[MS-MQBR]:
Message Queuing (MSMQ): Binary Reliable Message Routing Algorithm

Revision Summary

Date	Revision History	Revision Class	Comments
07/20/2007	0.1	Major	MCPP Milestone 5 Initial Availability
09/28/2007	1.0	Major	Updated and revised the technical content.
10/23/2007	1.0.1	Editorial	Revised and edited the technical content.
11/30/2007	2.0	Major	Updated and revised the technical content.
01/25/2008	2.0.1	Editorial	Revised and edited the technical content.
03/14/2008	2.1	Minor	Updated the technical content.
05/16/2008	2.1.1	Editorial	Revised and edited the technical content.
06/20/2008	2.1.2	Editorial	Revised and edited the technical content.
07/25/2008	2.1.3	Editorial	Revised and edited the technical content.
08/29/2008	3.0	Major	Updated and revised the technical content.
10/24/2008	3.0.1	Editorial	Revised and edited the technical content.
12/05/2008	3.0.2	Editorial	Revised and edited the technical content.
01/16/2009	3.0.3	Editorial	Revised and edited the technical content.
02/27/2009	3.0.4	Editorial	Revised and edited the technical content.
04/10/2009	3.0.5	Editorial	Revised and edited the technical content.
05/22/2009	4.0	Major	Updated and revised the technical content.
07/02/2009	4.1	Minor	Updated the technical content.
08/14/2009	4.1.1	Editorial	Revised and edited the technical content.
09/25/2009	4.2	Minor	Updated the technical content.
11/06/2009	5.0	Major	Updated and revised the technical content.
12/18/2009	5.0.1	Editorial	Revised and edited the technical content.
01/29/2010	5.1	Minor	Updated the technical content.
03/12/2010	5.1.1	Editorial	Revised and edited the technical content.
04/23/2010	5.1.2	Editorial	Revised and edited the technical content.
06/04/2010	5.1.3	Editorial	Revised and edited the technical content.
07/16/2010	6.0	Major	Significantly changed the technical content.

Date	Revision History	Revision Class	Comments
08/27/2010	7.0	Major	Significantly changed the technical content.
10/08/2010	7.0	No change	No changes to the meaning, language, or formatting of the technical content.
11/19/2010	7.0	No change	No changes to the meaning, language, or formatting of the technical content.
01/07/2011	7.0	No change	No changes to the meaning, language, or formatting of the technical content.
02/11/2011	8.0	Major	Significantly changed the technical content.
03/25/2011	9.0	Major	Significantly changed the technical content.
05/06/2011	10.0	Major	Significantly changed the technical content.
06/17/2011	10.1	Minor	Clarified the meaning of the technical content.

Contents

1 Introduction

The Message Queuing (MSMQ): Binary Reliable Message Routing Algorithm is used by **Message Queuing** (also known as MSMQ) to determine the next hop when routing a **message**.

The document describes **intra-site routing**, **inter-site routing** and direct connection between source and destination **MSMQ queue managers** where it is possible, as described in section 1.3.

For more information on MSMQ, see [MSDN-MSMQ].

1.1 Glossary

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

Active Directory
administrator
directory
directory service (DS)
globally unique identifier (GUID)

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

connected network
enterprise
format name
message
message queue
Message Queuing Information Store (MQIS)
Microsoft Message Queuing (MSMQ)
MSMQ Directory Service
MSMQ queue manager
MSMQ routing link
MSMQ routing server
MSMQ site
MSMQ site gate
queue
routing link
routing link cost

The following terms are specific to this document:

In-Routing Server: An **MSMQ routing server** that receives all **messages** on behalf of a particular client and forwards those **messages** to that client.

inter-site routing: The process of routing a **message** between different **MSMQ sites** within an **enterprise**.

intra-site routing: The process of routing a **message** within a single **MSMQ site**.

Out-Routing Server: An **MSMQ routing server** that receives all **messages** sent by a particular client and routes those **messages** on behalf of that client.

routing table: A table maintained by each **MSMQ site gate** for **inter-site routing**. For each **MSMQ site** in an **enterprise**, the table specifies the **MSMQ site** to which a **message** should be forwarded in order to minimize the total **routing link cost** for that **message**.

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

References to Microsoft Open Specification documents do not include a publishing year because links are to the latest version of the documents, which are updated frequently. References to other documents include a publishing year when one is available.

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.

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

[MS-MQDMPR] Microsoft Corporation, "Message Queuing (MSMQ): Common Data Model and Processing Rules".

[MS-MQDSSM] Microsoft Corporation, "Message Queuing (MSMQ): Directory Service Schema Mapping".

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

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

[MS-MQSO] Microsoft Corporation, "Message Queuing System Overview".

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

1.2.2 Informative References

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

[MSDN-MSMQ] Microsoft Corporation, "Message Queuing (MSMQ)", http://msdn.microsoft.com/en-us/library/ms711472.aspx

[RFC1] Cormen, T. H., Leiserson, C. E., and Rivest, R.L., "Introduction to Algorithms".

1.3 Overview

The Message Queuing (MSMQ): Binary Reliable Message Routing Algorithm describes message routing within an **enterprise** network.

Message queuing clients transfer messages either by direct connection with the destination or by sending to an **MSMQ Routing Server**.<1> If a direct connection is not possible or the client is

configured to use a routing service, MSMQ routing servers can temporarily store messages and subsequently forward them to the destination MSMQ queue manager or to another MSMQ routing server.

Message routing occurs when at least one of the following conditions exists.

- The source MSMQ queue manager and the destination MSMQ queue manager belong to different **MSMQ sites**.
- The source MSMQ queue manager and the destination MSMQ queue manager do not share a **connected network**.
- The source MSMQ queue manager is configured to use an **out-routing server**. Every outgoing message from the source MSMQ queue manager is routed through the configured out-routing server.
- The destination MSMQ queue manager is configured to use an **in-routing server**. Every incoming message to the destination MSMQ queue manager is routed through the configured in-routing server.

More information on queuing messages is specified in [MS-MQQB].

1.3.1 Direct Connection

A direct connection between two MSMQ queue managers that share a connected network is possible when the source MSMQ queue manager is not configured to use one or more out-routing server(s), and the destination MSMQ queue manager is not configured to use one or more in-routing server(s). A MSMQ queue manager may belong to more than one connected network.

1.3.2 Intra-Site Routing

If a source MSMQ queue manager is configured to use an out-routing server, every outgoing message is routed through that out-routing server. Similarly, if a destination MSMQ queue manager is configured to use an in-routing server, every incoming message is routed through that in-routing server. Using in-routing and out-routing servers to route messages within an MSMQ site may reduce network bandwidth consumption by providing session concentration.

An MSMQ routing server may also be used to exchange messages between two MSMQ queue managers within an MSMQ site that do not share a common connected network.

An MSMQ routing server may also be used to exchange messages between two MSMQ queue managers within an MSMQ site when direct connection between those MSMQ queue managers fails.

1.3.3 Inter-Site Routing

Administrators can model the physical topology of an enterprise as properties in the **directory**. The Message Queuing (MSMQ): Binary Reliable Message Routing Algorithm uses this model to make routing decisions.

MSMQ sites represent a grouping of MSMQ queue managers in the enterprise network according to physical location. MSMQ queue managers in one MSMQ site use **MSMQ site gates** within the same MSMQ site to route messages to MSMQ queue managers in other MSMQ sites. An MSMQ site gate can route a message to another MSMQ site by sending that message to another MSMQ site gate.

Routing inter-site traffic only through MSMQ site gates often results in session concentration, which can reduce network bandwidth consumption between physically distant nodes.

If an enterprise network has more than one MSMQ site, an **administrator** creates **MSMQ routing links** to allow messages to be routed between those MSMQ sites. MSMQ routing links identify neighboring MSMQ sites whose MSMQ site gates can communicate directly. Each MSMQ routing link includes a **routing link cost** that represents how expensive it is to transfer messages directly between the two sites.

A message may be transferred through multiple MSMQ sites on the way to the destination MSMQ site. Each MSMQ site gate along the way uses a **routing table** to find the next hop in a least-cost path to the destination MSMQ site.

To build the routing table, MSMQ site gates consider the enterprise as a graph with vertices as MSMQ sites and bidirectional nonnegative edge weights as the routing link costs. An MSMQ site gate builds a least-cost spanning tree using its MSMQ site as the root and uses this tree to populate its routing table.

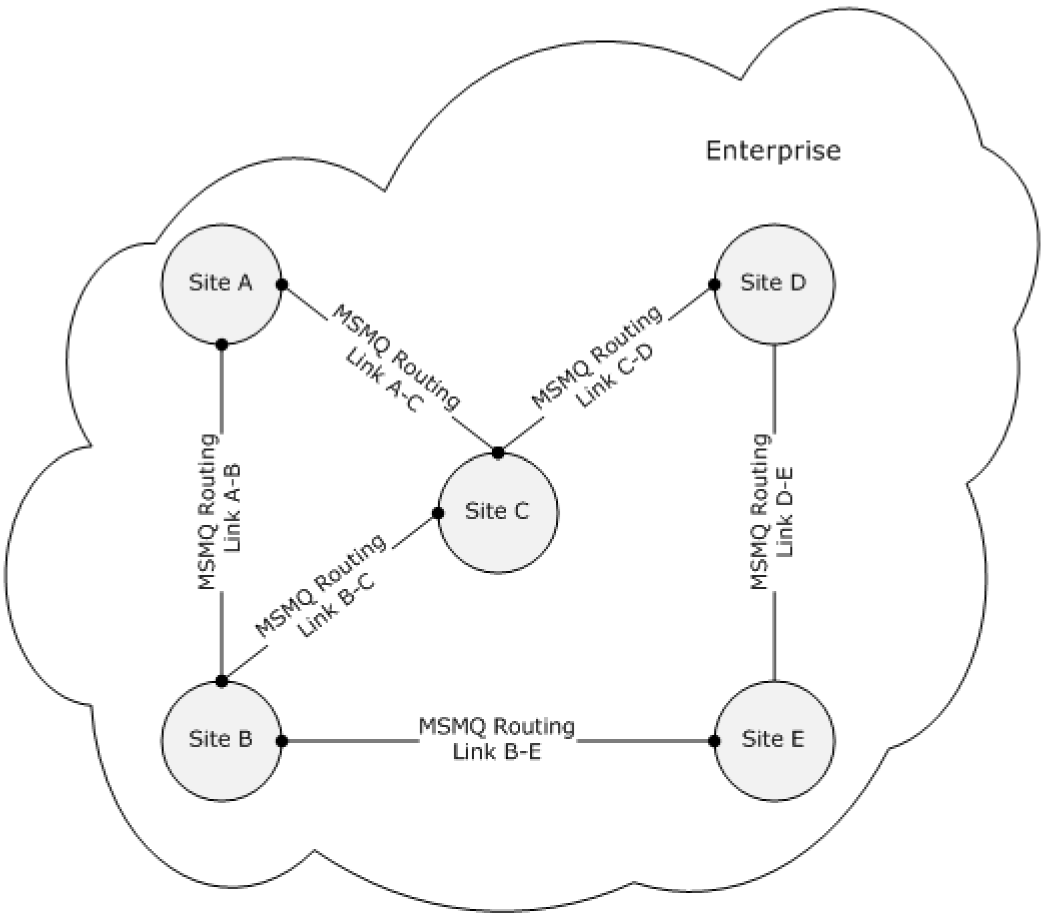

Figure 1: Enterprise as a set of MSMQ sites and MSMQ routing links

Routing link cost provides a mechanism for administrators to enforce one route over another in cases where multiple routes exist.

As described in section 3.1.6.1, each MSMQ queue manager that runs MSMQ within an enterprise periodically queries the directory to determine whether it should act as an MSMQ site gate, and to build a routing table if the querying MSMQ queue manager is an MSMQ site gate.

1.4 Relationship to Other Protocols

The Message Queuing (MSMQ): Binary Reliable Messaging Protocol, as specified in [MS-MQQB], may rely upon the Message Queuing (MSMQ): Binary Reliable Message Routing Algorithm to determine the next hop when messages are sent using public and private **format names**.

The algorithm uses shared state and processing rules defined in Message Queuing (MSMQ): Common Data Model and Processing Rules.

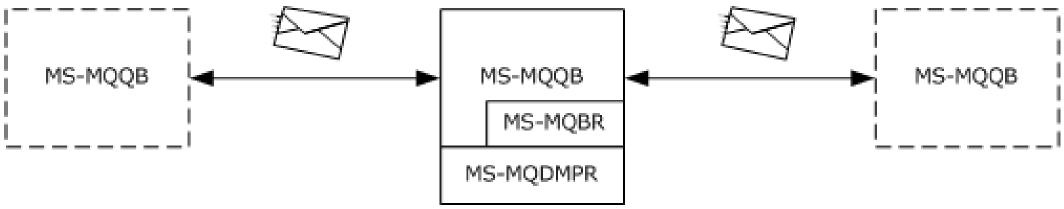

Figure 2: MS-MQQB uses the MS-MQBR algorithm when message routing is required

1.5 Prerequisites/Preconditions

If an enterprise has more than one MSMQ site, each MSMQ site has one or more MSMQ site gates assigned to it.

The following are the requirements for an MSMQ queue manager to be an MSMQ site gate:

- The MSMQ queue manager must belong to the MSMQ site for which it is an MSMQ site gate.
- The MSMQ queue manager must be able to connect directly to each MSMQ site gate in each neighboring MSMQ site.

MSMQ routing servers within an MSMQ site must be able to communicate on all connected networks used by the MSMQ queue managers within that MSMQ site.

1.6 Applicability Statement

The Message Queuing (MSMQ): Binary Reliable Message Routing Algorithm is applicable when a public or private format name message should be routed within an enterprise.

1.7 Versioning and Capability Negotiation

None.

1.8 Vendor-Extensible Fields

None.

1.9 Standards Assignments

None.

2 Messages

2.1 Transport

None.

2.2 Message Syntax

None.

2.3 Directory Service Schema Elements

This algorithm uses abstract data model (ADM) elements specified in section 3.1.1. A subset of these elements can be published in a directory. This algorithm SHOULD<2> access the directory using the algorithm specified in [MS-MQDSSM] and using LDAP [MS-ADTS]. The Directory Service schema elements for ADM elements published in the directory are defined in [MS-MQDSSM] section 2.4.<3>

3 Protocol Details

3.1 Algorithm Details

This section describes a conceptual model of possible data organization that an implementation maintains to implement in this algorithm. The described organization is provided to facilitate the explanation of how the algorithm 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.

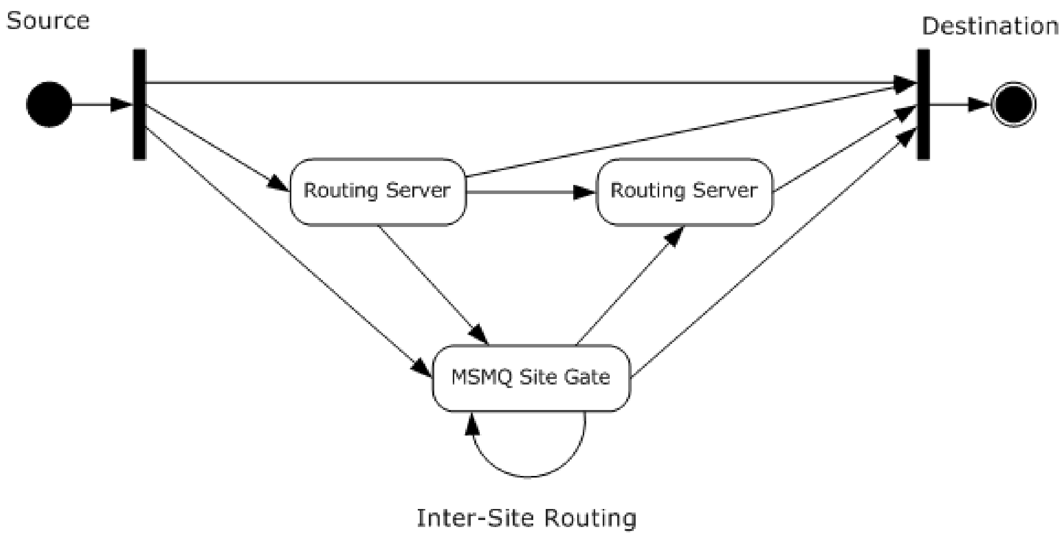

Figure 3: Message Routing

This diagram represents all possible paths that a message may take through an MSMQ enterprise. Each box represents a class of MSMQ queue manager participating in message transfer. Each arrow represents a transfer of a message from an MSMQ queue manager in the class at the tail of the arrow to an MSMQ queue manager in the class at the head of the arrow. In this diagram, the only arrow that represents a message transfer between MSMQ queue managers in different MSMQ sites is the inter-site routing arrow between two MSMQ site gates. All other arrows represent message transfer between MSMQ queue managers within the same MSMQ site.

The Message Queuing (MSMQ): Binary Reliable Message Routing Algorithm determines the path that a message traverses between MSMQ queue managers. The path computed by this algorithm is agnostic of message or **queue** priorities.

3.1.1 Abstract Data Model

The abstract data model for the Message Queuing (MSMQ): Binary Reliable Message Routing Algorithm comprises elements that are private to this algorithm and others that are shared between multiple MSMQ protocols that are co-located at a common MSMQ queue manager. The shared abstract data model is defined in [MS-MQDMPR] section 3.1.1, and the relationship between Message Queuing (MSMQ): Binary Reliable Message Routing Algorithm and other protocols that share a common MSMQ queue manager is described in [MS-MQSO].

Section 3.1.1.1 details the elements from the shared data model that are manipulated by this algorithm, and section 3.1.1.2 details the data model element that is private to Message Queuing (MSMQ): Binary Reliable Message Routing Algorithm.

3.1.1.1 Shared Data Elements

This algorithm manipulates the following abstract data model elements from the shared abstract data model defined in [MS-MQDMPR] section 3.1.1.

DirectoryObject: As defined in [MS-MQDMPR] section 3.1.1.

QueueManager: As defined in [MS-MQDMPR] section 3.1.1.1.

Site: As defined in [MS-MQDMPR] section 3.1.1.7.

RoutingLink: As defined in [MS-MQDMPR] section 3.1.1.8.

3.1.1.2 Routing Table

The RoutingTable contains records of mappings from the unique identifier of the destination Site to the unique identifier of the next-hop Site on the least-cost path to the destination Site.

DestinationSiteID: Site.Identifier that is the unique identifier of the destination Site.

NextHopSiteID: Site.Identifier that is the unique identifier of the next-hop Site.

3.1.2 Timers

The Message Queuing (MSMQ): Binary Reliable Message Routing Algorithm SHOULD maintain the following timer.

3.1.2.1 QueueManager Query Timer

This timer regulates the amount of time that the protocol waits before updating the information in the RoutingTable. This timer fires the QueueManager Query Timer Event, as described in section 3.1.6.1.

3.1.3 Initialization

The QueueManager Query Timer SHOULD be started and SHOULD be initialized internally to 3,600 seconds.

3.1.3.1 RoutingTable Initialization

If IsSiteGate returns 0 when called with the **QueueManager.Identifier** initializing its RoutingTable, the RoutingTable is initialized to be empty.

To create a collection of **Sites** that belong to the enterprise, the GetDirectoryData (section 3.1.5.9) method MUST be called, where the *DataElementType* parameter is the string "Site", and the *FilterArray* parameter has no elements.

To create a collection of **RoutingLinks** that belong to the enterprise, the GetDirectoryData method MUST be called, where the *DataElementType* parameter is the string "RoutingLink", and the *FilterArray* parameter has no elements.

If the collection of **Sites** or the collection of **RoutingLinks** is empty when initializing the RoutingTable, the RoutingTable is initialized to be empty.

Otherwise, to populate the RoutingTable, consider an enterprise as a connected, nonnegative, weighted, directed graph E with vertices S and directed edges L, as follows:

E = (S, L).

Each vertex in S represents a **Site**. Each edge in L represents one direction of a **RoutingLink** between two **Sites**. That is, for any two **Sites** x and y, the directed edge (x, y) exists if and only if a **RoutingLink** exists where x is equal to the **RoutingLink.Site1Identifier** and y is equal to the **RoutingLink.Site2Identifier** or where x is equal to the **RoutingLink.Site2Identifier** and y is equal to the **RoutingLink.Site1Identifier**.

Associated with each edge (x, y) is a weight that is equal to the **RoutingLink.ActualCost** value of the corresponding **RoutingLink**.

The cost of a path between two vertices from S in graph E is a sum of costs of all of the edges in that path. The least-cost path between any two vertices from S in graph E is the path with the lowest cost.

Given such a mapping of an enterprise to a directed graph, discovering the least-cost path between two **Sites** corresponds exactly to finding the least-cost path through a directed graph. Dijkstra's algorithm SHOULD be used to find least-cost paths (as described in [RFC1]) to each destination site by finding a spanning tree that covers the graph. The Message Queuing (MSMQ): Binary Reliable Message Routing Algorithm populates the RoutingTable with a row for each site in the enterprise. To initialize the row for a given site S, the **DestinationSiteId** property contains the **Site.Identifier** of S, and the **NextHopSiteId** contains the **Site.Identifier** of the next-hop site on the path in the spanning tree from the current site to S.

When for a given pair of source and destination vertices there are two or more routes with equal cost, the algorithm selects one of them.

3.1.4 Higher-Layer Triggered Events

None.

3.1.5 Message Processing Events and Sequencing Rules

This section describes how the Message Queuing (MSMQ): Binary Reliable Message Routing Algorithm uses **QueueManager**, **Site**, **RoutingLink** (section 3.1.1 of [MS-MQDMPR]), and RoutingTable (section 3.1.1.2) data elements to determine the next hop for a message.

3.1.5.1 GetNextHops

GetNextHops defines the algorithm that a **QueueManager** ADM element uses to determine the next possible hops for a message to get from the current **QueueManager** ADM element (identified by the *SourceID* parameter) to the ultimate destination (identified by the *DestinationID* parameter). The return value is the list of identifiers of the **QueueManager** ADM elements that can be used as the possible next hop.

```
GetNextHops(SourceID of type GUID, DestinationID of type GUID)

;SourceID      - QueueManager.Identifier that identifies the source
;DestinationID - QueueManager.Identifier that identifies the destination

INIT NextHops of type List of GUID              ;Next hop QueueManager.Identifier
INIT SourceMachine of type QueueManager
INIT DestinationMachine of type QueueManager

SET NextHops to an empty list
SET SourceMachine to result of CALL GetQueueManager(SourceID)
```

```
SET DestinationMachine to result of
    CALL GetQueueManager(DestinationID)

IF SourceMachine <> Nothing AND DestinationMachine <> Nothing THEN
   IF SourceMachine.RoutingServer = True THEN
      SET NextHops to result of GetNextHopsForRouter(SourceMachine,
                                                     DestinationMachine)
   ELSE
      IF SourceMachine.OutRoutingServerIdentifierList is not empty THEN
         SET NextHops to SourceMachine.OutRoutingServerIdentifierList
      ELSE
         IF intersection of SourceMachine.SiteIdentifierList AND
            DestinationMachine.SiteIdentifierList is not empty THEN
            IF DestinationMachine.InRoutingServerIdentifierList is not empty THEN
               SET NextHops to Destination.InRoutingServerIdentifierList
            ELSE
               ADD DestinationID to NextHops
            ENDIF
         ELSE

            ADD the result of CALL GetRoutingServer(SourceID) to NextHops
         ENDIF
      ENDIF
   ENDIF
ENDIF

IF NextHops = Nothing
   Raise Exception   ; Routing Attempt Fails
ENDIF

RETURN with NextHops
```

3.1.5.2 GetNextHopsForRouter

GetNextHopsForRouter defines the algorithm that a **QueueManager** ADM element uses to determine the next hops for a message to get from the source **QueueManager** ADM element (identified by the *SourceMachine* parameter) to the ultimate destination (identified by the *DestinationMachine* parameter), and the source queue manager is an MSMQ Routing Server. The return value is the list of identifiers of the **QueueManager** ADM elements that can be used as the possible next hop. It returns Nothing on failure.

```
GetNextHopsForRouter(SourceMachine of type QueueManager,
                    DestinationMachine of type QueueManager)

INIT NextHops of type List of GUID        ;Next hop QueueManager.Identifier
INIT IsSourceSiteGate of type Boolean

SET NextHops to an empty list


SET IsSourceSiteGate to result of
    CALL IsSiteGate(SourceMachine.Identifier)

IF IsSourceSiteGate = 1 THEN
     SET NextHops to result of
```

```
          CALL GetNextHopsForSiteGate(SourceMachine, DestinationMachine)
ELSE IF SourceMachine.Identifier is one of
     DestinationMachine.InRoutingServerIdentifierList THEN
     ADD DestinationMachine.Identifier to NextHops
ELSE IF intersection of SourceMachine.SiteIdentifierList AND
           DestinationMachine.SiteIdentifierList is not empty THEN
        IF DestinationMachine.InRoutingServerIdentifierList is not
empty THEN
           SET NextHops to DestinationMachine.InRoutingServerIdentifierList
        ELSE
           ADD DestinationMachine.Identifier to NextHops
        ENDIF
ELSE
   ADD the return of
       CALL GetSiteGate(SourceMachine.Identifier) to NextHops
ENDIF

RETURN with NextHops
```

3.1.5.3 GetNextHopsForSiteGate

GetNextHopsForSiteGate defines the algorithm that a **QueueManager** ADM element uses to determine the next possible hops for a message to get from the source **QueueManager** (identified by the *SourceMachine* parameter) to the ultimate destination (identified by the *DestinationMachine* parameter), and the source queue manager is an MSMQ Site Gate. The return value is the list of identifiers of the **QueueManager** ADM elements that can be used as the possible next hop. It returns Nothing on failure.

```
GetNextHopsForSiteGate(SourceMachine of type QueueManager,
                      DestinationMachine of type QueueManager)

INIT NextHops of type List of GUID         ;Next hop QueueManager.Identifier
SET NextHops to an empty list


IF SourceMachine.Identifier
   is one of DestinationMachine.InRoutingServerIdentifierList
THEN
   ADD DestinationMachine.Identifier to NextHops
ELSE IF intersection of SourceMachine.SiteIdentifierList AND
DestinationMachine.SiteIdentifierList in not empty THEN
   IF DestinationMachine.InRoutingServerIdentifierList is not empty THEN
      SET NextHops to DestinationMachine.InRoutingServerIdentifierList
   ELSE
      ADD DestinationMachine.Identifier to NextHops
   ENDIF
ELSE
   INIT NextSite of type GUID
   INIT Entry of type RECORD of RoutingTable

   SET NextSite to Nothing

   FOREACH Entry FROM RoutingTable DO
      IF Entry.DestinationSiteID in
 DestinationMachine.SiteIdentifierList THEN
```

```
            SET NextSite = Entry.NextHopSiteID
         ENDIF
      END FOREACH

      IF NextSite <> Nothing THEN

         ADD the return of CALL GetSiteGateForSite(NextSite) to NextHops

      ENDIF

   ENDIF

   RETURN with NextHops
```

3.1.5.4 GetQueueManager

GetQueueManager returns the **QueueManager** record from the directory identified by the *MachineID* parameter. If no such record is found, it returns Nothing.

```
GetQueueManager(MachineID of type GUID)

;MachineID - QueueManager.Identifier
;corresponding to the requested QueueManager

INIT Machine of type QueueManager
INIT ArrayQM of type vector of QueueManager
INIT StrMachineID of type string

SET Machine to Nothing

SET StrMachineID to MachineID as string

SET ArrayQM to return of CALL GetDirectoryData ("QueueManager",
                     one-element array: "Identifier" EQUALS StrMachineID)
IF ArrayQM <> Nothing THEN
    SET Machine to the first element of ArrayQM
ENDIF

RETURN with Machine
```

3.1.5.5 GetRoutingServer

GetRoutingServer returns the **QueueManager.Identifier** of the MSMQ Routing Server for a given **QueueManager** identified by the input parameter *MachineID*. If there is no MSMQ Routing Server within the Site for the given **QueueManager**, this method returns Nothing.

```
GetRoutingServer(MachineID of type GUID)
;MachineID - QueueManager.Identifier that identifies the QueueManager that
;requested an MSMQ routing server

INIT SourceMachine of type QueueManager
```

```
INIT MachineTemp of type QueueManager
INIT Router type of GUID
INIT ArrayQM of type vector of QueueManager

SET SourceMachine to result of CALL GetQueueManager(MachineID)
IF SourceMachine = Nothing THEN
    RETURN with Nothing
ENDIF

SET Router to Nothing

SET ArrayQM to result of CALL GetDirectoryData("QueueManager", Nothing)

IF ArrayQM = Nothing THEN
    RETURN with Router
ENDIF


FOREACH MachineTemp FROM ArrayQM DO
   IF (intersection MachineTemp.SiteIdentifierList  AND
       SourceMachine.SiteIdentifierList is not empty) AND
       MachineTemp.RoutingServer = True THEN
        SET Router to MachineTemp.Identifier
        BREAK FOREACH
   ENDIF
END FOREACH

RETURN with Router
```

3.1.5.6 GetSiteGate

GetSiteGate returns the **QueueManager.Identifier** of an MSMQ Site Gate for a given **QueueManager** identified by the *MachineID* parameter. If no MSMQ Site Gate is found within the Site to which the QueueManager belongs, this method returns Nothing.

```
GetSiteGate(MachineID of type GUID)

;MachineID – QueueManager.Identifier that identifies the machine

INIT SourceMachine of type QueueManager
INIT SiteGate of type GUID
INIT SiteTemp of type Site
INIT ArraySite of type vector of Site

SET SourceMachine to result of CALL GetQueueManager(MachineID)
IF SourceMachine = Nothing THEN
    RETURN with Nothing
ENDIF

SET SiteGate to Nothing

SET ArraySite to result of CALL GetDirectoryData("Site", Nothing)

IF ArraySite = Nothing THEN
    RETURN with SiteGate
```

```
ENDIF

FOREACH SiteTemp FROM ArraySite DO
   IF SiteTemp.Identifier in
 SourceMachine.SiteIdentifierList THEN
        SET SiteGate to one of SiteTemp.SiteGateCollection
        BREAK FOREACH
   ENDIF
END FOREACH

RETURN with SiteGate
```

3.1.5.7 GetSiteGateForSite

GetSiteGateForSite returns the **QueueManager.Identifier** of the MSMQ site gate for a given MSMQ site identified by the *SiteID* parameter. If no MSMQ site gate is found within the **Site**, this method returns Nothing.

```
GetSiteGateForSite(SiteID of type GUID)

;SiteID  - Site.Identifier  that identifies Site

INIT SiteGate of type GUID
INIT SiteTemp of type Site
INIT ArraySite of type vector of Site
INIT StrSiteID of type string

SET SiteGate to Nothing
SET StrSiteID to SiteID as string


SET ArraySite to result of GetDirectoryData("Site", one-element array: "Identifier" EQUALS
StrSiteID)

IF ArraySite = Nothing THEN
    RETURN with SiteGate
ENDIF

SET SiteTemp to one of ArraySite

SET SiteGate to one of SiteTemp.SiteGateCollection

RETURN with SiteGate
```

3.1.5.8 IsSiteGate

IsSiteGate returns 1 if a given **QueueManager** identified by the *MachineID* parameter is an MSMQ Site Gate. Otherwise this method returns 0.

```
IsSiteGate(MachineID of type GUID)
```

```
;MachineID - QueueManager.Identifier

INIT Machine of type QueueManager
INIT Site of type Site
INIT ArraySite of type vector of Site

SET Machine to result of CALL GetQueueManager(MachineID)
IF Machine = Nothing THEN
    RETURN with 0
ENDIF

IF Machine.RoutingServer = False THEN
   RETURN with 0
ENDIF

SET ArraySite to result of GetDirectoryData("Site", Nothing)
IF ArraySite = Nothing THEN
    RETURN with 0
ENDIF


FOREACH Site FROM ArraySite DO
   IF (Site.Identifier = in Machine.SiteIdentifierList)  THEN
      FOREACH SiteGate in Site.SiteGateCollection DO
         IF (MachineID = SiteGate.Identifier) THEN
            RETURN with 1
         ENDIF
      END FOREACH
   ENDIF
END FOREACH

RETURN with 0
```

3.1.5.9 GetDirectoryData

The GetDirectoryData method returns a vector of **DirectoryObject**. It accepts two parameters. The *DataElementType* parameter is a string that specifies the type of the **DirectoryObject**. The *FilterArray* parameter is an array of **attribute-filter expressions**, as specified in [MS-MQDMPR] section 3.1.7.1.20. Each element in the *FilterArray* parameter specifies a query constraint that MUST be satisfied by all **DirectoryObject(s)** corresponding to the object type specified in the *DataElementType* parameter. This method generates Read Directory Begin, Read Directory Next, and Read Directory End events as specified in [MS-MQDMPR] section 3.1.7.1.21, 3.1.7.1.22, and 3.1.7.1.23 respectively, by using the data type HANDLE as specified in [MS-DTYP] section 2.2.16. This method uses the DirectoryOperationResult enumeration as specified in [MS-MQDMPR] section 3.1.1.17, to determine the outcome of these events.

```
GetDirectoryData( DataElementType of type string,
                  FilterArray of type array of string )

;DataElementType - identifies the type of the Directory's data element to be requested
;FilterArray - identifies constraints for reading data from the Directory

INIT Result of type DirectoryOperationResult
INIT DataTemp of type DirectoryObject
```

```
INIT RequestedData of type vector of DirectoryObject
INIT Handle of type HANDLE


SET RequestedData to Nothing
SET Handle to Nothing

SET (Result, Handle) to result of RAISE Read Directory Begin event with
(iDirectoryObjectType = DataElementType, iFilter = FilterArray)

IF Result <> DirectoryOperationResult.Success THEN
    RETURN with RequestedData
ENDIF

ReadLoop:

SET (Result, DataTemp) to result of RAISE Read Directory Next event with
(Handle)
IF Result = DirectoryOperationResult.Success THEN
   Add DataTemp to RequestedData
   GOTO ReadLoop

ENDIF
IF Result <> DirectoryOperationResult.EndOfData THEN
    SET RequestedData to Nothing
ENDIF


SET Result to result of RAISE Read Directory End event with (Handle)


RETURN with RequestedData
```

3.1.6 Timer Events

3.1.6.1 QueueManager Query Timer Event

When this timer fires, a **QueueManager** reinitializes its RoutingTable, as specified in section 3.1.3.1.

3.1.7 Other Local Events

None.

4 Protocol Examples

The Message Queuing (MSMQ): Binary Reliable Message Routing Algorithm calculates inter-site routing based on the routing link costs.

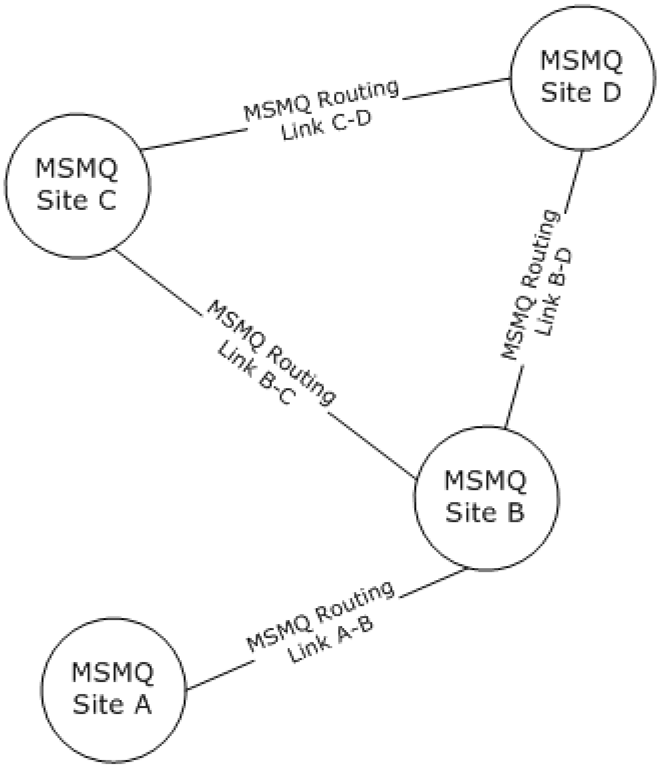

Figure 4: Enterprise example

If the cost associated with the MSMQ routing link A-B is 3 and the cost associated with the other three MSMQ routing links B-C, B-D, and C-D is 1, then messages routed by the Message Queuing (MSMQ): Binary Reliable Messaging Protocol from A to C always travel from A to B and then from B to C. However, if the cost associated with MSMQ routing links A-B and B-C is 3 and the cost associated with MSMQ routing links C-D and B-D is 1, then messages routed from A to C always travel from A to B, from B to D, and then from D to C.

For more details on message packet sequence see [MS-MQQB] section 4.

Release: Friday, June 10, 2011

5 Security

5.1 Security Considerations for Implementers

None.

5.2 Index of Security Parameters

None.

6 Appendix A: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs:

- Microsoft Windows NT® operating system
- Microsoft Windows® 2000 operating system
- Windows® XP operating system
- Windows Server® 2003 operating system
- Windows Vista® operating system
- Windows Server® 2008 operating system
- Windows® 7 operating system
- Windows Server® 2008 R2 operating system

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that the product does not follow the prescription.

<1> Section 1.3: Only Windows NT, Windows 2000, Windows Server 2003, Windows Server 2008, and Windows Server 2008 R2 can be configured as MSMQ routing servers.

<2> Section 2.3: For Windows NT and Windows 2000, this protocol uses the Message Queuing (MSMQ): Directory Service Protocol [MS-MQDS].

<3> Section 2.3: For the Message Queuing (MSMQ): Directory Service Protocol [MS-MQDS], the Directory Service schema elements are described in [MS-MQDS] sections 2.2.10 and 3.1.4.21.1 through 3.1.4.21.4.

7 Change Tracking

This section identifies changes that were made to the [MS-MQBR] protocol document between the May 2011 and June 2011 releases. Changes are classified as New, Major, Minor, Editorial, or No change.

The revision class **New** means that a new document is being released.

The revision class **Major** means that the technical content in the document was significantly revised. Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements or functionality.
- An extensive rewrite, addition, or deletion of major portions of content.
- The removal of a document from the documentation set.
- Changes made for template compliance.

The revision class **Minor** means that the meaning of the technical content was clarified. Minor changes do not affect protocol interoperability or implementation. Examples of minor changes are updates to clarify ambiguity at the sentence, paragraph, or table level.

The revision class **Editorial** means that the language and formatting in the technical content was changed. Editorial changes apply to grammatical, formatting, and style issues.

The revision class **No change** means that no new technical or language changes were introduced. The technical content of the document is identical to the last released version, but minor editorial and formatting changes, as well as updates to the header and footer information, and to the revision summary, may have been made.

Major and minor changes can be described further using the following change types:

- New content added.
- Content updated.
- Content removed.
- New product behavior note added.
- Product behavior note updated.
- Product behavior note removed.
- New protocol syntax added.
- Protocol syntax updated.
- Protocol syntax removed.
- New content added due to protocol revision.
- Content updated due to protocol revision.
- Content removed due to protocol revision.
- New protocol syntax added due to protocol revision.

- Protocol syntax updated due to protocol revision.
- Protocol syntax removed due to protocol revision.
- New content added for template compliance.
- Content updated for template compliance.
- Content removed for template compliance.
- Obsolete document removed.

Editorial changes are always classified with the change type **Editorially updated.**

Some important terms used in the change type descriptions are defined as follows:

- **Protocol syntax** refers to data elements (such as packets, structures, enumerations, and methods) as well as interfaces.
- **Protocol revision** refers to changes made to a protocol that affect the bits that are sent over the wire.

The changes made to this document are listed in the following table. For more information, please contact protocol@microsoft.com.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
1.2 References	Added explanatory statement regarding the removal of the publishing year from Microsoft Open Specification document references.	N	Content updated.

8 Index