This interface also provides a reference to a message through the **IAQMessage** interface.

The **IAQMessage** interface provides methods to retrieve information about messages. This interface also provides a method called **GetContentStream** to retrieve the contents of a message through an **IStream** pointer.

The following example shows a client application for getting a reference to the virtual server advance queue admin interface (**IVSAQAdmin**), and starting all the links in the virtual server. This example also shows how to enumerate all the links in the virtual server.

```
int __cdecl main(int argc, char **argv)
{
    HRESULT hr = S_OK;
    IAQAdmin *pIAQAdmin = NULL;
    IVSAQAdmin *pIVSAQAdmin = NULL;
    IEnumVSAQLinks *pIEnumVSAQLinks = NULL;
    IVSAQLink *pILink = NULL;

    // Initialze COM
```

```
hr = CoInitializeEx(NULL, COINIT_MULTITHREADED);
if(FAILED(hr))
{
    printf("CoInitializeEx failed w/ 0x%x\n", hr);
    return hr;
}

// Create Advance Queue Administration class
hr = CoCreateInstance( CLSID_AQAdmin,
                       NULL,
                       CLSCTX_INPROC_SERVER,
                       IID_IAQAdmin,
                       (void **) &pIAQAdmin);
if(FAILED(hr))
{
    printf("CoCreateInstance failed with 0x%x\n", hr);
    goto Exit;
}

// Get the admin interface to first Virtual Server
hr = m_pIAQAdmin->GetVirtualServerAdminITF(NULL,
                                           L"1",
                                           &pIVSAQAdmin);
if(FAILED(hr))
{
    printf("GetVirtualServerAdminITF failed with 0x%x\n", hr);
    goto Exit;
}

// start all the link in this virtual server.
hr = pIVSAQAdmin->StartAllLinks();
if(FAILED(hr))
{
    printf("StartAllLinks failed with 0x%x\n", hr);
    goto Exit;
}

printf("StartAllLinks succeeded\n", hr);

// Now get a reference to IEnumVSAQLinks enumerator interface
// to iterate all the links in this virtual server.
hr = pIVSAQAdmin->GetLinkEnum(&pIEnumVSAQLinks);
if(FAILED(hr))
{
    printf("GetLinkEnum failed with 0x%x\n", hr);
    goto Exit;
}

LINK_INFO linkInfo;
ZeroMemory(&linkInfo, sizeof(LINK_INFO));

for(int nCrtLink = 1; TRUE; nCrtLink++)
{
    // get the pointer to the link interface
    hr = GetLink(pIEnumVSAQLinks, &pILink, &linkInfo);
    if(hr == S_FALSE)
    {
        if(nCrtLink == 1)
```

```
        {
            puts("No links.");
            break;
        }
        else if(FAILED(hr))
        {
            break;
        }
        else if(hr == S_OK)
        {
            // Succeeded.
            // Enumerate queues in the link similar way
        }

        if(NULL != pLink)
        {
            pLink->Release();
            pLink = NULL;
        }
        FreeStruct(&linkInfo);
    }

Exit:
    if (NULL != pIVSAQAdmin) pIVSAQAdmin->Release();
    if (NULL != pIVSAQAdmin) pIVSAQAdmin->Release();
    if (NULL != pIEnumVSAQLinks) pIEnumVSAQLinks->Release();
    if(NULL != pLink) pLink->Release();
}

HRESULT GetLink( IN IEnumVSAQLinks *pIEnumVSAQLinks,
                 OUT IVSAQLink **ppILink,
                 IN OUT LINK_INFO *pLinkInfo)
{
    HRESULT hr;
    DWORD cFetched;

    hr = pIEnumVSAQLinks->Next(1, ppILink, &cFetched);
    if(hr == S_FALSE)
    {
        goto Exit;
    }
    else if(FAILED(hr))
    {
        printf("pIEnumVSAQLinks->Next failed with 0x%x\n", hr);
        goto Exit;
    }
    else if(NULL == (*ppILink))
    {
        printf("pLink is NULL.\n", hr);
        goto Exit;
    }
    else
    {
        ZeroMemory(pLinkInfo, sizeof(LINK_INFO));
        pLinkInfo->dwVersion = CURRENT_QUEUE_ADMIN_VERSION;
        hr = (*ppILink)->GetInfo(pLinkInfo);
        if(FAILED(hr))
        {
```

```
            printf("pLink->GetInfo failed with 0x%x\n", hr);
            if ( HRESULT_FROM_WIN32(RPC_S_SERVER_UNAVAILABLE)
                 == hr )
                printf("RPC Server Unavailable.\n");
            else if ( hr == E_POINTER )
                printf("Null pointer.\n");
            else if ( hr == E_OUTOFMEMORY )
                printf("Out of memory.\n");
            else if ( hr == E_INVALIDARG )
                printf("Invalid argument.\n");
            else
                printf("Unknown error.\n");
            goto Exit;
        }
    }

Exit:
    return hr;
}


void FreeStruct(LINK_INFO *pStruct)
{
    if(NULL != pStruct->szLinkName)
    {
        pStruct->szLinkName = NULL;
    }
}
```

5 Security

The following sections specify security considerations for implementers of the IIS IAQ AdminRPC Protocol.

5.1 Security Considerations for Implementers

Implementers should review the security considerations listed in [MS-DCOM] section 5.1, as these are also valid for the IIS IAQ AdminRPC Protocol.

5.2 Index of Security Parameters

None.

6 Appendix A: Full IDL

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

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

[
    object,
    uuid(0c733a30-2a1c-11ce-ade5-00aa0044773d),
    pointer_default(unique)
]
interface ISequentialStream : IUnknown
{
    [local]
    HRESULT Read(
        [out, size_is(cb), length_is(*pcbRead)]
        void *pv,
        [in] ULONG cb,
        [out] ULONG *pcbRead);

    [call_as(Read)]
    HRESULT RemoteRead(
        [out, size_is(cb), length_is(*pcbRead)]
        byte *pv,
        [in] ULONG cb,
        [out] ULONG *pcbRead);

    [local]
    HRESULT Write(
        [in, size_is(cb)] void const *pv,
        [in] ULONG cb,
        [out] ULONG *pcbWritten);

    [call_as(Write)]
    HRESULT RemoteWrite(
        [in, size_is(cb)] byte const *pv,
        [in] ULONG cb,
        [out] ULONG *pcbWritten);
}


[
    object,
    uuid(0000000c-0000-0000-C000-000000000046),
    pointer_default(unique)
]
interface IStream : ISequentialStream
{

    typedef [unique] IStream *LPSTREAM;

    /* Storage stat buffer */

    typedef struct tagSTATSTG
```

```
{
    LPOLESTR pwcsName;
    DWORD type;
    ULARGE_INTEGER cbSize;
    FILETIME mtime;
    FILETIME ctime;
    FILETIME atime;
    DWORD grfMode;
    DWORD grfLocksSupported;
    CLSID clsid;
    DWORD grfStateBits;
    DWORD reserved;
} STATSTG;

/* Storage element types */
typedef enum tagSTGTY
{
    STGTY_STORAGE   = 1,
    STGTY_STREAM    = 2,
    STGTY_LOCKBYTES = 3,
    STGTY_PROPERTY  = 4
} STGTY;

typedef enum tagSTREAM_SEEK
{
    STREAM_SEEK_SET = 0,
    STREAM_SEEK_CUR = 1,
    STREAM_SEEK_END = 2
} STREAM_SEEK;

typedef enum tagLOCKTYPE
{
    LOCK_WRITE      = 1,
    LOCK_EXCLUSIVE  = 2,
    LOCK_ONLYONCE   = 4
} LOCKTYPE;

[local]
HRESULT Seek(
    [in] LARGE_INTEGER dlibMove,
    [in] DWORD dwOrigin,
    [out] ULARGE_INTEGER *plibNewPosition);

[call_as(Seek)]
HRESULT RemoteSeek(
    [in] LARGE_INTEGER dlibMove,
    [in] DWORD dwOrigin,
    [out] ULARGE_INTEGER *plibNewPosition);

HRESULT SetSize(
    [in] ULARGE_INTEGER libNewSize);

[local]
HRESULT CopyTo(
    [in, unique] IStream *pstm,
    [in] ULARGE_INTEGER cb,
    [out] ULARGE_INTEGER *pcbRead,
    [out] ULARGE_INTEGER *pcbWritten);
```

```
    [call_as(CopyTo)]
    HRESULT RemoteCopyTo(
        [in, unique] IStream *pstm,
        [in] ULARGE_INTEGER cb,
        [out] ULARGE_INTEGER *pcbRead,
        [out] ULARGE_INTEGER *pcbWritten);

    HRESULT Commit(
        [in] DWORD grfCommitFlags);

    HRESULT Revert();

    HRESULT LockRegion(
        [in] ULARGE_INTEGER libOffset,
        [in] ULARGE_INTEGER cb,
        [in] DWORD dwLockType);

    HRESULT UnlockRegion(
        [in] ULARGE_INTEGER libOffset,
        [in] ULARGE_INTEGER cb,
        [in] DWORD dwLockType);

    HRESULT Stat(
        [out] STATSTG *pstatstg,
        [in] DWORD grfStatFlag);

    HRESULT Clone(
        [out] IStream **ppstm);
}


interface IVSAQAdmin;
interface IEnumVSAQLinks;
interface IAQEnumMessages;
interface IEnumLinkQueues;
interface IAQMessage;
interface IVSAQLink;
interface ILinkQueue;

typedef  enum tagQUEUE_ADMIN_VERSIONS
{
    CURRENT_QUEUE_ADMIN_VERSION = 4
} QUEUE_ADMIN_VERSIONS;

typedef struct tagMESSAGE_FILTER {
    DWORD dwVersion;
    DWORD fFlags;
    [string] LPCWSTR szMessageId;
    [string] LPCWSTR szMessageSender;
    [string] LPCWSTR szMessageRecipient;
    DWORD dwLargerThanSize;
    SYSTEMTIME stOlderThan;
} MESSAGE_FILTER,
 *PMESSAGE_FILTER;

typedef  enum tagMESSAGE_FILTER_FLAGS
{
```

```
    MF_MESSAGEID   = 0x00000001,
    MF_SENDER      = 0x00000002,
    MF_RECIPIENT   = 0x00000004,
    MF_SIZE        = 0x00000008,
    MF_TIME        = 0x00000010,
    MF_FROZEN      = 0x00000020,
    MF_FAILED      = 0x00000100,
    MF_ALL         = 0x40000000,
    MF_INVERTSENSE = 0x80000000
} MESSAGE_FILTER_FLAGS;

typedef  enum tagMESSAGE_ACTION
{
    MA_THAW_GLOBAL   = 0x00000001,
    MA_COUNT         = 0x00000002,
    MA_FREEZE_GLOBAL = 0x00000004,
    MA_DELETE        = 0x00000008,
    MA_DELETE_SILENT = 0x00000010
} MESSAGE_ACTION;

typedef  enum tagMESSAGE_ENUM_FILTER_TYPE
{
    MEF_FIRST_N_MESSAGES   = 0x00000001,
    MEF_SENDER             = 0x00000002,
    MEF_RECIPIENT          = 0x00000004,
    MEF_LARGER_THAN        = 0x00000008,
    MEF_OLDER_THAN         = 0x00000010,
    MEF_FROZEN             = 0x00000020,
    MEF_N_LARGEST_MESSAGES = 0x00000040,
    MEF_N_OLDEST_MESSAGES  = 0x00000080,
    MEF_FAILED             = 0x00000100,
    MEF_ALL                = 0x40000000,
    MEF_INVERTSENSE        = 0x80000000
} MESSAGE_ENUM_FILTER_TYPE;

typedef struct tagMESSAGE_ENUM_FILTER {
    DWORD dwVersion;
    DWORD mefType;
    DWORD cMessages;
    DWORD cbSize;
    DWORD cSkipMessages;
    SYSTEMTIME stDate;
    [string] LPCWSTR szMessageSender;
    [string] LPCWSTR szMessageRecipient;
} MESSAGE_ENUM_FILTER,
 *PMESSAGE_ENUM_FILTER;

typedef  enum tagLINK_INFO_FLAGS
{
    LI_ACTIVE               = 0x00000001,
    LI_READY                = 0x00000002,
    LI_RETRY                = 0x00000004,
    LI_SCHEDULED            = 0x00000008,
    LI_REMOTE               = 0x00000010,
    LI_FROZEN               = 0x00000020,
    LI_TYPE_REMOTE_DELIVERY = 0x00000100,
    LI_TYPE_LOCAL_DELIVERY  = 0x00000200,
    LI_TYPE_PENDING_ROUTING = 0x00000400,
```

```
    LI_TYPE_PENDING_CAT     = 0x00000800,
    LI_TYPE_CURRENTLY_UNREACHABLE = 0x00001000,
    LI_TYPE_DEFERRED_DELIVERY     = 0x00002000,
    LI_TYPE_INTERNAL        = 0x00004000,
    LI_TYPE_PENDING_SUBMIT  = 0x00008000
} LINK_INFO_FLAGS;

typedef  enum tagLINK_ACTION
{
    LA_INTERNAL = 0x00000000,
    LA_KICK = 0x00000001,
    LA_FREEZE = 0x00000020,
    LA_THAW = 0x00000040
} LINK_ACTION;

typedef struct tagLINK_INFO
{
    DWORD dwVersion;
    [string] LPWSTR szLinkName;
    DWORD cMessages;
    DWORD fStateFlags;
    SYSTEMTIME stNextScheduledConnection;
    SYSTEMTIME stOldestMessage;
    ULARGE_INTEGER cbLinkVolume;
    [string] LPWSTR szLinkDN;
    [string] LPWSTR szExtendedStateInfo;
    DWORD dwSupportedLinkActions;
} LINK_INFO,
 *PLINK_INFO;

typedef struct tagQUEUE_INFO {
    DWORD dwVersion;
    [string] LPWSTR szQueueName;
    [string] LPWSTR szLinkName;
    DWORD cMessages;
    ULARGE_INTEGER cbQueueVolume;
    DWORD dwMsgEnumFlagsSupported;
} QUEUE_INFO,
 *PQUEUE_INFO;

typedef  enum tagAQ_MESSAGE_FLAGS
{
    MP_HIGH = 0x00000001,
    MP_NORMAL = 0x00000002,
    MP_LOW = 0x00000004,
    MP_MSG_FROZEN = 0x00000008,
    MP_MSG_RETRY = 0x00000010,
    MP_MSG_CONTENT_AVAILABLE = 0x00000020
} AQ_MESSAGE_FLAGS;

typedef struct tagMESSAGE_INFO
{
    DWORD dwVersion;
    [string] LPWSTR szMessageId;
    [string] LPWSTR szSender;
    [string] LPWSTR szSubject;
    DWORD cRecipients;
    [string] LPWSTR szRecipients;
```

```
    DWORD cCCRecipients;
    [string] LPWSTR szCCRecipients;
    DWORD cBCCRecipients;
    [string] LPWSTR szBCCRecipients;
    DWORD fMsgFlags;
    DWORD cbMessageSize;
    SYSTEMTIME stSubmission;
    SYSTEMTIME stReceived;
    SYSTEMTIME stExpiry;
    DWORD cFailures;
    DWORD cEnvRecipients;
    DWORD cbEnvRecipients;
    [size_is(cbEnvRecipients/sizeof(WCHAR))]
        WCHAR* mszEnvRecipients;
} MESSAGE_INFO,
 *PMESSAGE_INFO;

typedef  enum tagQUEUELINK_TYPE
{
    QLT_QUEUE,
    QLT_LINK,
    QLT_NONE
} QUEUELINK_TYPE;

typedef struct tagQUEUELINK_ID
{
    GUID uuid;
    [string] LPWSTR szName;
    DWORD dwId;
    QUEUELINK_TYPE qltType;
} QUEUELINK_ID;


[
    helpstring("Advanced Queue Administration Object"),
    object,
    pointer_default(unique),
    uuid(476D70A6-1A90-11d3-BFCB-00C04FA3490A)
]
interface IAQAdmin : IUnknown
{
    HRESULT GetVirtualServerAdminITF(
        [in]  LPCWSTR wszComputer,
        [in]  LPCWSTR wszVirtualServer,
        [out] IVSAQAdmin **ppivsaqadmin
    );
}

[
    helpstring("Advanced Queue Message Action Interface"),
    object,
    pointer_default(unique),
    uuid(1EB44A71-1E95-11d3-BFCC-00C04FA3490A)
]
interface IAQMessageAction : IUnknown
{
    HRESULT ApplyActionToMessages(
        [in] PMESSAGE_FILTER Filter,
```

```
        [in] MESSAGE_ACTION Action,
        [out] DWORD *pcMsgs
    );

    HRESULT QuerySupportedActions(
        [out] DWORD *pdwSupportedActions,
        [out] DWORD *pdwSupportedFilterFlags
    );
}

/*
    @interface IVSAQAdmin | Interface to per-virtual-server AQ
    Administration
    @meth HRESULT | GetLinkEnum | Get an enumerator over all links on
    this VS
    @meth HRESULT | StopAllLinks | Stop all active outbound links
    @meth HRESULT | StartAllLinks | Start all eligible outbound links
*/
[
    helpstring("Advanced Queue Administration Object"),
    object,
    pointer_default(unique),
    uuid(E2ED3340-1E96-11d3-BFCC-00C04FA3490A)
]
interface IVSAQAdmin : IUnknown
{
    HRESULT GetLinkEnum(
        [out] IEnumVSAQLinks **ppenum
    );

    HRESULT StopAllLinks();

    HRESULT StartAllLinks();

    HRESULT GetGlobalLinkState();
}


[
    helpstring("Advanced Queue Administration Object"),
    object,
    pointer_default(unique),
    uuid(ba7af300-7373-11d2-94e4-00c04fa379f1)
]
interface IEnumVSAQLinks : IUnknown
{
    HRESULT Next(
        [in] ULONG cElt,
        [out] IVSAQLink **rgelt,
        [out] ULONG *pcFetched
    );

    HRESULT Skip(
        [in] ULONG cElt
    );

    HRESULT Reset();
```

```
    HRESULT Clone(
        [out] IEnumVSAQLinks **ppenum
    );
}

[
    helpstring("Advanced Queue Administration Object"),
    object,
    pointer_default(unique),
    uuid(3F962F94-1ECD-11d3-BFCC-00C04FA3490A)
]
interface IVSAQLink : IUnknown
{
    HRESULT GetInfo(
        [in, out] PLINK_INFO pli
    );

    HRESULT SetLinkState(
        [in] LINK_ACTION la
    );

    HRESULT GetQueueEnum(
        [out] IEnumLinkQueues **ppenum
    );
}


[
    helpstring("Advanced Queue Administration Object"),
    object,
    pointer_default(unique),
    uuid(ba7af302-7373-11d2-94e4-00c04fa379f1)
]
interface IAQEnumMessages : IUnknown
{
    HRESULT Next(
        [in] ULONG cElt,
        [out] IAQMessage **rgelt,
        [out] ULONG *pcFetched
    );

    HRESULT Skip(
        [in] ULONG cElt
    );

    HRESULT Reset();

    HRESULT Clone(
        [out] IAQEnumMessages **ppenum
    );
}

[
    helpstring("Advanced Queue Administration Object"),
    object,
    pointer_default(unique),
    uuid(ba7af303-7373-11d2-94e4-00c04fa379f1)
]
```

```
interface IEnumLinkQueues : IUnknown
{
    HRESULT Next(
        [in] ULONG cElt,
        [out] ILinkQueue **rgelt,
        [out] ULONG *pcFetched
    );

    HRESULT Skip(
        [in] ULONG cElt
    );

    HRESULT Reset();

    HRESULT Clone(
        [out] IEnumLinkQueues **ppenum
    );
}


[
    helpstring("Advanced Queue Administration Object"),
    object,
    pointer_default(unique),
    uuid(FF9A1BB6-1E96-11d3-BFCC-00C04FA3490A)
]
interface ILinkQueue : IUnknown
{
    HRESULT GetInfo(
        [in, out] PQUEUE_INFO pqi
    );

    HRESULT GetMessageEnum(
        [in] PMESSAGE_ENUM_FILTER pFilter,
        [out] IAQEnumMessages **ppenum
    );
}


[
    helpstring("Advanced Queue Administration Object"),
    object,
    pointer_default(unique),
    uuid(ba7af305-7373-11d2-94e4-00c04fa379f1)
]
interface IAQMessage : IUnknown
{
    HRESULT GetInfo(
        [in, out] PMESSAGE_INFO pmi
    );

    HRESULT GetContentStream(
        [out] IStream **ppIStream,
        [out] LPWSTR *pwszContentType
    );
}
```

```
[
    helpstring("Queue/Link Id Object"),
    object,
    pointer_default(unique),
    uuid(EA4DFDF2-9E87-4c57-B845-123872C5649F)

]
interface IUniqueId : IUnknown
{
    HRESULT GetUniqueId(
        [out] QUEUELINK_ID **ppqlid
    );
}
```

7 Appendix B: Windows Behavior

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

- Windows 2000
- Windows Server 2003
- Windows XP
- 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 Windows does not follow the prescription.

8 Index

R

S

T

V

W