[MS-MQDS]:
Message Queuing (MSMQ):
Directory Service Protocol Specification

Revision Summary

Date	Revision History	Revision Class	Comments
04/03/2007	0.01		MCPP Milestone Longhorn Initial Availability
07/03/2007	1.0	Major	MLonghorn+90
07/20/2007	2.0	Major	Updated and revised the technical content.
08/10/2007	3.0	Major	Updated and revised the technical content.

Date	Revision History	Revision Class	Comments
09/28/2007	3.0.1	Editorial	Revised and edited the technical content.
10/23/2007	3.0.2	Editorial	Revised and edited the technical content.
11/30/2007	3.0.3	Editorial	Revised and edited the technical content.
01/25/2008	3.0.4	Editorial	Revised and edited the technical content.

Table of Contents

1 Introduction

This document specifies the Message Queuing (MSMQ): Directory Service Protocol, a **remote procedure call (RPC)**–based protocol that is used by **Message Queuing (MSMQ)** clients and Message Queuing servers to remotely access and maintain **MSMQ directory objects** in the MSMQ: Directory Service Protocol.

1.1 Glossary

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

Access Control List (ACL)
Certificate
Digital Signature
Dynamic Endpoint
Endpoint
Global Catalog Server
Globally Unique Identifier (GUID)
Interface Definition Language (IDL)
MD5 Hash
Network Data Representation (NDR)
Opnum
Private Key
Public Key
Remote Procedure Call (RPC)
RPC Protocol Sequence
RPC Transport
Security Identifier (SID)
Universally Unique Identifier (UUID)
Windows NT Domain Name

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

Backup Site Controller (BSC)
Connected Network
Enterprise
Message
Message Queuing (MSMQ)
MSMQ Directory Service
MSMQ Queue Manager
MSMQ Site
MSMQ Site Link
Primary Site Controller (PSC)
Property Identifier
Queue

The following terms are specific to this document:

Directory Object Property: A property of an **MSMQ directory object**. Each directory object contains a collection of directory properties. The set of properties varies by type of directory object.

MSMQ Directory Object: An object in the directory service representing an entity related to **message queuing**. An object may represent any one of the following entities: **Enterprise**, **Connected Network**, **Site**, **Site Link**, Machine, User, or **Queue**.

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

[GSS] Piper, D. and Swander, B., "A GSS-API Authentication Method for IKE", Internet Draft, July 2001, http://www3.ietf.org/proceedings/02mar/I-D/draft-ietf-ipsec-isakmp-gss-auth-07.txt

If you have any trouble finding [GSS], please check here.

[LANMAN] Microsoft Corporation, "LAN Manager Authentication Level", http://msdn2.microsoft.com/en-us/library/ms814176.aspx

If you have any trouble finding [LANMAN], please check here.

[MS-ADTS] Microsoft Corporation, "Active Directory Technical Specification", June 2007.

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

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

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

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

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

[MS-RDPBCGR] Microsoft Corporation, "Remote Desktop Protocol: Basic Connectivity and Graphics Remoting Specification", June 2007.

[NTLM] Microsoft Corporation, "Microsoft NTLM", http://msdn2.microsoft.com/en-us/library/aa378749.aspx

If you have any trouble finding [NTLM], please check here.

[PCT1] Benalogh, J., Lampson, B., Simon, D., Spies, T., and Yee, B., "The Private Communication Technology (PCT) Protocol", October 1995, http://tools.ietf.org/html/draft-benaloh-pct-00

If you have any trouble finding [PCT1], please check here.

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

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

[RFC2743] Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, January 2000, http://www.ietf.org/rfc/rfc2743.txt

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

1.2.2 Informative References

[LDAP] Microsoft Corporation, "About Lightweight Directory Access Protocol", http://msdn2.microsoft.com/en-us/library/aa366075.aspx

If you have any trouble finding [LDAP], please check here.

1.3 Protocol Overview (Synopsis)

Message Queuing is a communications service that provides asynchronous and reliable **message** passing between client applications, including between client applications running on different hosts. In Message Queuing, clients send messages to **queues** and consume messages from queues. Queues provide message persistence, enabling the sending and receiving of client applications to operate asynchronously from each other.

Because Message Queuing involves message passing between nodes, a directory service can be useful to Message Queuing services in several ways. First, a directory service can provide network topology information that the Message Queuing services can use to route messages between nodes. Second, a directory service can be used as a key distribution mechanism for security services that are used by Message Queuing to secure messages and authenticate clients. Third, a directory service can provide clients with discovery capabilities, allowing clients to discover the queues available within the network.

The Message Queuing (MSMQ): Directory Service Protocol provides a set of procedure calls that can be made between a client and an **MSMQ** Directory Server. The client uses these calls to access the Directory Service remotely. For example, a client may use this protocol to create queue objects in a directory. This protocol is intended for use by Message Queuing clients and Message Queuing servers.

The directory defined by the MSMQ: Directory Service Protocol is composed of eight types of directory objects representing **enterprises**, **sites**, **site links**, machines, users, queues, **connected networks**, and deleted objects.

Each directory object is composed of a collection of properties. Each property has an integer **property identifier** and a variant property value. Properties are specific to the directory object type. Most directory object types include a **GUID** property to identify a particular object instance, a pathname property specifying where in the directory the object is stored, and security properties. Some **directory object properties** are assigned by the directory service while other directory object properties are specified by the client. Some properties are immutable; some properties are mutable by the directory server but not by the client while other properties are mutable by both.

The MSMQ: Directory Service Protocol provides methods to create, update, retrieve, and delete objects from the directory service by using either the object name or the unique object GUID as a key to identify the object. Separate interface methods are implemented to manipulate object security properties.

The MSMQ: Directory Service Protocol also provides a simple query mechanism that allows the enumeration of directory objects through comparison with client-supplied values. The client can specify the matching criteria, the properties to be returned, and the sort order for the results. The

server computes the result set. Thereafter, the client retrieves the results in order, in an iterative manner through repeated calls to the server, each call returning the next portion of the result set.

The MSMQ: Directory Service Protocol includes a method for RPC **endpoint** port negotiation. Through this, the client can determine the RPC endpoint port to use for this protocol.

Generally, for methods that create, update, or delete information in the directory service, the MSMQ: Directory Service Protocol relies on security mechanisms of the underlying **RPC transport** to provide client authentication information to the server. There are two exceptions to this. When setting properties on a machine object, and when retrieving the security properties of a machine object, the server may invoke a challenge/response callback to the client to authenticate the client. This client signs the challenge by using a **private key**, and the server validates the signature by using a corresponding **public key** stored with the machine object in the directory.

Because the directory service provides network topology information and security key distribution, clients must be able to trust the source of this data. Therefore, this protocol includes methods for a security handshake to allow mutual authentication and to establish cryptographic keys that are used to compute **digital signatures**. These handshake methods tunnel Generic Security Service API (GSS-API), as specified in [RFC2743], operations to establish a security context. See [RFC2743] section 2.2.

All methods that return data to the client include signed hashes over returned data, allowing the client to authenticate the source of the data and verify that the data has not been tampered with en route. The signed hashes are computed by using the established security context.

This is an RPC–based protocol consisting of simple request-response exchanges. For every method request that the server receives, it executes the method and returns a completion. The client simply returns the completion status to the caller.

1.4 Relationship to Other Protocols

The Message Queuing (MSMQ): Directory Service Protocol depends on RPC for its transport and uses RPC, as specified in section 2.1.

The following diagram illustrates protocol layering.

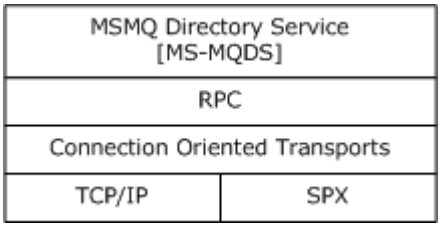


Figure 1: Protocol layering

The MSMQ: Directory Service Protocol is deprecated. Implementers SHOULD use Lightweight Directory Access Protocol (LDAP) [MS-ADTS] instead of this protocol, except where compatibility requirements necessitate use of this protocol.<1>

This protocol relies on The Private Communication Technology (PCT) Protocol, as specified in [PCT1], for authentication and message security. This protocol uses PCT as the security mechanism underlying the GSS API, as specified in [RFC2743], and cannot be configured to use other security mechanisms, such as SSL or TLS.

1.5 Prerequisites/Preconditions

The Message Queuing (MSMQ): Directory Service Protocol is an RPC interface, and as a result has the prerequisites identified in Remote Procedure Call Protocol Extensions, as specified in [MS-RPCE], as being common to RPC interfaces.

It is assumed that an MSMQ: Directory Service Protocol client has obtained the name of a remote computer that supports the MSMQ: Directory Service Protocol before this protocol is invoked. This specification does not mandate how a client acquires this information.

It is assumed that the directory has been configured with an enterprise object and at least one site object. It is also assumed that the protocol server is configured to belong to a site of the directory. Furthermore, it is assumed that the protocol server has been configured with a list of sites in the enterprise, and the list of **Primary Site Controllers (PSC)** and **Backup Site Controllers (BSC)** for each site. Clients can retrieve this list of sites from the directory service through the **S_DSCreateServersCache** method.

The MSMQ: Directory Service Protocol requires authentication and message protection through the use of the GSS-API, as specified in [RFC2743]; therefore, the client and server require infrastructure supporting it.

The client and server must possess valid security credentials suitable for mutual authentication and supported by [PCT1].

1.6 Applicability Statement

The Message Queuing (MSMQ): Directory Service Protocol can be used to provide directory service functionality. It is intended for use by message queuing clients and message queuing servers, and it supports a fixed and limited set of directory object types and directory object properties. It is not intended as a general-purpose directory service.

The functionality of this protocol has been superseded by Active Directory and [MS-ADTS]. Future development based on this protocol is strongly discouraged.<2>

1.7 Versioning and Capability Negotiation

This document covers versioning issues in the following areas:

- **Supported Transports:** The Message Queuing (MSMQ): Directory Service Protocol uses multiple **RPC protocol sequences**, as specified in section 2.1.
- **Protocol Versions:** This protocol has multiple interfaces, as specified in sections 3.1 and 3.3.
- **Security and Authentication Methods:** This protocol supports the following authentication methods: [NTLM] and Kerberos.
- **Capability Negotiation:** The MSMQ: Directory Service Protocol does not support negotiation of the interface version to use. Instead, this protocol uses only the interface version number in the **IDL** for versioning and capability negotiation.<3>
- **Localization:** This protocol passes text strings in various methods, but these strings are not subject to localization because they are location-invariant.

1.8 Vendor-Extensible Fields

The Message Queuing (MSMQ): Directory Service Protocol uses HRESULTs, as specified in [MS-DTYP] section 2.2.19. Vendors are free to choose their own values for this field as long as the C bit (0x20000000) is set, indicating it is a customer code.

1.9 Standards Assignments

Parameter	Value	Reference
RPC Interface **UUID** for dscomm	{77df7a80-f298-11d0-8358-00a024c480a8}	As specified in [C706]
RPC Interface UUID for dscomm2	{708cca10-9569-11d1-b2a5-0060977d8118}	As specified in [C706]

2 Messages

The following sections specify how to establish a connection to an RPC server.

2.1 Transport

The Message Queuing (MSMQ): Directory Service Protocol uses RPC **dynamic endpoints**, as specified in [C706].<4>

Servers MUST support the following RPC protocol sequence: RPC over TCP/IP, as specified in [MS-RPCE]. Servers MAY support the following RPC protocol sequence: RPC over Sequenced Packet Exchange (SPX), as specified in [MS-RPCE].<5>

The MSMQ: Directory Service Protocol MUST use the UUID, as specified in section 1.9. The RPC interface version number MUST be 1.0.

This protocol uses security information as specified in [MS-RPCE]. This protocol uses NTLM and Kerberos security providers.<6>

This protocol allows any user to establish a connection to an RPC server. The server uses the underlying RPC protocol to retrieve the identity of the method caller, as specified in [MS-RPCE] section 3.3.3.4.3. The server SHOULD use this identity to perform method-specific access checks, as specified in section 3.1.4.

2.2 Common Data Types

The Message Queuing (MSMQ): Directory Service Protocol MUST indicate to the RPC runtime that it is to support both the NDR20 and NDR64 transfer syntaxes, and MUST provide a negotiation mechanism for determining what transfer syntax will be used, as specified in [MS-RPCE] section 3.

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

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

The following table summarizes the types defined in this specification.

Type	Description
HRESULT	A result handle.
GUID	A globally unique identifier.
PROPVARIANT	A variant type for property values.
SECURITY_DESCRIPTOR	Constructed security type.
SID	Constructed security type.
SERVER_AUTH_STRUCT	Server authorization credentials.
PCONTEXT_HANDLE_SERVER_AUTH_TYPE	An RPC context handle used to provide security context.
PCONTEXT_HANDLE_TYPE	An RPC context handle used in search enumerations.

Type	Description
PCONTEXT_HANDLE_DELETE_TYPE	An RPC context handle used in sending delete notifications.
Object types	Object type code.
Queue property identifiers	Queue property codes.
Machine property identifiers	Machine property codes.
Site property identifiers	Site property codes.
Deleted object property identifiers	Deleted object property codes.
Connected network property identifiers	Connected network property codes.
Enterprise property identifiers	Enterprise property codes.
User property identifiers	User property codes.
Site link property identifiers	Site link property codes.
MQPROPERTYRESTRICTION	A directory query restriction based on a property.
MQRESTRICTION	A set of property restrictions.
MQCOLUMNSET	A set of property names.
MQSORTKEY	A sort key based on a property.
MQSORTSET	A set of sort keys.
Server Specification List String	List of server specifications.
Server List String	List of servers associated with a site.
MQDSPUBLICKEY	A public key **certificate**.
MQDSPUBLICKEYS	A set of public key certificates.
BLOBHEADER	A public key binary large object (BLOB) header.
MQDS_PublicKey	A structure to hold a public key BLOB.

2.2.1 HRESULT

This specification uses the HRESULT type, as specified in [MS-DTYP] section **2.2.16**.

2.2.2 GUID

This specification uses a GUID. See [MS-DTYP] section **2.3.7**.

2.2.3 PROPVARIANT

This specification uses the PROPVARIANT type. See [MS-MQMQ] section **2.2.13.2**.

2.2.4 SECURITY_DESCRIPTOR

This specification uses Security Descriptors. See [MS-DTYP] section **2.4.6**.

2.2.5 SERVER_AUTH_STRUCT

The **SERVER_AUTH_STRUCT** structure is used by the server to maintain negotiated security context information. The contents of the structure do not appear on the wire. The client treats the structure as opaque. It is included here because it is used in the definition of the RPC context handle type, as specified in **PCONTEXT_HANDLE_SERVER_AUTH_TYPE**. RPC context handles are as specified in [C706] section 6.1.6.

```
typedef struct SERVER_AUTH_STRUCT_tag {
  void* pvReserved;
  unsigned long ulReserved1;
  unsigned long ulReserved2;
} SERVER_AUTH_STRUCT;
```

2.2.6 PCONTEXT_HANDLE_SERVER_AUTH_TYPE

The **PCONTEXT_HANDLE_SERVER_AUTH_TYPE** is an RPC context handle type. This type identifies the security context established through the [GSS] negotiation. The security context is used to construct digital signatures over the returned data. See **S_DSValidateServer**.

An RPC context handle specifies the information that is necessary to enable the RPC subsystem on the server to keep state information on a per-session basis, and to do resource cleanup if the session is broken and the client cannot close the connection in an orderly manner. RPC context handles are as specified in [C706] section 6.1.6.

```
typedef [context handle] SERVER AUTH STRUCT* PCONTEXT HANDLE SERVER AUTH TYPE;

typedef [ref] PCONTEXT_HANDLE_SERVER_AUTH_TYPE* PPCONTEXT_HANDLE_SERVER_AUTH_TYPE;
```

2.2.7 PCONTEXT_HANDLE_TYPE

The **PCONTEXT_HANDLE_TYPE** is an RPC context handle type. This type is used to identify a directory query result set. See **S_DSLookupBegin**.

An RPC context handle specifies the information necessary to enable the RPC subsystem on the server to keep state information on a per-session basis and to do resource cleanup if the session is broken and the client cannot close the connection in an orderly manner. RPC context handles are as specified in [C706] section 6.1.6.

```
typedef [context_handle] void* PCONTEXT_HANDLE_TYPE;

typedef [ref] PCONTEXT_HANDLE_TYPE* PPCONTEXT_HANDLE_TYPE;
```

2.2.8 PCONTEXT_HANDLE_DELETE_TYPE

The **PCONTEXT_HANDLE_DELETE_TYPE** is an RPC context handle type. This type is used to identify a pending directory object delete notification. See **S_DSBeginDeleteNotification (section 3.3.4.4)**.

An RPC context handle specifies the information that is necessary to enable the RPC subsystem on the server to keep state information on a per-session basis and to do resource cleanup if the session is broken and the client cannot close the connection in an orderly manner. RPC context handles are as specified in [C706] section 6.1.6.

```
typedef [context handle] void* PCONTEXT HANDLE DELETE TYPE;

typedef [ref] PCONTEXT_HANDLE_DELETE_TYPE* PPCONTEXT_HANDLE_DELETE_TYPE;
```

2.2.9 Directory Object Types

The directory consists of eight types of directory objects. The directory object type is specified by a **DWORD** value as specified by the following table.

Name	Value (decimal)	Meaning
MQDS_QUEUE	1	Object represents a **message queue**.
MQDS_MACHINE	2	Object represents a **queue manager**.
MQDS_SITE	3	Object represents a site.
MQDS_DELETEDOBJECT	4	Object has been deleted.
MQDS_CN	5	Object represents a connected network.
MQDS_ENTERPRISE	6	Object represents an enterprise.
MQDS_USER	7	Object represents a user.
MQDS_SITELINK	8	Object represents a site link.

2.2.10 Directory Object Properties

Each directory object type, as specified in section 2.2.9, has a set of directory object properties associated with it. Each directory object property has a property identifier, a variant type, and a semantic.

Some directory object properties are interpreted by this protocol; other directory object properties are simply treated as payload by this protocol. The following sections define the property identifier ranges associated with each directory object type and the subset of the directory object properties that are interpreted by this protocol.

2.2.10.1 Object Property Identifiers

Each directory object property has associated with it a unique **DWORD** property identifier. The following table specifies the valid property identifier ranges for each directory object type.

Directory object type	Range of valid property identifiers (decimal)
MQDS_QUEUE	101-126
MQDS_MACHINE	201-243
MQDS_SITE	301-312
MQDS_DELETEDOBJECT	None
MQDS_CN	501-505
MQDS_ENTERPRISE	601-618
MQDS_USER	701-706
MQDS_SITELINK	801-813

Property identifiers equal to or greater than 1000 (decimal) are reserved for use of the server implementation. Clients cannot get or set properties with property identifiers in this reserved range but can associate some properties with property identifiers in this range at object creation time. Properties in this reserved range, which can be specified at object creation time, will be listed in the tables in the following sections in this topic.

2.2.10.2 Queue Object Properties

The following table specifies the set of properties associated with directory objects of type MQDS_QUEUE that are used by this protocol.

Property identifier	Variant type	Meaning
PROPID_Q_INSTANCE 101	VT_CLSID	GUID identifier for the directory object instance.
PROPID_Q_QMID 115	VT_CLSID	GUID identifier for the machine directory object that owns this queue.
PROPID_Q_OBJ_SECURITY 1102	VT_BLOB	Security descriptor for the queue object in Active Directory format.

2.2.10.3 Machine Object Properties

The following table specifies the set of directory object properties associated with directory objects of type MQDS_MACHINE that are used by this protocol.

Property identifier	Variant type	Meaning
PROPID_QM_SITE_ID 201	VT_CLSID	The GUID identifier for the site directory object that owns this machine.
PROPID_QM_MACHINE_ID 202	VT_CLSID	The GUID identifier for the directory object instance.
PROPID_QM_SIGN_PK	VT_BLOB	The machine's public key certificates formatted as an

Property identifier	Variant type	Meaning
1202		MQDSPUBLICKEYS structure. This property may only be specified at object creation time.
PROPID_QM_ENCRYPT_PK 1203	VT_BLOB	The machine's public key certificates formatted as an MQDSPUBLICKEYS structure. The certificates contain public keys used to validate signatures. This property may only be specified at object creation time.
PROPID_QM_OBJ_SECURITY 234	VT_BLOB	The security descriptor for the machine object in Active Directory format.
PROPID_QM_ENCRYPT_PKS 238	VT_BLOB	The machine's public key certificates formatted as an MQDSPUBLICKEYS structure.
PROPID_QM_SIGN_PKS 239	VT_BLOB	The machine's public key certificates formatted as an MQDSPUBLICKEYS structure.
PROPID_QM_SITE_IDS 222	(VT_CLSID | VT_VECTOR)	Array of site IDs to which the machine belongs.

2.2.10.4 Site Object Properties

The following table specifies the set of directory object properties associated with directory objects of type MQDS_SITE that are used by this protocol.

Property identifier	Variant type	Meaning
PROPID_S_SITEID 302	VT_CLSID	GUID identifier for the directory object instance.
PROPID_S_PSC_SIGNPK 1302	VT_BLOB	The site's public key certificate used for signing, formatted as an MQDSPUBLICKEYS structure. This property may only be specified at object creation time.

2.2.10.5 Connected Network Object Properties

The following table specifies the set of directory object properties associated with directory objects of type MQDS_CN that are used by this protocol.

Property identifier	Variant type	Meaning
PROPID_CN_GUID 503	VT_CLSID	GUID identifier for the directory object instance.

2.2.10.6 Enterprise Object Properties

The following table specifies the set of directory object properties associated with directory objects of type MQDS_ENTERPRISE that are used by this protocol.

Property identifier	Variant type	Meaning
PROPID_E_ID 609	VT_CLSID	GUID identifier for the directory object instance.

2.2.10.7 User Object Properties

The following table specifies the set of directory object properties associated with directory objects of type MQDS_USER that are used by this protocol.

Property identifier	Variant type	Meaning
PROPID_U_ID 706	VT_CLSID	GUID identifier for the directory object instance.

2.2.10.8 Site Link Object Properties

The following table specifies the set of directory object properties associated with directory objects of type MQDS_SITELINK that are used by this protocol.

Property identifier	Variant type	Meaning
PROPID_L_ID 806	VT_CLSID	GUID identifier for the directory object instance.

2.2.11 MQPROPERTYRESTRICTION

The **MQPROPERTYRESTRICTION** structure specifies a logical expression over a directory object property. The logical expression evaluates to TRUE or FALSE. The logical expression is defined through a relational comparison operation between a directory object property and a specified constant value.

```
typedef struct tagMQPROPERTYRESTRICTION {
  unsigned long rel;
  unsigned long prop;
  PROPVARIANT prval;
} MQPROPERTYRESTRICTION;
```

rel: Specifies the binary relation to be computed between the directory object property identified by prop and the constant specified by prval. The value of this field MUST be one of the values as defined below.

Value	Meaning
PRLT 0x00000000	Less than.
PRLE 0x00000001	Less than or equal to.
PRGT	Greater than.

Value	Meaning
0x00000002	
PRGE 0x00000003	Greater than or equal to.
PREQ 0x00000004	Equal to.
PRNE 0x00000005	Not equal to.

prop: A property identifier specifying what directory object property to be used as the left operand in the binary relation specified in rel. MUST be one of the values specified in the object property identifier table, as specified in section 2.2.10.1.

prval: A constant value to be used as the right operand in the binary relation specified in rel. The variant type of prval MUST match the variant type of the directory object property identified by prop as specified in the property identifier tables in section 2.2.8.

This structure is used in directory query operations to define a single constraint over the set of directory objects to be returned. An object is deemed to satisfy the constraint if the binary expression, as specified by the **MQPROPERTYRESTRICTION** structure, evaluates to TRUE, and is deemed not to satisfy the constraint otherwise. See section 3.1.4.14.

2.2.12 MQRESTRICTION

The **MQRESTRICTION** structure specifies a set of **MQPROPERTYRESTRICTION** structures. This structure is used in directory query operations to define a set of constraints over the set of directory objects to be returned. An object is deemed to satisfy the constraint if all the binary expressions specified by the **MQPROPERTYRESTRICTION** array evaluate to TRUE, and is deemed not to satisfy the constraint otherwise. See section 3.1.4.14.

```
typedef struct tagMQRESTRICTION {
  unsigned long cRes;
  [size_is(cRes)] MQPROPERTYRESTRICTION* paPropRes;
} MQRESTRICTION;
```

cRes: MUST be set to the count of **MQPROPERTYRESTRICTION** structures in the **paPropRes** array.

paPropRes: A pointer to an array of **MQPROPERTYRESTRICTION** structures.

2.2.13 MQCOLUMNSET

The **MQCOLUMNSET** structure specifies a list of directory object property identifiers. This structure is used in directory query operations to define the set of directory object properties to be returned. See section 3.1.4.14.

```
typedef struct tagMQCOLUMNSET {
  unsigned long cCol;
  [size_is(cCol)] PROPID* aCol;
} MQCOLUMNSET;
```

cCol: MUST be set to the count of property identifiers in **aCol**.

aCol: A pointer to an array of property identifiers. Each element of the array MUST be one of the values specified in the object property identifier table in section 2.2.10.1.

2.2.14 MQSORTKEY

The **MQSORTKEY** structure specifies a sort key and sort order. This structure is used in directory query operations to identify a directory object property to be used as a sort key by which to sort the result set, and to define the sort order for that key.

```
typedef struct tagMQSORTKEY {
  unsigned long propColumn;
  unsigned long dwOrder;
} MQSORTKEY;
```

propColumn: The property identifier to be used as the sort key. MUST be one of the values specified in the object property identifier table in section 2.2.10.1.

dwOrder: A symbolic constant specifying whether the sort is to be done in ascending or descending order. MUST be set to one of the following values:

Value	Meaning
QUERY_SORTASCEND 0x00000000	Ascending sort.
QUERY_SORTDESCEND 0x00000001	Descending sort.

2.2.15 MQSORTSET

The **MQSORTSET** structure specifies a multipart sort key. This structure is used in directory query operations to define a collection of sort keys and sort orders by which to sort the result set. See section 3.1.4.14.

```
typedef struct tagMQSORTSET {
  unsigned long cCol;
  [size_is(cCol)] MQSORTKEY* aCol;
} MQSORTSET;
```

cCol: MUST be set to the count of **MQSORTKEY** structures referenced by **aCol**.

aCol: A pointer to an array of **MQSORTKEY** structures.

2.2.16 Server Specification List String

A Server Specification List String is a **UNICODE** string that specifies a set of directory servers. The following ABNF notation defines the format of this string:

```
<server-spec-list> = *(<server-spec> ",") <server-spec>

<server-spec> = <support-IP> <support-IPX> <name>

<support-IP> = "0" / "1"
; "0" if the server does not support IP addressing
; "1" if the server supports IP addressing

<support-IPX> = "0" / "1"
; "0" if the server does not support IPX addressing
; "1" if the server supports IPX addressing

; <name> is the name of a directory service server
```

2.2.17 Server List String

A Server List String is a **UNICODE** string that specifies a set of directory servers associated with a site. The following ABNF notation defines the format of this string:

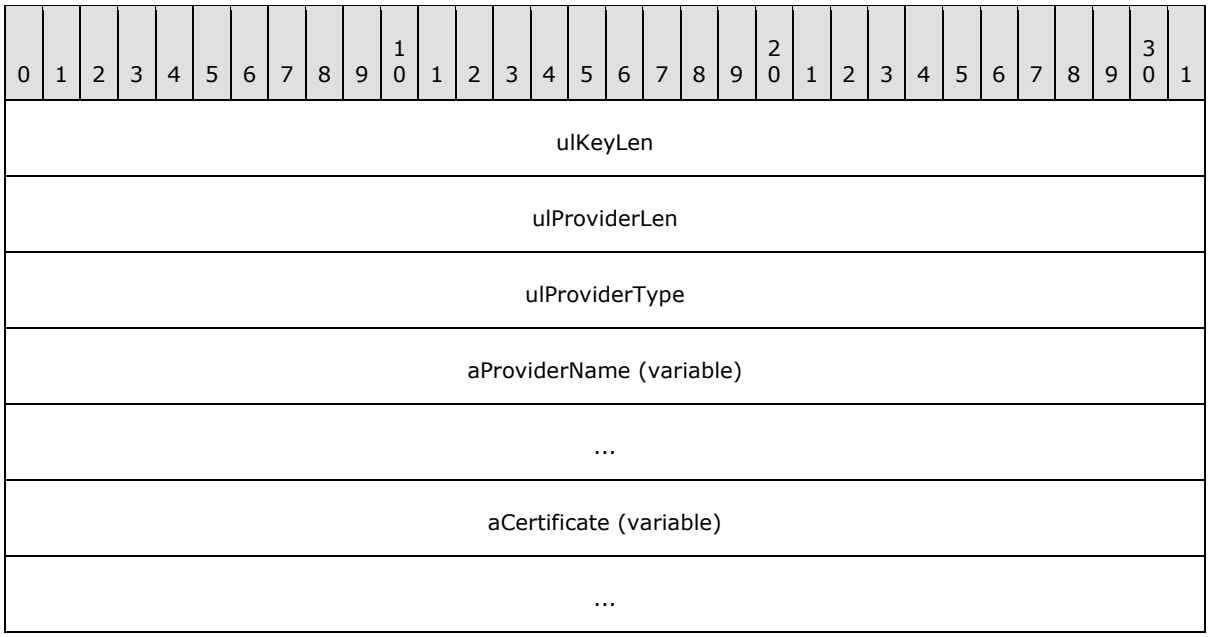
```
<server-list> = <site-name> ";" "\" <server-spec-list>

; <site-name> is the name of the site

; <server-spec-list> is a server specification list as defined in section
2.2.16
```

2.2.18 MQDSPUBLICKEY

The MQDSPUBLICKEY structure defines a public key certificate.



ulKeyLen (4 bytes): An unsigned 32-bit integer that MUST contain the size in bytes of the MQDSPUBLICKEY structure.

ulProviderLen (4 bytes): An unsigned 32-bit integer that MUST contain the size in bytes of the provider name, including the terminating null.

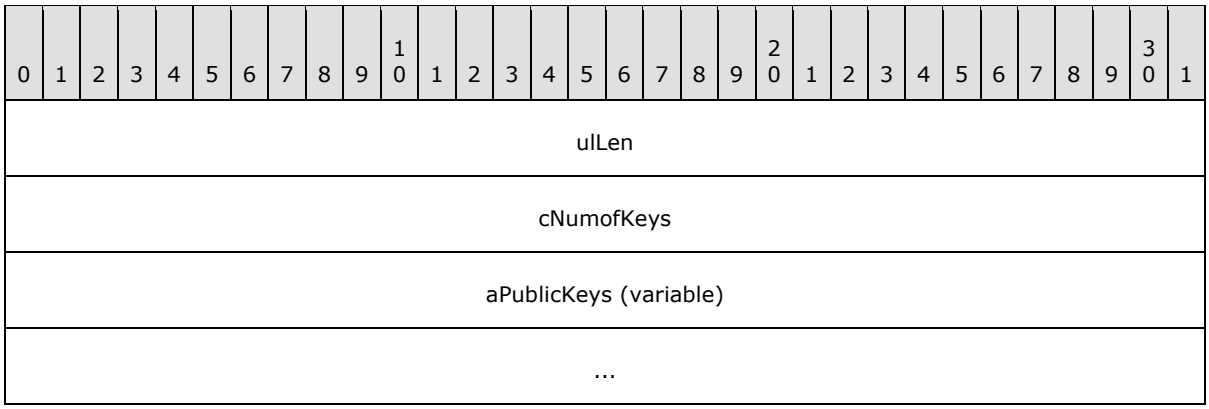
ulProviderType (4 bytes): An unsigned 32-bit integer that MUST contain an enumerated constant for the provider type code. The only value that has meaning to this protocol is PROV_RSA_FULL (0x00000001).

aProviderName (variable): A buffer containing the provider name as a **UNICODE** string, null-terminated. The only value that has meaning to this protocol is "Microsoft Base Cryptographic Provider 1.0".

aCertificate (variable): A buffer containing the public key certificate, formatted as specified in [MS-RDPBCGR] section 2.2.1.4.3.1.1.

2.2.19 MQDSPUBLICKEYS

The MQDSPUBLICKEYS structure defines a set of MQDSPUBLICKEY structures.



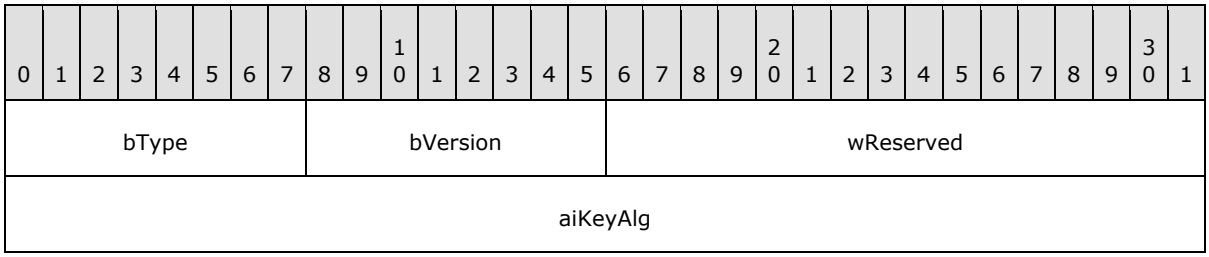
ulLen (4 bytes): An unsigned 32-bit integer that MUST contain the size in bytes of the MQDSPUBLICKEYS structure.

cNumofKeys (4 bytes): An unsigned 32-bit integer that MUST contain the count of MQDSPUBLICKEY structures in the array **aPublicKeys**.

aPublicKeys (variable): An array of MQDSPUBLICKEY structures.

2.2.20 BLOBHEADER

The BLOBHEADER structure indicates a key's binary large object (BLOB) type and the algorithm that the key uses.



bType (1 byte): An unsigned 8-bit integer that MUST contain the key BLOB type. The following key BLOB type is defined.<7>

Value	Meaning
PUBLICKEYBLOB 0x06	The key is a public key.

bVersion (1 byte): An unsigned 8-bit integer that contains the version number of the BLOBHEADER. The only value supported by this protocol is 0x02.

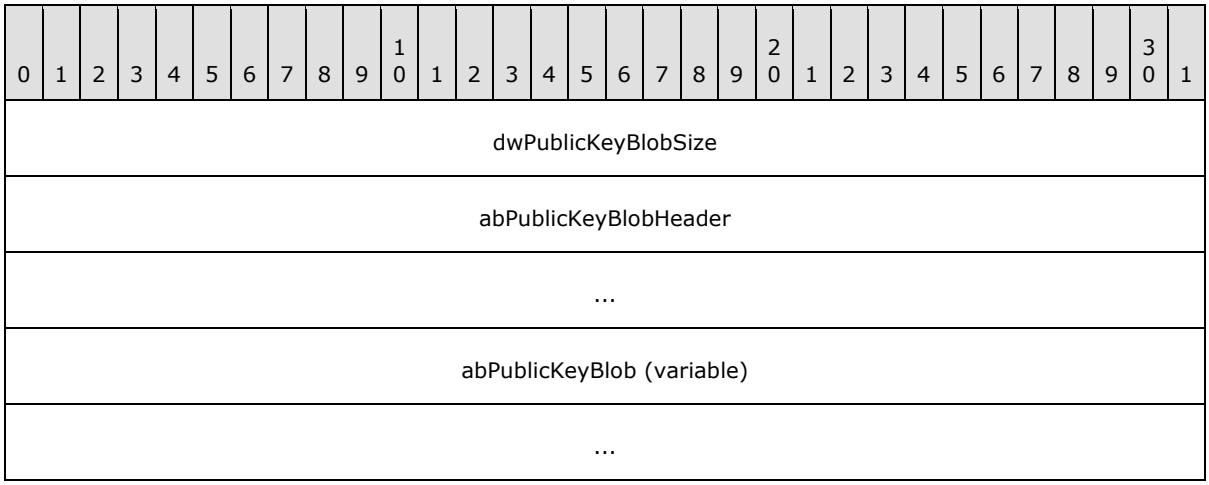
wReserved (2 bytes): An unsigned 16-bit integer that MUST be set to zero.

aiKeyAlg (4 bytes): An unsigned 32-bit integer that specifies the key algorithm. The following algorithm identifiers are defined.<8>

Value	Meaning
CALG_RSA_KEYX 0x0000a400	RSA public key exchange algorithm.
CALG_RSA_SIGN 0x00002400	RSA public key signature algorithm.

2.2.21 MQDS_PublicKey

The MQDS_PublicKey structure defines a public key certificate.



dwPublicKeyBlobSize (4 bytes): An unsigned 32-bit integer that MUST contain the size in bytes of the **abPublicKeyBlobHeader** and **abPublicKeyBlob** fields.

abPublicKeyBlobHeader (8 bytes): A buffer containing a BLOBHEADER structure.

abPublicKeyBlob (variable): A buffer containing a public key certificate formatted as an RSA_PUBLIC_KEY, as specified in [MS-RDPBCGR] section 2.2.1.4.3.1.1.1.

3 Protocol Details

The methods that comprise this RPC interface return MQ_OK (0x00000000) on success and a nonzero implementation-specific error code on failure. Unless otherwise specified, a server-side implementation of the Message Queuing (MSMQ): Directory Service Protocol may choose any nonzero Win32 error value to signify an error condition, as specified in section 1.8. Unless otherwise specified, the client side of the MSMQ: Directory Service Protocol MUST NOT interpret returned error codes, and MUST return the error codes to the invoking application without taking any protocol action.

Note that the phrases "client side" and "server side" refer to the initiating and receiving ends of the MSMQ: Directory Service Protocol, respectively, rather than to the client or server versions of an operating system. These methods MUST all behave in the same manner, whether the server side of the protocol is running on a client or server version of an operating system.

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

3.1 dscomm Server Details

3.1.1 Abstract Data Model

This section describes a conceptual model of a 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 server MUST maintain the GSS security context acquired through the **S_DSValidateServer** sequence. This security context is used during the mutual authentication handshake and subsequently in the generation of the signature in each protocol method that has an output signature parameter.

The server MUST retain internal state for the RPC context handle types that are shown in the following table.

RPC context handle	Description
PCONTEXT_HANDLE_SERVER_AUTH_TYPE	A table of negotiated security contexts.
PCONTEXT_HANDLE_TYPE	A table of in-progress queries.

3.1.1.1 Directory Service Objects

The directory defined by this protocol is composed of eight types of directory service object, representing enterprises, sites, site links, machines, users, queues, connected networks, and deleted objects. Seven of the eight directory service object types model elements in an MSMQ enterprise, while the deleted object type is used in replication of the directory between site controllers. The following diagram shows the concept of ownership between these directory service object types.

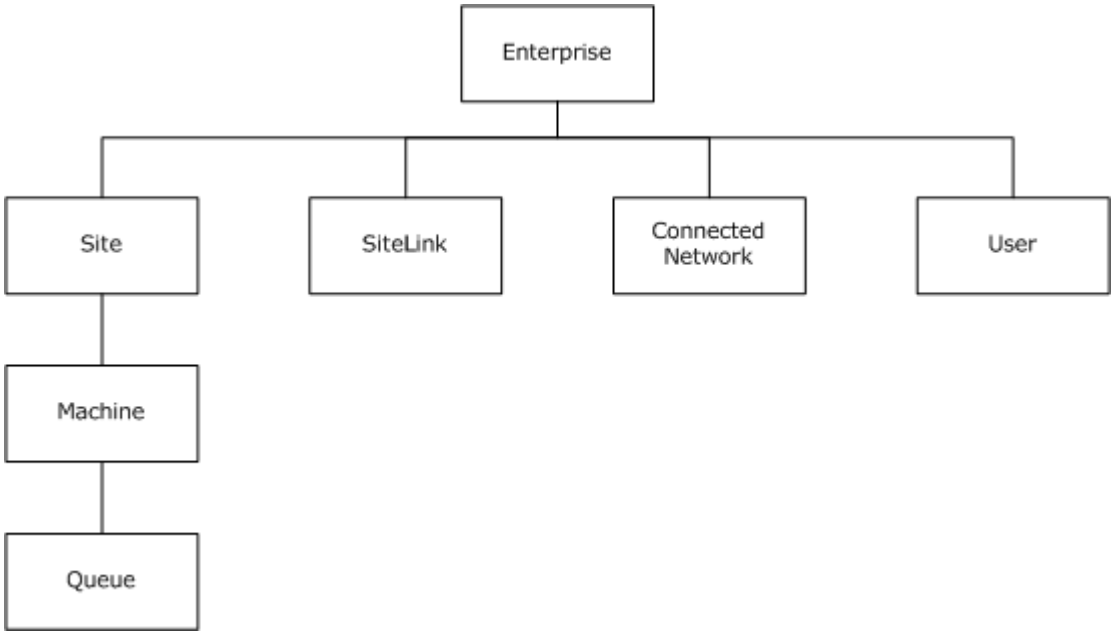


Figure 2: Directory service object type ownership

3.1.1.2 Directory Service Object Properties

Each directory service object type is composed of a collection of directory service object properties. These properties form the schema of the directory service object. Note that there is no type inheritance; each directory service object type stands alone. Directory service object properties are specific to the directory service object type.

Each directory service object property has an integer property identifier and a variant property value. Most of these directory service object properties are not interpreted by this protocol, but are passed from the directory service to the user without translation or other manipulations. Some directory service object properties, however, are used by this protocol and are called out specifically in section 2.2.10.

3.1.1.2.1 Directory Service Object Properties Specifying an Identifier

Most directory service object types include a unique identifier, assigned by the directory service at the time of directory service object creation and stored as a directory service object property.

The following table summarizes the directory service object properties that contain a unique identifier for a directory service object. The directory service object property value is a GUID.

Directory service object type	Directory service object property identifier
Queue	PROPID_Q_INSTANCE 101
Machine	PROPID_QM_MACHINE_ID 202
Site	PROPID_S_SITEID

Directory service object type	Directory service object property identifier
	302
Connected Network	PROPID_CN_GUID 503
Enterprise	PROPID_E_ID 609
User	PROPID_U_ID 706
Site Link	PROPID_L_ID 806

3.1.1.2.2 Directory Service Object Properties Specifying a Name

Many directory service objects include a textual name, intended to provide a mechanism for people to more easily distinguish between instances of directory service object types. These names are generally free-form, with no particular naming convention required by the system.

The following table summarizes the directory service object properties that contain a name for a directory service object. The directory service object property value is a NULL terminated Unicode string.

Directory service object type	Directory service object property identifier
Queue	PROPID_Q_PATHNAME 103
Machine	PROPID_QM_PATHNAME 203
Site	PROPID_S_PATHNAME 301
Connected Network	PROPID_CN_NAME 502
Enterprise	PROPID_E_NAME 601

3.1.1.3 Directory Service Object Relationships

A relationship between directory service objects is established either by storing the GUID or the name of a directory service object as a property of the related directory service object. For example, queues belong to a machine, and each queue directory service object stores the GUID of the machine directory service object that owns it as a directory service object property with identifier PROPID_Q_QMID. Machines belong to one or more sites, so each machine directory service object stores the GUIDs of the site directory service objects that own it as a directory service object property with identifier PROPID_QM_SITE_IDS. A site has a PSC, and the distinguished name of the machine that serves as the PSC is stored as a directory service object property of the site directory service object.

There is no explicit relationship between enterprise objects and the other object types; the enterprise object is presumed to be singular, and is retrieved by its name rather than by following property relations.

Note This protocol does not check referential integrity for these relationships.

The following diagram shows all the relationships which are explicitly modeled in the directory service object schema.

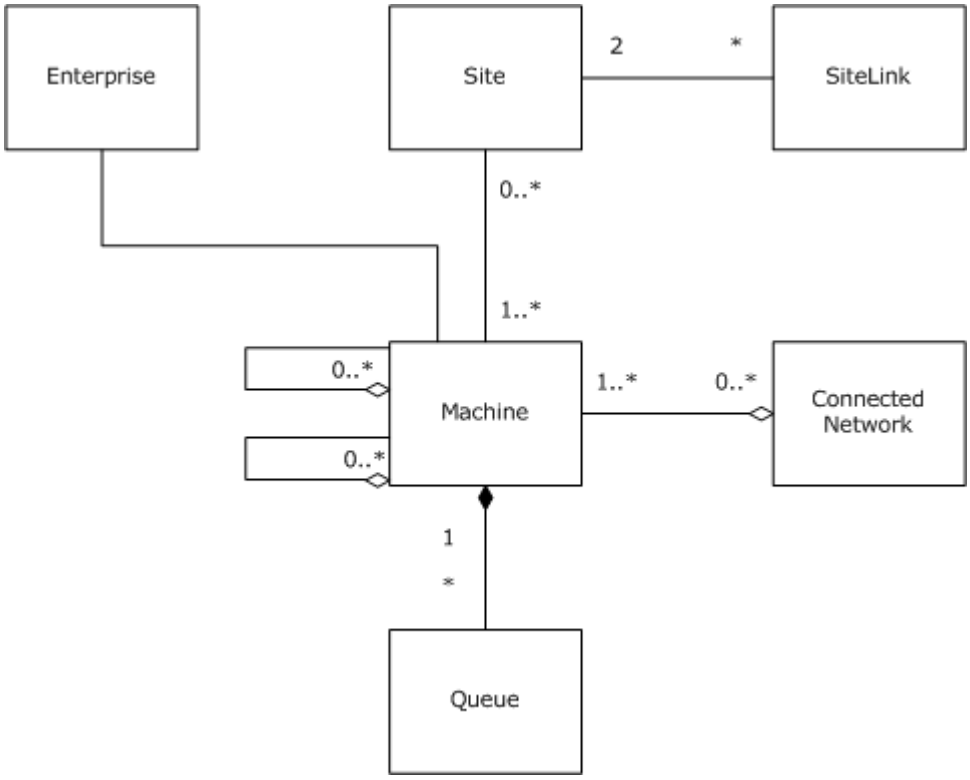


Figure 3: Directory service object schema relationships

The following table summarizes the relationships between directory service object types.

Directory service object type	Description of relationship	Related directory service object type	Directory service object property identifier
Queue	Is deployed on	Machine	PROPID_Q_QMID 115
Queue	Is deployed on	Machine	PROPID_Q_PATHNAME 103
Machine	Is member of	Site	PROPID_QM_SITE_ID 201
Machine	Is member of	Connected Network	PROPID_QM_CNS 207

Directory service object type	Description of relationship	Related directory service object type	Directory service object property identifier
Machine	Routes outgoing messages through	Machine	PROPID_QM_OUTFRS 208
Machine	Receives incoming messages routed through	Machine	PROPID_QM_INFRS 209
Site	PSC is deployed on	Machine	PROPID_S_PSC 304
Enterprise	PEC is deployed on	Machine	PROPID_E_PECNAME 604
Site link	Links site	Site	PROPID_L_NEIGHBOR1 801
Site link	Links sites	Site	PROPID_L_NEIGHBOR2 802

3.1.1.4 Directory Service Object Access Control

Each directory service object has an associated **access control list (ACL)** in the directory service. The ACL is generally assigned by the directory service, but may be assigned by the user. These ACLs are generally mutable.

Unlike other object properties, the ACLs are not retrievable or settable by a property identifier in the protocol. Instead, they are retrieved or set through dedicated RPC methods.

3.1.1.5 Directory Service Object Types

This section presents a summary of the different directory service object types.

3.1.1.5.1 Queue Directory Service Object

A queue directory service object captures information about a particular public queue. Private queues do not have representation in the directory service and consequently cannot be discovered through directory service means.

3.1.1.5.2 Machine Directory Service Object

A machine directory service object captures information about a machine that is part of an MSMQ installation. A machine directory service object is created for each machine that has an MSMQ queue manager installed on it, including each independent client, each supporting server, and each routing server.

3.1.1.5.3 Site Directory Service Object

A site directory service object captures information about a physical collection of machines in which communication between any two computers is considered fast and inexpensive.

3.1.1.5.4 Connected Network Directory Service Object

A connected network directory service object captures the concept of connectedness in that any two machines in the same connected network can directly communicate.

The enterprise object is found by constructing its directory service pathname from standardized components, rather than by following properties from other directory service objects.

3.1.1.5.5 Enterprise Directory Service Object

The enterprise object captures information applicable to all elements of the enterprise, and was originally intended to define the scope of the enterprise. It is currently deprecated and should not be used for new development.

The enterprise object is found by constructing its directory service pathname from standardized components, rather than by following properties from other directory service objects.

3.1.1.5.6 User Directory Service Object

The user directory service object captures information about a user of the system.

3.1.1.5.7 Site Link Directory Service Object

The site link directory service object captures information about the relative cost of communications between two sites. The concept of non-connectedness is modeled by a site link directory service object with a cost set to zero.

3.1.1.6 Data Organization

The abstract data model for the directory uses a relational database to store the directory information. The database consists of a table per directory service object type. The schema for each table consists of a field per directory service object property associated with the corresponding directory service object type. Each table has a GUID primary key corresponding to the GUID identifier, as specified in the table in section 3.1.1.2.1. An index is defined for each table in which the corresponding directory service object has a name property, as specified in the table in section 3.1.1.2.2. Each table defines an additional ACL field. This field corresponds to the ACL for the directory object. See section 3.1.1.4.

3.1.1.7 Data Partitioning

To provide scale, the directory is partitioned. There is one partition consisting of the tables containing the enterprise object, connected network objects, site objects, site link objects, and user objects. There is an additional partition per site consisting of the tables containing the machine objects and queue objects that belong to that site.

To support this partitioning, two directory service roles are defined. The first directory service role is Enterprise Controller, which maintains the enterprise partition. The second role is Site Controller, which maintains a site partition. A Site Controller exists for each site.

3.1.1.8 Data Replication

To provide high availability, partitions may be replicated. Multiple Site Controllers can exist within a site, each of which has a replicated copy of the enterprise partition and the site partition. How the data is replicated is outside the scope of this specification.

Due to latency of replication, there is no assurance that a change to the directory will be immediately visible to all clients. Clients of the protocol must be aware of this limitation.

3.1.1.9 Negotiation Token

The server MUST maintain the security context acquired through the **S_DSValidateServer** sequence. This security context is used during the mutual authentication handshake and subsequently in the generation of the signature in each protocol method that has an output signature parameter.

The server MUST retain internal state for the RPC context handle types that are shown in the following table.

RPC context handle	Description
PCONTEXT_HANDLE_SERVER_AUTH_TYPE	A table of negotiated security contexts.
PCONTEXT_HANDLE_TYPE	A table of in-progress queries.

3.1.1.10 PCONTEXT_HANDLE_SERVER_AUTH_TYPE RPC Context Handle

The **PCONTEXT_HANDLE_SERVER_AUTH_TYPE** represents the server's security context established with a client.

A protocol client acquires a **SERVER_AUTH_STRUCT** RPC context handle through a call to the **S_DSValidateServer** method and releases the context handle through a subsequent call to **S_DSCloseServerHandle**.

Each instance of this RPC context handle represents internal state that the server needs to maintain. The server SHOULD register a rundown method to close these RPC context handles in the event that the client fails to call the **S_DSCloseServerHandle** method. See section 3.1.6.1.

The server MUST retain this information until the client closes the RPC context handle through a call to the **S_DSCloseServerHandle** method, or until the RPC context handle rundown occurs.

3.1.1.11 PCONTEXT_HANDLE_TYPE RPC Context Handle

The **PCONTEXT_HANDLE_TYPE** represents the server's state associated with an in-progress directory query.

A protocol client acquires a **PCONTEXT_HANDLE_TYPE** RPC context handle through a call to the **S_DSLookupBegin** method and releases the context handle through a call to **S_DSLookupEnd**.

Each instance of this RPC context handle represents internal state that the server needs to maintain. The server SHOULD register a rundown method to close these RPC context handles in the event that the client fails to call the **S_DSLookupEnd** method. See section 3.1.6.2.

The server MUST retain the following state for each in-progress directory query operation.

- The result set MUST be the set of directory objects that satisfy the query specified by the client. In particular:
  - The objects returned MUST satisfy all of the restrictions in the *pRestriction* parameter.
  - The properties returned for each object MUST be limited to only those specified in the *pColumns* parameter.

- The objects MUST be returned in the order specified in the *pSort* parameter.

- An index into the result set that represents the directory object returned in the previous call to **S_DSLookupNext**.

For more information, see **S_DSLookupBegin** (section 3.1.4.17).

The server MUST retain this information until the client closes the context handle through a call to the **S_DSLookupEnd** method, or until the RPC context handle rundown occurs.

3.1.2 Timers

No protocol timers are required beyond those used internally by RPC to implement resiliency to network outages. See Remote Procedure Call Protocol Extensions, as specified in [MS-RPCE].

3.1.3 Initialization

Parameters necessary to initialize the RPC protocol are specified in section 2.1.

The server MAY<9> register static RPC ports. If it does, it MUST return those ports in a call to the **S_DSGetServerPort** method. For more information, see section 3.1.4.1.

3.1.4 Message Processing Events and Sequencing Rules

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

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

The server SHOULD enforce appropriate security measures to make sure that the caller has the required permissions to execute the following routines.

The methods specified below MUST NOT throw exceptions.

The following methods comprise the dscomm server interface:

Methods in RPC Opnum Order

Method	Description
S_DSCreateObject	Creates a directory object. Opnum: 0
S_DSDeleteObject	Deletes a directory object specified by path name. Opnum: 1
S_DSGetProps	Returns properties associated with a directory object specified by path name. Opnum: 2
S_DSSetProps	Sets properties associated with a directory object specified by path name. Opnum: 3

Method	Description
S_DSGetObjectSecurity	Returns security properties associated with a directory object specified by path name. Opnum: 4
S_DSSetObjectSecurity	Sets security properties associated with a directory object specified by path name. Opnum: 5
S_DSLookupBegin	Begins a directory query to enumerate directory objects matching selection criteria. Opnum: 6
S_DSLookupNext	Returns the next item in a directory query result set. Opnum: 7
S_DSLookupEnd	Ends a directory query and closes the RPC context handle acquired from a previous call to **S_DSLookupBegin**. Opnum: 8
Opnum9NotUsedOnWire	Reserved for local use. Opnum: 9
S_DSDeleteObjectGuid	Deletes a directory object specified by object identifier. Opnum: 10
S_DSGetPropsGuid	Returns properties associated with a directory object specified by object identifier. Opnum: 11
S_DSSetPropsGuid	Sets property identifiers associated with a directory object specified by object identifier. Opnum: 12
S_DSGetObjectSecurityGuid	Returns security properties associated with a directory object specified by object identifier. Opnum: 13
S_DSSetObjectSecurityGuid	Sets security properties associated with a directory object specified by object identifier. Opnum: 14
Opnum15NotUsedOnWire	Reserved for local use. Opnum: 15
Opnum16NotUsedOnWire	Reserved for local use. Opnum: 16
Opnum17NotUsedOnWire	Reserved for local use. Opnum: 17
Opnum18NotUsedOnWire	Reserved for local use. Opnum: 18

Method	Description
S_DSQMSetMachineProperties	Sets the properties of the MSMQ machine object in the directory service. Opnum: 19
S_DSCreateServersCache	Returns a list of BSCs associated with the site the caller is in. Opnum: 20
S_DSQMGetObjectSecurity	Returns the security properties for the specified machine object. Opnum: 21
S_DSValidateServer	Initiates an authentication handshake to authenticate the directory server to the client. Opnum: 22
S_DSCloseServerHandle	Closes an RPC context handle acquired from a previous call to **S_DSValidateServer**. Opnum: 23
Opnum24NotUsedOnWire	Reserved for local use. Opnum: 24
Opnum25NotUsedOnWire	Reserved for local use. Opnum: 25
Opnum26NotUsedOnWire	Reserved for local use. Opnum: 26
S_DSGetServerPort	Returns the port number assigned for the RPC dynamic endpoint. Opnum: 27

In the preceding table, the phrase "Reserved for local use" means that the client MUST NOT send the **opnum** and that the server behavior is undefined<10> because it does not affect interoperability.

3.1.4.1 S_DSGetServerPort (Opnum 27)

The **S_DSGetServerPort** method returns the RPC endpoint port for a transport protocol. The client establishes a new binding to the server by using the returned port number.

```
unsigned long S_DSGetServerPort(
  [in] handle_t hBind,
  [in] unsigned long fIP
);
```

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

fIP: Specifies the connected network protocol for which an RPC endpoint port is to be returned.

Value	Meaning
0xFFFFFFFF	Causes a value of 0 to be returned.

Value	Meaning
0x00000000	Causes the RPC endpoint port for an RPC over SPX protocol sequence, as specified in [MS-RPCE], to be returned.
0x00000001 — 0xFFFFFFFE	Causes the RPC endpoint port for an RPC over TCP/IP protocol sequence, as specified in [MS-RPCE], to be returned.

Return Values: If the method succeeds, the return value is the RPC endpoint port number. If the method fails, the server MUST return 0.

0x00000000

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

The server MUST support the RPC over TCP/IP protocol sequence. The server MAY support the RPC over SPX protocol sequence.<11> If the server does not support the RPC over SPX protocol sequence, the server MUST return 0 when the *fIP* parameter is zero.

3.1.4.2 S_DSValidateServer (Opnum 22)

The **S_DSValidateServer** method performs mutual authentication between the client and server, and establishes a security context, as specified in [RFC2743]. This security context is used by the server to construct a digital signature in subsequent method calls of this protocol, and it is used by the client to validate the digital signature. The digital signature is used in methods that return data to the client. It allows the client to authenticate the source of the data and ensures that it has not been tampered with enroute from the server to the client.

When the client has no further use of the RPC context handle returned from this method, it releases the handle through a subsequent call to **S_DSCloseServerHandle**.

```
HRESULT S_DSValidateServer(
  [in] handle_t hBind,
  [in] const GUID* pguidEnterpriseId,
  [in] long fSetupMode,
  [in] unsigned long dwContext,
  [in] unsigned long dwClientBuffMaxSize,
  [in, size_is (dwClientBuffMaxSize),
         length_is(dwClientBuffSize)]
    unsigned char* pClientBuff,
  [in] unsigned long dwClientBuffSize,
  [out] PPCONTEXT_HANDLE_SERVER_AUTH_TYPE pphServerAuth
);
```

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

pguidEnterpriseId: MUST be set by the client to a pointer to the globally unique identifier (GUID) of the enterprise that owns the server. The server SHOULD ignore it.<12>

fSetupMode: Boolean value that defines the setup mode. Clients SHOULD set this value to FALSE (0x00000000). The server SHOULD ignore it.<13>

dwContext: MUST be set by the client to a value that the client can use to correlate callbacks with the initial call to **S_DSValidateServer**. The server MUST supply this value in the *dwContext* parameter in the related calls of the **S_InitSecCtx** callback method.

dwClientBuffMaxSize: MUST be set by the client to the size of the buffer pointed to by *pClientBuff*.

pClientBuff: Pointer MUST be set by the client to point to a buffer that contains the output_token from an initial call to GSS_Init_sec_context, as specified in [RFC2743] section 2.2.1.

dwClientBuffSize: MUST be set by the client to the size in bytes of the output_token within the client buffer pointed at by *pClientBuff*.

pphServerAuth: MUST be set by the server to a **PCONTEXT_HANDLE_SERVER_AUTH_TYPE** RPC context handle.

Return Values: If the method succeeds, the return value is 0. If the method fails, the return value is an implementation-specific error code.

MQ_OK (0x00000000)

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

If the *dwClientBuffSize* parameter is zero, the server MUST return a zero-filled **SERVER_AUTH_STRUCT** in the *pphServerAuth* parameter and MUST NOT invoke client callback methods. In addition, all signature parameters in other methods in this protocol that return server signatures MUST be set to all zeros.

If the *dwClientBuffSize* parameter is nonzero, the server MUST return a valid non-null **SERVER_AUTH_STRUCT** in the *pphServerAuth* parameter.

The client MUST supply an input_token in *pClientBuff*, acquired through an initial call to GSS_Init_sec_context, as specified in [RFC2743] section 2.2.1. The server MUST perform the following processing.

1. Call GSS_Accept_sec_context, as specified in [RFC2743] section 2.2.2, with the input_token specified by the client and a NULL input_context_handle. If GSS_Accept_sec_context returns GSS_S_COMPLETE, continue at step 5 below.

2. If GSS_Accept_sec_context returns GSS_S_CONTINUE_NEEDED, the server MUST issue a callback to the client through the **S_InitSecCtx** method with *dwContext* set to the value supplied by the client in the *dwContext* parameter and *pServerBuff* set to the output_token returned from GSS_Accept_sec_context.

3. When the callback to **S_InitSecCtx** returns, the server MUST pass the input_token returned in *pClientBuff* to GSS_Accept_sec_context, as specified in [RFC2743] section 2.2.2. If GSS_Accept_sec_context returns GSS_S_COMPLETE, continue at step 5 below.

4. If GSS returns GSS_S_CONTINUE_NEEDED, the server MUST repeat this sequence starting at step 2, setting *pServerBuff* to the output_token received from the last call to GSS_Accept_sec_context.

5. If GSS_Accept_sec_context returns GSS_S_COMPLETE, the negotiation is complete. The server MUST allocate a **PCONTEXT_HANDLE_SERVER_AUTH_TYPE** RPC context handle, and MUST

allocate a security context entry in the security table keyed by the context handle. The server MUST associate the GSS security context (output_context_handle from the GSSAPI call) with the security context entry. The server MUST return the RPC context handle in *pphServerAuth*.

The GSS security context is used by the server in subsequent calls to GSSWrap, as specified in [RFC2743] section 2.3.3.

On successful return, the client MUST retrieve the GSS security context associated with dwContext, and MUST associate it with the **PCONTEXT_HANDLE_SERVER_AUTH_TYPE** RPC context handle returned in *pphServerAuth*. The GSS security context will be used by the client in subsequent calls to GSSUnwrap, as specified in [RFC2743] section 2.3.4.

3.1.4.3 S_DSCloseServerHandle (Opnum 23)

The **S_DSCloseServerHandle** method closes the RPC context handle returned by a previous call to **S_DSValidateServer**. The server releases resources associated with the RPC context handle.

```
HRESULT S_DSCloseServerHandle(
  [in, out] PPCONTEXT_HANDLE_SERVER_AUTH_TYPE pphServerAuth
);
```

pphServerAuth: The **PCONTEXT_HANDLE_SERVER_AUTH_TYPE** RPC context handle to close. It MUST have been acquired from the *pphServerAuth* parameter in a previous call to **S_DSValidateServer**, and MUST NOT have been closed through a previous call to **S_DSCloseServerHandle**. The server MUST set this parameter to NULL.

Return Values: If the method succeeds, the return value is 0. If the method fails, the return value is an implementation-specific error code.

MQ_OK (0x00000000)

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

The server MUST use the *pphServerAuth* parameter as a key into the security context table. The server MUST delete the GSS security context associated with the security context entry through a call to GSS_Delete_sec_context, as specified in [RFC2743] section 2.2.3. The server MUST remove the security context entry from the security context table.

On return from this call, the client SHOULD delete the GSS security context associated with dwContext through a call to GSS_Delete_sec_context, as specified in [RFC2743] section 2.2.3.

3.1.4.4 S_DSCreateObject (Opnum 0)

The **S_DSCreateObject** method creates a new directory object, assigns the specified properties and security descriptor to that directory object, and returns a unique **GUID** identifier for that directory object.

```
HRESULT S_DSCreateObject(
  [in] handle_t hBind,
  [in] unsigned long dwObjectType,
  [in, string, unique] const wchar_t* pwcsPathName,
  [in] unsigned long dwSDLength,
  [in,size_is(dwSDLength), unique]
```

```
    char* SecurityDescriptor,
  [in] unsigned long cp,
  [in, size_is(cp)] unsigned long aProp[],
  [in, size_is(cp)] PROPVARIANT apVar[],
  [in, out, unique] GUID* pObjGuid
);
```

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

dwObjectType: Specifies the type of directory object to create. MUST be set to one of the object types, as specified in section 2.2.9.

pwcsPathName: Pointer to a NULL-terminated **UNICODE_STRING** that MUST contain the pathname for the object to be created in the directory service.

dwSDLength: MUST contain the size in bytes of the buffer pointed to by *SecurityDescriptor*.

SecurityDescriptor: MUST contain a pointer to a security descriptor, as specified in [MS-DTYP] section **2.4.6**, or NULL.

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

aProp: An array of directory object property identifiers of properties to associate with the created object. Each element MUST specify a value from the property identifiers table as specified in section 2.2.10.1, for the directory object type specified in *dwObjectType*. Each element MUST specify the property identifier for the corresponding property value at the same element index in *apVar*. The array MUST contain at least one element.

apVar: An array of property values to associate with the created object. Each element MUST specify the property value for the corresponding property identifier at the same element index in *aProp*. The array MUST contain at least one element.

pObjGuid: MUST be set by the server to the **GUID** of the created object.

Return Values: If the method succeeds, the return value is 0. If the method fails, the return value is an implementation-specific error code.

MQ_OK (0x00000000)
MQ_ERROR_ILLEGAL_PROPERTY_VALUE (0xC00E0018)
MQ_ERROR_ILLEGAL_ENTERPRISE_OPERATION (0xC00E0071)

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

For each property identifier in *aProp*, the server MUST verify that the property identifier is valid for the directory object type specified in *dwObjectType*, and that the corresponding variant in *apVar* is of the type defined for the property identifier, as specified in the tables in section 2.2.10.1.

The server response depends on the *dwObjectType* parameter value. If the value of this parameter is unsupported, the server MUST NOT execute the call and MUST return an error.<14>

If the *dwObjectType* is not MQDS_QUEUE, the *SecurityDescriptor* parameter SHOULD be null.<15>
If the *dwObjectType* parameter is not equal to *MQDS_QUEUE*, and the *SecurityDescriptor* parameter is not NULL, the server SHOULD return an error.<16>

If the *dwObjectType* is MQDS_QUEUE, the *SecurityDescriptor* parameter MUST NOT be null.

If the *dwObjectType* parameter is equal to MQDS_USER, and the *SecurityDescriptor* parameter is NULL, the server MUST construct a default security descriptor with the owner SID set to the SID of the calling user. If the *SecurityDescriptor* parameter is not NULL, the server SHOULD return MQ_ERROR_ILLEGAL_PROPERTY_VALUE (0xC00E0018) but MAY ignore the parameter and construct a security descriptor with the owner SID set to the SID of the calling user.<17>

If the *dwObjectType* parameter is not equal to MQDS_QUEUE, MQDS_ENTERPRISE, or MQDS_USER and the *SecurityDescriptor* parameter is NULL, the server MUST construct a default security descriptor with the owner SID set to the SID of the calling user. If the *SecurityDescriptor* parameter is not NULL, the server SHOULD return MQ_ERROR_ILLEGAL_PROPERTY_VALUE (0xC00E0018) but MAY use the supplied security descriptor.<18>

If the *dwObjectType* parameter is equal to MQDS_MACHINE, the client MUST include public key certificates formatted as an MQDSPUBLICKEYS structure, as specified in section 2.2.19, associated with property identifier PROPID_QM_SIGN_PK. At least one of these public keys MUST have a provider type code equal to PROV_RSA_FULL (0x00000001) with a provider name equal to "Microsoft Base Cryptographic Provider 1.0". This key will be used to verify the signature on changes to the associated machine object.

The server MUST create and assign a new **GUID** to the created object and MUST set the *pObjGuid* parameter to the assigned **GUID**. The server MUST assign an object property to the object, setting the property type to VT_CLSID, the property value to the created **GUID**, and the property identifier according to the *dwObjectType* as shown in the following table.

Object type	Object property identifier
MQDS_QUEUE	PROPID_Q_INSTANCE
MQDS_MACHINE	PROPID_QM_MACHINE_ID
MQDS_SITE	PROPID_S_SITEID
MQDS_CN	PROPID_CN_GUID
MQDS_ENTERPRISE	PROPID_E_ID
MQDS_USER	PROPID_U_ID
MQDS_SITELINK	PROPID_L_ID

The server MUST associate the security descriptor with the created object in the directory service.

All other properties MUST be provided by the client. If the client does not provide all required object type properties, the server MUST NOT execute the call and MUST return an error.

3.1.4.5 S_DSDeleteObject (Opnum 1)

The **S_DSDeleteObject** method deletes a directory object specified by a path name.

```
HRESULT S_DSDeleteObject(
  [in] handle_t hBind,
  [in] unsigned long dwObjectType,
  [in, string] const wchar_t* pwcsPathName
);
```

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

dwObjectType: Specifies the type of object to delete. MUST be set to one of the directory object types specified in section 2.2.9.

pwcsPathName: Pointer to a NULL-terminated **UNICODE_STRING** that MUST contain the pathname to the object in the directory service.

Return Values: If the method succeeds, the return value is 0. If the method fails, the return value is an implementation-specific error code.

MQ_OK (0x00000000)

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

The server SHOULD enforce appropriate security measures to make sure that the caller has the required permissions to execute this routine. If the server enforces security measures, and the caller does not have the required credentials, the server MUST return an error.

3.1.4.6 S_DSDeleteObjectGuid (Opnum 10)

The **S_DSDeleteObjectGuid** method deletes a directory object specified by an object identifier.

```
HRESULT S_DSDeleteObjectGuid(
  [in] handle_t hBind,
  [in] unsigned long dwObjectType,
  [in] const GUID* pGuid
);
```

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

dwObjectType: Specifies the type of object to delete. MUST be set to one of the object types specified in section 2.2.9.

pGuid: A pointer to the globally unique identifier **GUID** of the object to delete. This MUST be the identifier assigned to the object by the server when the object was created. See section 3.1.4.1.

Return Values: If the method succeeds, the return value is 0. If the method fails, the return value is an implementation-specific error code.

MQ_OK (0x00000000)

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

3.1.4.7 S_DSGetProps (Opnum 2)

The **S_DSGetProps** method returns the properties associated with a directory object specified by path name.

```
HRESULT S_DSGetProps(
```

```
  [in] handle_t hBind,
  [in] unsigned long dwObjectType,
  [in, string] const wchar_t* pwcsPathName,
  [in] unsigned long cp,
  [in, size_is(cp)] unsigned long aProp[],
  [in, out, size_is(cp)] PROPVARIANT apVar[],
  [in] PCONTEXT_HANDLE_SERVER_AUTH_TYPE phServerAuth,
  [out, size_is(*pdwServerSignatureSize)]
    unsigned char* pbServerSignature,
  [in, out] unsigned long* pdwServerSignatureSize
);
```

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

dwObjectType: Specifies the type of object for which properties are to be retrieved. MUST be set to one of the object types specified in section 2.2.9.

pwcsPathName: Pointer to a NULL-terminated **UNICODE_STRING** that MUST contain the pathname to the object in the directory service.

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

aProp: An array of property identifiers specifying the set of object properties to be returned. Each element MUST specify a value from the property identifiers table defined in section 2.2.10.1 for the object type specified in *dwObjectType*. Each element MUST specify the property identifier for the corresponding property value at the same element index in *apVar*. MUST contain at least one element.

apVar: An array that the server sets to the property values associated with the object. Each element MUST specify the property value for the corresponding property identifier at the same element index in *aProp*. MUST contain at least one element.

phServerAuth: A **PCONTEXT_HANDLE_SERVER_AUTH_TYPE** RPC context handle acquired from the *pphServerAuth* parameter in a previous call to **S_DSValidateServer**. The server MUST use this parameter as a key to locate the GSS security context used to compute the signature returned in pbServerSignature. See section 3.1.4.2.

pbServerSignature: MUST be set by the server to a buffer that contains a signed hash over the returned property values. The server MUST construct this signature by creating a hash by using the MD5 algorithm, as specified in [RFC1321], and sealing it, as specified by the following pseudocode:

```
Initialize an MD5 hash context

Add to the hash context the DWORD data value representing the count
       of properties to be returned (from parameter cp).

FOR each property returned in apVar

    Add to the hash context the DWORD value of the property
        identifier (from aProp[])

    Add to the hash context the type-specific data value of the
        property (from apVar[]). The data value and length are
```

```
        defined by the variant type of the property (apVar[].vt)
    ENDFOR

    Call GSS_Wrap using the token from output context handle from the
    GSS security context and the computed MD5 hash

    Set pbServerSignature to the wrapped MD5 hash

    Set *pdwServerSignatureSize to the size of the wrapped MD5 hash
```

pdwServerSignatureSize: MUST be set by the client to point to a **DWORD** that contains the maximum length in bytes of the server signature to return. MUST be set by the server to the actual length in bytes of the server signature on output. If the server signature is larger than the supplied buffer, the server MUST return MQ_ERROR_USER_BUFFER_TOO_SMALL (0xC00E0028).

Return Values: If the method succeeds, the return value is 0. If the method fails, the return value is an implementation-specific error code.

MQ_OK (0x00000000)
MQ_ERROR_USER_BUFFER_TOO_SMALL (0xC00E0028)

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

For each property identifier in *aProp*, the server MUST verify that the property identifier is valid for the object type specified in *dwObjectType* as specified in the tables in section 2.2.10.1.

3.1.4.8 S_DSGetPropsGuid (Opnum 11)

The **S_DSGetPropsGuid** method returns properties associated with a directory object specified by an object identifier.

```
HRESULT S_DSGetPropsGuid(
  [in] handle_t hBind,
  [in] unsigned long dwObjectType,
  [in, unique] const GUID* pGuid,
  [in] unsigned long cp,
  [in, size_is(cp)] unsigned long aProp[],
  [in, out, size_is(cp)] PROPVARIANT apVar[],
  [in] PCONTEXT_HANDLE_SERVER_AUTH_TYPE phServerAuth,
  [out, size_is(*pdwServerSignatureSize)]
    unsigned char* pbServerSignature,
  [in, out] unsigned long* pdwServerSignatureSize
);
```

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

dwObjectType: Specifies the type of object for which properties are retrieved. MUST be set to one of the object types specified in section 2.2.9.

pGuid: MUST be set by the client to a pointer to the **GUID** of the object for which properties are retrieved.

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

aProp: An array of identifiers of properties to retrieve from the object designated by pGuid. Each element MUST specify a value from the property identifiers table (defined in section 2.2.10.1) for the object type specified in *dwObjectType*. Each element MUST specify the property identifier for the corresponding property value at the same element index in *apVar*. MUST contain at least one element.

apVar: An array that the server sets to the property values associated with the object identified by pGuid. Each element MUST specify the property value for the corresponding property identifier at the same element index in *aProp*. MUST contain at least one element.

phServerAuth: A **PCONTEXT_HANDLE_SERVER_AUTH_TYPE** RPC context handle acquired from the *pphServerAuth* parameter in a previous call to **S_DSValidateServer**. The server MUST use this parameter as a key to locate the GSS security context used to compute the signature returned in pbServerSignature. See section 3.1.4.2.

pbServerSignature: MUST point to the signed hash of the property values. See the *pbServerSignature* parameter description, as specified in section 3.1.4.7.

pdwServerSignatureSize: MUST be set by the client to point to a **DWORD** that contains the maximum length in bytes of the server signature to return. MUST be set by the server to the actual length in bytes of the server signature on output. If the server signature is larger than the supplied buffer, the server MUST return MQ_ERROR_USER_BUFFER_TOO_SMALL (0xC00E0028).

Return Values: If the method succeeds, the return value is 0. If the method fails, the return value is an implementation-specific error code.

MQ_OK (0x00000000)
MQ_ERROR_USER_BUFFER_TOO_SMALL (0xC00E0028)

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

For each property identifier in *aProp*, the server MUST verify that the property identifier is valid for the object type specified in *dwObjectType*, and that the corresponding variant in *apVar* is of the type defined for the property identifier, as specified in the tables in section 2.2.10.1.<19>

3.1.4.9 S_DSSetProps (Opnum 3)

The **S_DSSetProps** method sets the specified properties for a directory object specified by a pathname.

```
HRESULT S_DSSetProps(
  [in] handle_t hBind,
  [in] unsigned long dwObjectType,
  [in, string] const wchar_t* pwcsPathName,
  [in] unsigned long cp,
  [in, size_is(cp)] unsigned long aProp[],
  [in, size_is(cp)] PROPVARIANT apVar[]
);
```

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

dwObjectType: MUST specify the type of object for which properties are to be set. For supported object types, see the table of object types specified in section 2.2.9.

pwcsPathName: Pointer to a NULL-terminated **UNICODE_STRING** that MUST contain the pathname to the object in the directory service.

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

aProp: An array of identifiers of properties to associate with the object identified by *pwcsPathName*. Each element MUST specify a value from the property identifiers table defined in section 2.2.10.1 for the object type specified in *dwObjectType*. Each element MUST specify the property identifier for the corresponding property value at the same element index in *apVar*. The array MUST contain at least one element.

apVar: An array that specifies the property values to associate with the created object. Each element MUST specify the property value for the corresponding property identifier at the same element index in *aProp*. The array MUST contain at least one element.

Return Values: If the method succeeds, the return value is 0. If the method fails, the return value is an implementation-specific error code.

MQ_OK (0x00000000)
MQ_ERROR_ILLEGAL_PROPID (0xC00E0039)

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

For each property identifier in *aProp*, the server MUST verify that the property identifier is valid for the object type specified in *dwObjectType*, and that the corresponding variant in *apVar* is of the type defined for the property identifier, as specified in the tables in section 2.2.10.1.<20>

3.1.4.10 S_DSSetPropsGuid (Opnum 12)

The **S_DSSetPropsGuid** method sets properties for a directory object specified by an object identifier.

```
HRESULT S_DSSetPropsGuid(
  [in] handle_t hBind,
  [in] unsigned long dwObjectType,
  [in] const GUID* pGuid,
  [in] unsigned long cp,
  [in, size_is(cp)] unsigned long aProp[],
  [in, size_is(cp)] PROPVARIANT apVar[]
);
```

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

dwObjectType: Specifies the type of object for which properties are to be set. MUST be one of the object types specified in section 2.2.9.

pGuid: Pointer to the **GUID** of the object for which properties are to be set.

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

aProp: An array of identifiers of properties to associate with the object designated by *pGuid*. Each element MUST specify a value from the property identifiers table, as specified in section 2.2.10.1, for the object type specified in *dwObjectType*. Each element MUST specify the property identifier for the corresponding property value at the same element index in *apVar*. MUST contain at least one element.

apVar: An array that specifies the property values to associate with the created object. Each element MUST specify the property value for the corresponding property identifier at the same element index in *aProp*. MUST contain at least one element.

Return Values: If the method succeeds, the return value is 0. If the method fails, the return value is an implementation-specific error code.

MQ_OK (0x00000000)
MQ_ERROR_ILLEGAL_PROPID (0xC00E0039)

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

For each property identifier in *aProp*, the server MUST verify that the property identifier is valid for the object type specified in *dwObjectType*, and that the corresponding variant in *apVar* is of the type defined for the property identifier, as specified in the tables in section 2.2.10.1.<21>

3.1.4.11 S_DSGetObjectSecurity (Opnum 4)

The **S_DSGetObjectSecurity** method gets security properties for a directory object specified by a path name.

```
HRESULT S_DSGetObjectSecurity(
  [in] handle_t hBind,
  [in] unsigned long dwObjectType,
  [in, string] const wchar_t* pwcsPathName,
  [in] unsigned long SecurityInformation,
  [out, size_is(nLength)] unsigned char* pSecurityDescriptor,
  [in] unsigned long nLength,
  [out] unsigned long* lpnLengthNeeded,
  [in] PCONTEXT_HANDLE_SERVER_AUTH_TYPE phServerAuth,
  [out, size_is(*pdwServerSignatureSize)]
    unsigned char* pbServerSignature,
  [in, out] unsigned long* pdwServerSignatureSize
);
```

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

dwObjectType: Specifies the type of object for which security properties are to be retrieved. MUST be set to one of the object types defined in section 2.2.9.

pwcsPathName: Pointer to a NULL-terminated **UNICODE_STRING** that MUST contain the path to the object in the directory service.

SecurityInformation: A bitwise mask of the information to be returned in the *pSecurityDescriptor* parameter. The bit fields are defined by the following table:

Value	Meaning
OWNER_SECURITY_INFORMATION 0x00000001	**OWNER** field from the security descriptor.
GROUP_SECURITY_INFORMATION 0x00000002	**GROUP** field from the security descriptor.
DACL_SECURITY_INFORMATION 0x00000004	**Discretionary ACL** field from the security descriptor.
SACL_SECURITY_INFORMATION 0x00000008	**System ACL** field from the security descriptor.
MQDS_SIGN_PUBLIC_KEY 0x80000000	Signing public key. See the table in the *pSecurityDescriptor* parameter.
MQDS_KEYX_PUBLIC_KEY 0x40000000	Encrypting public key. See the table in the *pSecurityDescriptor* parameter.

The *SecurityInformation* parameter MUST specify one of:

- MQDS_SIGN_PUBLIC_KEY, or
- MQDS_KEYX_PUBLIC_KEY, or
- A bitwise OR of one or more of:
  - OWNER_SECURITY_INFORMATION
  - GROUP_SECURITY_INFORMATION
  - DACL_SECURITY_INFORMATION
  - SACL_SECURITY_INFORMATION

If the *SecurityInformation* parameter includes an invalid combination, the server MUST NOT complete the call, and MUST return an error.

pSecurityDescriptor: MUST be set by the server to contain a pointer to a security descriptor, as specified in [MS-DTYP] section **2.4.6**, or to a BLOBHEADER structure followed by an RSA_PUBLIC_KEY structure <22>, depending on the *SecurityInformation* parameter.

The returned security descriptor MUST include the information specified by the *SecurityInformation* parameter.

The source of the information returned in the parameter is a function of *dwObjectType* and *SecurityInformation* as follows.

dwObjectType	SecurityInformation	pSecurityDescriptor contains
MQDS_MACHINE (0x00000002)	MQDS_KEYX_PUBLIC_KEY (0x40000000)	Public key BLOB retrieved from property PROPID_QM_ENCRYPT_PK, formatted as a BLOBHEADER structure followed by an RSA_PUBLIC_KEY structure.

dwObjectType	SecurityInformation	pSecurityDescriptor contains
MQDS_MACHINE (0x00000002)	MQDS_SIGN_PUBLIC_KEY (0x80000000)	Public key BLOB retrieved from property PROPID_QM_SIGN_PK, formatted as a BLOBHEADER structure followed by an RSA_PUBLIC_KEY structure.
MQDS_MACHINE (0x00000002)	Bitwise OR of one or more of OWNER_SECURITY_INFORMATION, GROUP_SECURITY_INFORMATION, SACL_SECURITY_INFORMATION, and DACL_SECURITY_INFORMATION.	Security descriptor associated with the machine object in the directory service.
MQDS_SITE (0x00000003)	MQDS_SIGN_PUBLIC_KEY (0x80000000)	Public key BLOB retrieved from property PROPID_S_PSC_SIGNPK, as specified in section 2.2.10.4, formatted as a BLOBHEADER structure followed by an RSA_PUBLIC_KEY structure.
MQDS_SITE (0x00000003)	Bitwise OR of one or more of OWNER_SECURITY_INFORMATION, GROUP_SECURITY_INFORMATION, SACL_SECURITY_INFORMATION, and DACL_SECURITY_INFORMATION.	Security descriptor associated with the site object in the directory service.
MQDS_QUEUE (0x00000001)	Any other legal value.	Security descriptor associated with the queue object in the directory service.
MQDS_MACHINE (0x00000002)	Any other legal value.	Security descriptor associated with the machine object in the directory service.
MQDS_SITE (0x00000003)	Any other legal value.	Security descriptor associated with the site object in the directory service.
MQDS_CN (0x00000005)	Any other legal value.	Security descriptor associated with the connected network object in the directory service.
MQDS_ENTERPRISE (0x00000006)	Any other legal value.	Security descriptor associated with the enterprise object in the directory service.

nLength: MUST be set by the client to the length in bytes of the *pSecurityDescriptor* buffer.

lpnLengthNeeded: A **DWORD** that the server MUST set to the length in bytes of the requested security descriptor or public key. If the requested security descriptor or public key is larger than *nLength*, the server MUST set this parameter to the size needed for the requested security descriptor or public key, and return MQ_ERROR_SECURITY_DESCRIPTOR_TOO_SMALL(0xC00E0023).

phServerAuth: A **PCONTEXT_HANDLE_SERVER_AUTH_TYPE** RPC context handle acquired from the *pphServerAuth* parameter in a previous call to **S_DSValidateServer**. The server

MUST use this parameter as a key to locate the GSS security context used to compute the signature returned in *pbServerSignature*. See section 3.1.4.2.

pbServerSignature: MUST be set by the server to a buffer that contains a signed hash over the returned property values. The server MUST construct this signature by creating a hash by using the MD5 algorithm, as specified in [RFC1321], and sealing it, as defined by the following pseudocode:

```
Initialize an MD5 hash context

Add to the hash context the DWORD data value 0x00000001.

Add to the hash context the byte array pSecurityDescriptor. The data
       length is defined by nLength.

Call GSS_Wrap using the output context handle from the GSS
                    security context and the computed MD5 hash

Set pbServerSignature to the wrapped MD5 hash
Set *pdwServerSignatureSize to the size of the wrapped MD5 hash
```

pdwServerSignatureSize: MUST be set by the client to point to a **DWORD** that contains the maximum length in bytes of the server signature to return. MUST be set by the server to the actual length in bytes of the server signature on output. If the server signature is larger than the supplied buffer, the server MUST return MQ_ERROR_USER_BUFFER_TOO_SMALL (0xC00E0028).

Return Values: If the method succeeds, the return value is 0. If the method fails, the return value is an implementation-specific error code.

MQ_OK (0x00000000)
MQ_ERROR_USER_BUFFER_TOO_SMALL (0xC00E0028)
MQ_ERROR_SECURITY_DESCRIPTOR_TOO_SMALL (0xC00E0023)

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

3.1.4.12 S_DSGetObjectSecurityGuid (Opnum 13)

The **S_DSGetObjectSecurityGuid** method retrieves the security descriptor for a directory object specified by an object identifier.

```
HRESULT S_DSGetObjectSecurityGuid(
  [in] handle_t hBind,
  [in] unsigned long dwObjectType,
  [in] const GUID* pGuid,
  [in] unsigned long SecurityInformation,
  [out, size_is(nLength)] unsigned char* pSecurityDescriptor,
  [in] unsigned long nLength,
  [out] unsigned long* lpnLengthNeeded,
  [in] PCONTEXT_HANDLE_SERVER_AUTH_TYPE phServerAuth,
  [out, size_is(*pdwServerSignatureSize)]
    unsigned char* pbServerSignature,
```

```
  [in, out] unsigned long* pdwServerSignatureSize
);
```

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

dwObjectType: Specifies the type of object for which security properties are to be retrieved. MUST be set to one of the object types specified in section 2.2.9.

pGuid: MUST be set by the client to a pointer to the **GUID** of the object for which to retrieve security information.

SecurityInformation: MUST be set by the client to a bitwise mask specifying the information to return in the *pSecurityDescriptor* parameter. See the *SecurityInformation* parameter description in section 3.1.4.11.

pSecurityDescriptor: MUST contain a pointer to a security descriptor, as specified in [MS-DTYP] section **2.4.6**, or to a BLOBHEADER structure followed by an RSA_PUBLIC_KEY structure.<23> See the *pSecurityDescriptor* parameter description in section 3.1.4.11.

nLength: MUST be set by the client to the length in bytes of the *pSecurityDescriptor* buffer.

lpnLengthNeeded: A **DWORD** that the server MUST set to the length in bytes of the requested security descriptor or public key. If the requested security descriptor or public key is larger than *nLength*, the server MUST set this parameter to the size in bytes needed for the requested security descriptor or public key, and return MQ_ERROR_SECURITY_DESCRIPTOR_TOO_SMALL (0xC00E0023).

phServerAuth: A **PCONTEXT_HANDLE_SERVER_AUTH_TYPE** RPC context handle acquired from the *pphServerAuth* parameter in a previous call to **S_DSValidateServer**. The server MUST use this parameter as a key to locate the GSS security context used to compute the signature returned in pbServerSignature. See section 3.1.4.2.

pbServerSignature: MUST point to the signed hash of the security descriptor. See the *pbServerSignature* parameter description in section 3.1.4.11.

pdwServerSignatureSize: MUST be set by the client to point to a **DWORD** that contains the maximum length in bytes of the server signature to return. MUST be set by the server to the actual length in bytes of the server signature on output. If the server signature is larger than the supplied buffer, the server MUST return MQ_ERROR_USER_BUFFER_TOO_SMALL (0xC00E0028).

Return Values: If the method succeeds, the return value is 0. If the method fails, the return value is an implementation-specific error code.

MQ_OK (0x00000000)
MQ_ERROR_SECURITY_DESCRIPTOR_TOO_SMALL (0xC00E0023)
MQ_ERROR_USER_BUFFER_TOO_SMALL (0xC00E0028)

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

3.1.4.13 S_DSSetObjectSecurity (Opnum 5)

The **S_DSSetObjectSecurity** method sets security properties for a directory object specified by a path name.

```
HRESULT S_DSSetObjectSecurity(
  [in] handle_t hBind,
  [in] unsigned long dwObjectType,
  [in, string] const wchar_t* pwcsPathName,
  [in] unsigned long SecurityInformation,
  [in, unique, size_is(nLength)]
    unsigned char* pSecurityDescriptor,
  [in] unsigned long nLength
);
```

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

dwObjectType: Specifies the type of object for which security properties are to be set. MUST be one of the object types specified in section 2.2.9.

pwcsPathName: Pointer to a NULL-terminated Unicode string that MUST contain the pathname to the object in the directory service.

SecurityInformation: MUST be set by the client to a bitwise mask specifying the information to set from the *pSecurityDescriptor* parameter. See the *SecurityInformation* parameter description in section 3.1.4.11. Information in the *pSecurityDescriptor* parameter not associated with bits set in this field MUST be ignored.

pSecurityDescriptor: MUST contain a pointer to a security descriptor, as specified in [MS-DTYP] section **2.4.6**, or to an MQDS_PublicKey structure.<24> See the *pSecurityDescriptor* parameter description in section 3.1.4.11. Note that where 3.1.4.11 indicates that *pSecurityDescriptor* contains a BLOBHEADER followed by an RSA_PUBLIC_KEY structure, this method actually contains an MQDS_PublicKey structure, which is the same structure prefixed by a 4-byte length field).

nLength: MUST be set by the client to the length in bytes of the *pSecurityDescriptor* buffer.

Return Values: If the method succeeds, the return value is 0. If the method fails, the return value is an implementation-specific error code.

MQ_OK (0x00000000)

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

3.1.4.14 S_DSSetObjectSecurityGuid (Opnum 14)

The **S_DSSetObjectSecurityGuid** method sets security properties for a directory object specified by an object identifier.

```
HRESULT S_DSSetObjectSecurityGuid(
  [in] handle_t hBind,
  [in] unsigned long dwObjectType,
  [in] const GUID* pGuid,
```

```
  [in] unsigned long SecurityInformation,
  [in, unique, size_is(nLength)]
    unsigned char* pSecurityDescriptor,
  [in] unsigned long nLength
);
```

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

dwObjectType: Specifies the type of object for which security properties are set. MUST be set to one of the object types specified in section 2.2.9.

pGuid: MUST be set by the client to a pointer to the **GUID** of the object for which properties are to be set.

SecurityInformation: MUST be set by the client to a bitwise mask specifying the information to set from the *pSecurityDescriptor* parameter. See the *SecurityInformation* parameter description in section 3.1.4.11.

pSecurityDescriptor: MUST contain a pointer to security descriptor, as specified in [MS-DTYP] section **2.4.6**, or to an MQDS_PublicKey structure.<25> See the *pSecurityDescriptor* parameter description in section 3.1.4.11. Note that while section 3.1.4.11 indicates that *pSecurityDescriptor* contains a BLOBHEADER followed by an RSA_PUBLIC_KEY structure, this method actually contains an MQDS_PublicKey structure, which is the same structure prefixed by a 4-byte length field.

nLength: MUST be set by the client to the length in bytes of the *pSecurityDescriptor* buffer.

Return Values: If the method succeeds, the return value is 0. If the method fails, the return value is an implementation-specific error code.

MQ_OK (0x00000000)

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

3.1.4.15 S_DSQMGetObjectSecurity (Opnum 21)

The **S_DSQMGetObjectSecurity** retrieves the security descriptor for a machine object specified by an object identifier. This method is intended for use by a queue manager to manipulate its own directory service object of type MQDS_MACHINE. Client applications SHOULD NOT use this method.<26>

```
HRESULT S_DSQMGetObjectSecurity(
  [in] handle_t hBind,
  [in] unsigned long dwObjectType,
  [in] const GUID* pGuid,
  [in] unsigned long SecurityInformation,
  [out, size_is(nLength)] unsigned char* pSecurityDescriptor,
  [in] unsigned long nLength,
  [out] unsigned long* lpnLengthNeeded,
  [in] unsigned long dwContext,
  [in] PCONTEXT_HANDLE_SERVER_AUTH_TYPE phServerAuth,
  [out, size_is(*pdwServerSignatureSize)]
    unsigned char* pbServerSignature,
```

```
  [in, out] unsigned long* pdwServerSignatureSize
);
```

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

dwObjectType: Specifies the type of object for which the security descriptor is retrieved. MUST be set to one of the object types specified in section 2.2.9.

pGuid: MUST be set by the client to a pointer to the **GUID** of the object to retrieve security elements.

SecurityInformation: MUST be set by the client to a bitwise mask specifying the information to return in the *pSecurityDescriptor* parameter. The bit fields are defined by the following table:

Value	Meaning
OWNER_SECURITY_INFORMATION 0x00000001	OWNER field from the security descriptor.
GROUP_SECURITY_INFORMATION 0x00000002	GROUP field from the security descriptor.
DACL_SECURITY_INFORMATION 0x00000004	Discretionary ACL field from the security descriptor.
SACL_SECURITY_INFORMATION 0x00000008	System ACL field from the security descriptor.

The *SecurityInformation* parameter MUST specify a bit wise OR of one or more of:

- OWNER_SECURITY_INFORMATION
- GROUP_SECURITY_INFORMATION
- DACL_SECURITY_INFORMATION
- SACL_SECURITY_INFORMATION

If the *SecurityInformation* parameter includes any other value, the server MUST NOT complete the call and MUST return an error.

pSecurityDescriptor: MUST contain a pointer to a security descriptor, as specified in [MS-DTYP] section 2.4.6. See the *pSecurityDescriptor* parameter description in section 3.1.4.11.

nLength: MUST be set by the client to the length in bytes of the *pSecurityDescriptor* buffer.

lpnLengthNeeded: A **DWORD** that the server MUST set to the length of the requested security descriptor. If the requested security descriptor is larger than *nLength*, the server MUST set this parameter to the size in bytes needed for the requested security descriptor, and return MQ_ERROR_SECURITY_DESCRIPTOR_TOO_SMALL (0xC00E0023).

dwContext: MUST be set by the client to a value that the client can use to correlate callbacks with the initial call to **S_DSQMGetObjectSecurity**. The server MUST supply this value in the *dwContext* parameter in the related calls of the **S_DSQMGetObjectSecurityChallengeResponceProc** callback method.

phServerAuth: A **PCONTEXT_HANDLE_SERVER_AUTH_TYPE** RPC context handle acquired from the *pphServerAuth* parameter in a previous call to **S_DSValidateServer**. The server MUST use this parameter as a key to locate the GSS security context used to compute the signature returned in *pbServerSignature*. See section 3.1.4.2.

pbServerSignature: Pointer to a variable that contains the signed **MD5 hash**, as specified in [RFC1321], of the data pointed to by the **pSecurityDescriptor** field. The algorithm for creating this is specified by the following pseudocode:

```
Initialize an MD5 hash context

Add to the hash context the DWORD data value 0x00000001

Add to the hash context the byte array pSecurityDescriptor.
The data length is defined by nLength

Call GSS_Wrap using the output context handle from GSS
      security context and the computed MD5 hash

Set pbServerSignature to the wrapped MD5 hash

Set *pdwServerSignatureSize to the size in bytes of the wrapped MD5
      hash
```

pdwServerSignatureSize: MUST be set by the client to point to a **DWORD** that contains the maximum length in bytes of the server signature to return. MUST be set by the server to the actual length in bytes of the server signature on output. If the server signature is larger than the supplied buffer, the server MUST return MQ_ERROR_USER_BUFFER_TOO_SMALL (0xC00E0028).

Return Values: If the method succeeds, the return value is 0. If the method fails, the return value is an implementation-specific error code.

MQ_OK (0x00000000)
MQ_ERROR_SECURITY_DESCRIPTOR_TOO_SMALL (0xC00E0023)
MQ_ERROR_USER_BUFFER_TOO_SMALL (0xC00E0028)

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

The server MAY invoke the **S_DSQMGetObjectSecurityChallengeResponceProc** callback method to authenticate the client before performing this method.<27> If the server invokes the callback method, the server MUST use the value of the *dwContext* parameter as the *dwContext* parameter in the corresponding invocation of **S_DSQMGetObjectSecurityChallengeResponceProc**. If the callback method is invoked and returns an error, the server MUST NOT perform the method and MUST return an implementation-specific error code.

3.1.4.16 S_DSQMSetMachineProperties (Opnum 19)

The **S_DSQMSetMachineProperties** method sets properties associated with a machine object specified by a path name. This method is intended for use by a queue manager to manipulate its own directory service object of type MQDS_MACHINE. Client applications SHOULD NOT use this method.<28>

```
HRESULT S_DSQMSetMachineProperties(
  [in] handle_t hBind,
  [in, string] const wchar_t* pwcsPathName,
  [in] unsigned long cp,
  [in, size_is(cp)] unsigned long aProp[],
  [in, size_is(cp)] PROPVARIANT apVar[],
  [in] unsigned long dwContext
);
```

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

pwcsPathName: Pointer to a NULL-terminated **UNICODE_STRING** that MUST contain the pathname to a machine object in the directory service.

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

aProp: An array of identifiers of properties to associate with the machine object. Each element MUST specify a value from the property identifiers table, as specified in section 2.2.10.1. Each element MUST specify the property identifier for the corresponding property value at the same element index in *apVar*. The array MUST contain at least one element.

apVar: An array that specifies the property values to associate with the machine object. Each element MUST specify the property value for the corresponding property identifier at the same element index in *aProp*. The array MUST contain at least one element.

dwContext: MUST be set by the client to a value that the client can use to correlate callbacks with the initial call to **S_DSQMSetMachineProperties**. The server MUST supply this value in the *dwContext* parameter in the related calls of the **S_DSQMSetMachinePropertiesSignProc** callback method.

Return Values: If the method succeeds, the return value is 0. If the method fails, the return value is an implementation-specific error code.

MQ_OK (0x00000000)
MQ_ERROR_ILLEGAL_PROPID (0xC00E0039)

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

The server MAY use the **S_DSQMSetMachinePropertiesSignProc** callback method to authenticate the caller for this procedure.<29> If the server performs the security callback, the server MUST use the value of the *dwContext* parameter as the *dwContext* parameter in the corresponding invocation of **S_DSQMSetMachinePropertiesSignProc**. If the callback method is invoked and returns an error, the server MUST NOT perform the method and MUST return an implementation-specific error code.

For each property identifier in *aProp*, the server MUST verify that the property identifier is valid for an object of type MQDS_MACHINE, and that the corresponding variant in *apVar* is of the type defined for the property identifier, as specified in the tables in section 2.2.10.1.<30>

3.1.4.17 S_DSLookupBegin (Opnum 6)

The **S_DSLookupBegin** method performs a query over the directory objects and returns an RPC context handle that can be used to retrieve the result set through a subsequent series of calls to **S_DSLookupNext**. When the client has no further use of the RPC context handle returned from this method, the client can close the context handle through a call to **S_DSLookupEnd**.

```
HRESULT S_DSLookupBegin(
  [in] handle_t hBind,
  [out] PPCONTEXT_HANDLE_TYPE pHandle,
  [in, unique, string] wchar_t* pwcsContext,
  [in, unique] MQRESTRICTION* pRestriction,
  [in, ref] MQCOLUMNSET* pColumns,
  [in, unique] MQSORTSET* pSort,
  [in] PCONTEXT_HANDLE_SERVER_AUTH_TYPE phServerAuth
);
```

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

pHandle: MUST be set by the server to point to an RPC context handle to be used in subsequent calls to **S_DSLookupNext** and **S_DSLookupEnd**.

pwcsContext: Unicode string that specifies a starting point of the query within the directory service. The client SHOULD set this parameter to NULL, and the server MUST ignore it.

pRestriction: Pointer to an **MQRESTRICTION** structure specifying a set of constraints over the objects to be returned. The server MUST restrict the query results to include only objects that have properties that satisfy all of the restrictions specified in this parameter. See section 2.2.12.

pColumns: Pointer to an **MQCOLUMNSET** structure that specifies the object properties to be returned. The server MUST return (in the result set) only the properties specified by this parameter in the order specified by this parameter. See section 2.2.13.

pSort: Pointer to an **MQSORTSET** structure that defines the sort order of the result set. The server MUST sort the objects in the result set according to this multikey sort order. See section 2.2.15.

phServerAuth: A **PCONTEXT_HANDLE_SERVER_AUTH_TYPE** RPC context handle acquired from the *pphServerAuth* parameter in a previous call to **S_DSValidateServer**.

Return Values: If the method succeeds, the return value is 0. If the method fails, the return value is an implementation-specific error code.

MQ_OK (0x00000000)

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

If the client supplies a non-NULL *pRestriction* parameter, the server MUST use the *pRestriction* parameter to limit the objects returned from the query such that only objects that satisfy all of the restrictions in this parameter are in the result set. The server MUST use the *pColumns* parameter to limit the properties of the objects returned from the query. The server MUST use the *pSort* parameter to sort the objects returned from the query.

The server MUST add an entry to the in-progress query table keyed by *pHandle*. The entry must contain the result set from the query and a cursor index value initialized to zero.

3.1.4.18 S_DSLookupNext (Opnum 7)

The **S_DSLookupNext** method returns a portion of the data from the result set computed in a previous call to **S_DSLookupBegin**, and updates the cursor index to the first directory object that has not yet been returned to the client.

```
HRESULT S_DSLookupNext(
  [in] handle_t hBind,
  [in] PCONTEXT_HANDLE_TYPE Handle,
  [in] unsigned long* dwSize,
  [out] unsigned long* dwOutSize,
  [out, size_is(*dwSize), length_is(*dwOutSize)]
    PROPVARIANT pbBuffer[],
  [in] PCONTEXT_HANDLE_SERVER_AUTH_TYPE phServerAuth,
  [out, size_is(*pdwServerSignatureSize)]
    unsigned char* pbServerSignature,
  [in, out] unsigned long* pdwServerSignatureSize
);
```

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

Handle: MUST contain an RPC context handle acquired from a previous call to **S_DSLookupBegin**. The handle MUST NOT have been used in a previous call to **S_DSLookupEnd**.

dwSize: MUST point to an unsigned long that contains the maximum number of elements to be returned in the *pbBuffer* array. If this parameter is less than the number of elements in the *pColumns* parameter in the corresponding call to **S_DSLookupBegin**, the server MUST set the *dwOutSize* parameter to 0, and return without retrieving any object properties.

dwOutSize: Pointer to an unsigned long that the server MUST set to the number of properties returned in *pbBuffer* for the set of objects being returned from this invocation of the **S_DSLookupNext** method. The server MUST return as many completed sets of properties as will fit in the buffer. If no matching objects are found, the server MUST set this parameter to 0 to inform the client that there is no more data.

pbBuffer: MUST point to an array of PROPVARIANT structures to contain the returned properties.

phServerAuth: A **PCONTEXT_HANDLE_SERVER_AUTH_TYPE** RPC context handle acquired from the *pphServerAuth* parameter in a previous call to **S_DSValidateServer**. The server MUST use this parameter as a key to locate the GSS security context used to compute the signature returned in *pbServerSignature*. See section 3.1.4.2.

pbServerSignature: MUST point to the signed hash over the property values returned in *pbBuffer*. See the *pbServerSignature* parameter description in section 3.1.4.7.

pdwServerSignatureSize: MUST be set by the client to point to an unsigned long that contains the maximum length in bytes of the server signature to return. MUST be set by the server to the actual length in bytes of the server signature on output. If the server signature is larger than the supplied buffer, the server MUST return MQ_ERROR_USER_BUFFER_TOO_SMALL (0xC00E0028).

Return Values: If the method succeeds, the return value is 0. If the method fails, the return value is an implementation-specific error code.

MQ_OK (0x00000000)
MQ_ERROR_USER_BUFFER_TOO_SMALL (0xC00E0028)

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

With each call to **S_DSLookupNext**, the server MUST use the context handle in pHandle to locate the query entry in the query table. If there are no more results to return, the server MUST set *dwOutSize* to 0; otherwise, the server MUST populate the *pbBuffer* parameter with the properties of the result set at the current cursor index, and MUST increment the current cursor index. The server MUST set the *dwOutSize* parameter to the number of properties being returned.

3.1.4.19 S_DSLookupEnd (Opnum 8)

The **S_DSLookupEnd** method closes an opened RPC context handle created from a previous call to **S_DSLookupBegin**.

```
HRESULT S_DSLookupEnd(
  [in] handle_t hBind,
  [in, out] PPCONTEXT_HANDLE_TYPE phContext
);
```

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

phContext: MUST point to an RPC context handle returned by a previous call to **S_DSLookupBegin**. MUST NOT have been used in a previous call to **S_DSLookupEnd**. The server MUST set this parameter to NULL.

NULL

Return Values: If the method succeeds, the return value is 0. If the method fails, the return value is an implementation-specific error code.

MQ_OK (0x00000000)

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

The server MUST use the context handle in pHandle as a key to locate the query entry in the query table. The server MUST delete the result set associated with the query entry and remove the query entry from the table. The server MUST set the context handle to NULL before returning.

3.1.4.20 S_DSCreateServersCache (Opnum 20)

The **S_DSCreateServersCache** method returns a list of Backup Site Controllers (BSCs) associated with a specified site. The client calls this method to enumerate the BSCs associated with sites in the configured list of sites in the enterprise.

```
HRESULT S_DSCreateServersCache(
```

```
  [in] handle_t hBind,
  [in, out] unsigned long* pdwIndex,
  [in, out, ptr, string] wchar_t** lplpSiteServers,
  [in] PCONTEXT_HANDLE_SERVER_AUTH_TYPE phServerAuth,
  [out, size_is(*pdwServerSignatureSize)]
    unsigned char* pbServerSignature,
  [in, out] unsigned long* pdwServerSignatureSize
);
```

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

pdwIndex: Pointer to an unsigned long that contains an index into the configured list of sites in the enterprise indicating the site to which the list of BSCs is to be returned.

lplpSiteServers: Pointer to a pointer to a string that contains the list of servers associated with the indexed site. The string MUST be in Server List String (section 2.2.17) format.

phServerAuth: **PCONTEXT_HANDLE_SERVER_AUTH_TYPE** RPC context handle acquired from the *pphServerAuth* parameter in a previous call to **S_DSValidateServer**. The server MUST use this parameter as a key to locate the GSS security context used to compute the signature returned in *pbServerSignature*. See section 3.1.4.2.

pbServerSignature: MUST be set by the server to a buffer that contains a signed hash over the server list returned in *lplpSiteServers*. The server MUST construct this signature by creating a hash by using the MD5 algorithm (as specified in [RFC1321]) and sealing it, as specified by the following pseudocode:

```
Initialize an MD5 hash context
Add to the hash context the DWORD value 0x00000002

Construct a PROPVARIANT apVar as follows
SET apVar.vt to VT_UI4
SET apVar.ulVal to the contents of pdwIndex

Add to the hash context the property value apVar where
       the length is defined as appropriate for the variant type
       apVar.vt

Construct a PROPVARIANT apVar as follows
SET apVar.vt to VT_LPWSTR
SET apVar.pwszVal to the contents of lplpSiteServers

Add to the hash context the property value apVar where
       the length is defined as appropriate for the variant type
       apVar.vt

Call GSS_Wrap using the output_context_handle from GSS
       security context and the computed MD5 hash

SET pbServerSignature to the wrapped MD5 hash
SET *pdwServerSignatureSize to the size of the wrapped MD5 hash
```

pdwServerSignatureSize: MUST be set by the client to point to an unsigned long that contains the maximum length in bytes of the server signature to return. MUST be set by the server to the actual length in bytes of the server signature on output. If the server signature is larger than the supplied buffer, the server MUST return MQ_ERROR_USER_BUFFER_TOO_SMALL (0xC00E0028).

Return Values: If the method succeeds, the return value is 0. If the method fails, the return value is an implementation-specific error code. The server MUST return MQDS_E_NO_MORE_DATA (0xC00E0000), if *pdwIndex* is an invalid index, into the configured list of sites in the enterprise.

MQ_OK (0x00000000)
MQDS_E_NO_MORE_DATA (0xC00E0000)
MQ_ERROR_USER_BUFFER_TOO_SMALL (0xC00E0028)

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

The **S_DSCreateServersCache** method returns information on Backup Site Controllers (BSCs) associated with sites in the enterprise. The client uses this method to iterate through the list of sites in the enterprise by calling the method repeatedly. Before the first invocation, the client sets the *pdwIndex* parameter to 0. After each successful invocation, the client increments the *pdwIndex* parameter by 1 and calls the method again. The client repeats this sequence until the method call returns an error.

3.1.5 Timer Events

There are no timer events.

3.1.6 Other Local Events

3.1.6.1 PCONTEXT_HANDLE_SERVER_AUTH_TYPE Rundown

This event occurs when a **PCONTEXT_HANDLE_SERVER_AUTH_TYPE** RPC context handle has been established between a client and server through a call to **S_DSValidateServer**, and the RPC connection between the client and server is severed before the context handle has been closed via a call to **S_DSCloseServerHandle**.

The server MUST use the context handle (supplied as an event argument) as a key to locate the security context entry in the security context table. The server MUST delete the GSS security context associated with the security context entry through a call to GSS_Delete_sec_context, as specified in [RFC2743] section 2.2.3. The server MUST remove the security context entry from the security context table.

3.1.6.2 PCONTEXT_HANDLE_TYPE Rundown

This event occurs when a **PCONTEXT_HANDLE_TYPE** RPC context handle has been established between a client and server through a call to **S_DSLookupBegin**, and the RPC connection between the client and server is severed before the context handle has been closed via a call to **S_DSLookupEnd**.

The server MUST use the context handle (supplied as an event argument) as a key to locate the query entry in the query table. The server MUST delete the result set associated with the query entry and remove the query entry from the table.

3.2 dscomm Client Details

3.2.1 Abstract Data Model

The client needs to maintain the GSS security context acquired through the **S_InitSecCtx** callback of the **S_DSValidateServer** sequence. This security context is used during the mutual authentication handshake and subsequently in the validation of the *pServerSignature* parameter for each of the following protocol methods:

- **S_DSGetProps**
- **S_DSGetPropsGuid**
- **S_DSLookupNext**
- **S_DSGetObjectSecurity**
- **S_DSGetObjectSecurityGuid**
- **S_DSQMGetObjectSecurity**
- **S_DSCreateServersCache**

The client SHOULD maintain its site identifier.<31>

The client SHOULD maintain a list of known directory service servers, one of which is designated as the current server.<32><33> The client will connect to the current server to perform all [MS-MQDS] operations. If the operation fails, the client MUST either connect to another server from the list, or fail the operation.

3.2.2 Timers

No protocol timers are required beyond those that are used internally by RPC to implement resiliency to network outages. See [MS-RPCE].

3.2.3 Initialization

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

The client MAY<34> call the **S_DSGetServerPort** method. This method returns an RPC endpoint port number for either the RPC over TCP/IP protocol sequence or the RPC over SPX protocol sequence, as specified in [MS-RPCE], as specified by the client. The client MUST create an RPC binding handle to the remote computer by using the returned RPC port. All other method calls on the server interface MUST use the resulting binding handle.

Many of the methods of the dscomm interface require a **PCONTEXT_HANDLE_SERVER_AUTH_TYPE** RPC context handle as an input parameter. The client MUST call **S_DSValidateServer** to acquire a context handle prior to calling any method that requires this handle as input. The client MUST call **S_DSCloseServerHandle** when finished with the context handle.

3.2.4 Message Processing Events and Sequencing Rules

When a method completes, the values returned by RPC MUST be returned unmodified to the upper layer.

The client MUST ignore errors returned from the RPC server and MUST notify the application invoker of the error received in the higher layer. Otherwise, no special message processing is required on the client beyond the processing required in the underlying RPC protocol.

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

The methods specified below MUST NOT throw exceptions.

The following methods comprise the dscomm client interface.

Methods in RPC Opnum Order

Method	Description
S_DSQMSetMachinePropertiesSignProc	A callback method called by the server in response to a client call to **S_DSQMSetMachineProperties**. Through this method, the server provides a challenge that the client must sign to authenticate itself. Opnum: 0
S_DSQMGetObjectSecurityChallengeResponceProc	A callback method called by the server in response to a client call to **S_DSQMGetObjectSecurity**. Through this method, the server provides a challenge that the client must sign to authenticate itself. Opnum: 1
S_InitSecCtx	A callback method called by the server in response to a client call to **S_DSValidateServer**. This method is called for each leg of a mutual authentication security context negotiation. Opnum: 2

3.2.4.1 S_DSQMSetMachinePropertiesSignProc (Opnum 0)

S_DSQMSetMachinePropertiesSignProc is a callback method called by the server during a client call to **S_DSQMSetMachineProperties**. Through this method, the server provides a challenge that the client must sign to authenticate itself.

```
[callback] HRESULT S_DSQMSetMachinePropertiesSignProc(
  [in, size_is(dwCallengeSize)] byte* abChallenge,
  [in] unsigned long dwCallengeSize,
  [in] unsigned long dwContext,
  [in, out, size_is(dwSignatureMaxSize),
             length_is(*pdwSignatureSize)]
    byte* abSignature,
  [in, out] unsigned long* pdwSignatureSize,
  [in] unsigned long dwSignatureMaxSize
);
```

abChallenge: MUST be set by the caller to a pointer to a buffer that contains the challenge to be signed. The challenge SHOULD be cryptographically random.

dwCallengeSize: MUST be set by the caller to the size in bytes of the challenge in the *abChallenge* parameter.

dwContext: MUST be set by the caller to the value supplied by the client in the *dwContext* parameter of the corresponding call to the **S_DSQMSetMachineProperties** method. This parameter provides a way for the receiver to correlate the callback with the receiver's in-progress call to **S_DSQMSetMachineProperties**.

abSignature: MUST be set by the caller to a pointer to a buffer to contain the returned signature. MUST be set by the receiver to a signature over the challenge in abChallenge. The algorithm for creating this signature is specified by the following pseudocode:

```
Initialize an MD5 hash context

Add to the hash context the byte array abChallenge. The data length
       is defined by dwCallengeSize.

Add to the hash context a DWORD data value equal to the number of
       properties being set in the corresponding call to
       S DSQMSetMachineProperties.

FOR each property in the corresponding call to
    S_DSQMSetMachineProperties

    Add to the hash context the DWORD value of the property
           identifier (from aProp[])

    Add to the hash context the type-specific data value of the
           property (from apVar[]). The data value and length are
           defined by the variant type of the property
           (apVar[].vt)
END FOR


Sign the MD5 hash using the private key corresponding to the public key
stored in the certificate in the property PROPID QM SIGN PK associated
with the machine object in the directory service.
The machine object is the object specified by the pwcsPathName parameter
in the corresponding call to S_DSQMSetMachineProperties.
Set abSignature to the signed MD5 hash

Set *pdwSignatureSize to the size in bytes of the signed MD5 hash.
```

pdwSignatureSize: Size in bytes of the signature in the *abSignature* parameter. MUST be set by the receiver to the actual length in bytes of the signature returned in *abSignature* on output.

dwSignatureMaxSize: MUST be set by the caller to the maximum length in bytes of the server signature to be returned in *abSignature*. If the signature is larger than the supplied buffer, the server MUST return MQ_ERROR_USER_BUFFER_TOO_SMALL (0xC00E0028).

Return Values: This method is obsolete. The server SHOULD NOT call this method, and the client SHOULD return MQ_ERROR_NOT_SUPPORTED (0xC00E03EB).<35> If the method succeeds, the return value is 0. If the method fails, the return value is an implementation-specific error code.

ERROR_SUCCESS (0x00000000)
MQ_ERROR_NOT_SUPPORTED (0xC00E03EB)

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

3.2.4.2 S_DSQMGetObjectSecurityChallengeResponceProc (Opnum 1)

S_DSQMGetObjectSecurityChallengeResponceProc is a callback method called by the server during a client call to **S_DSQMGetObjectSecurity**. Through this method, the server provides a challenge that the client must sign to authenticate itself.

```
[callback] HRESULT S_DSQMGetObjectSecurityChallengeResponceProc(
  [in, size_is(dwCallengeSize)] byte* abChallenge,
  [in] unsigned long dwCallengeSize,
  [in] unsigned long dwContext,
  [in, out, size_is(dwCallengeResponceMaxSize),
             length_is(*pdwCallengeResponceSize)]
    byte* abCallengeResponce,
  [in, out] unsigned long* pdwCallengeResponceSize,
  [in] unsigned long dwCallengeResponceMaxSize
);
```

abChallenge: MUST be set by the caller to a pointer to a buffer that contains the challenge to be signed. The challenge SHOULD be cryptographically random.

dwCallengeSize: MUST be set by the caller to the size, in bytes, of the challenge in the *abChallenge* parameter.

dwContext: MUST be set by the caller to the value that was supplied in the *dwContext* parameter of the corresponding call to the **S_DSQMGetObjectSecurity** method. This parameter provides a way for the receiver to correlate the callback with the receiver's in-progress call to **S_DSQMGetObjectSecurity**.

abCallengeResponce: MUST be set by the caller to a pointer to a buffer to contain the returned signature. MUST be set by the receiver to a signature over the challenge in *abChallenge*. The algorithm for creating this signature is specified by the following pseudocode:

```
Initialize an MD5 hash context

Add to the hash context the byte array abChallenge. The data length
       is defined by dwCallengeSize.


Sign the MD5 hash using the private key corresponding to the
public key stored in the certificate in the property
PROPID_QM_SIGN_PK associated with the machine object in the
directory service. The machine object is the object specified by
the pGuid parameter in the corresponding call to
S_DSQMGetObjectSecurity.

Set abCallengeResponce to the signed MD5 hash
Set *pdwCallengeResponceSize to the size in bytes of the signed MD5
    hash.
```

pdwCallengeResponceSize: Size in bytes of the signature in the *abCallengeResponce* parameter. MUST be set by the receiver to the actual length, in bytes, of the signature returned in *abCallengeResponce* on output.

dwCallengeResponceMaxSize: MUST be set by the caller to the maximum length in bytes of the server signature to be returned in *abCallengeResponce*. If the server signature is larger than the supplied buffer, the server MUST return MQ_ERROR_USER_BUFFER_TOO_SMALL (0xC00E0028).

Return Values: This method is obsolete. The server SHOULD NOT call this method, and the client SHOULD return MQ_ERROR_NOT_SUPPORTED (0xC00E03EB).<36> If the method succeeds, the return value is 0. If the method fails, the return value is an implementation-specific error code.

ERROR_SUCCESS (0x00000000)
MQ_ERROR_NOT_SUPPORTED (0xC00E03EB)

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

3.2.4.3 S_InitSecCtx (Opnum 2)

S_InitSecCtx is a callback method called by the server during a client call to **S_DSValidateServer**. These two methods are used to tunnel a GSS (as specified in [RFC2743]) security negotiation to provide mutual authentication between the client and server.

```
[callback] HRESULT S_InitSecCtx(
  [in] unsigned long dwContext,
  [in, size_is(dwServerBuffSize)]
    unsigned char* pServerbuff,
  [in] unsigned long dwServerBuffSize,
  [in] unsigned long dwClientBuffMaxSize,
  [out, size_is(dwClientBuffMaxSize),
          length_is(*pdwClientBuffSize)]
    unsigned char* pClientBuff,
  [out] unsigned long* pdwClientBuffSize
);
```

dwContext: MUST be set by the caller to the correlation value supplied by the client in the *dwContext* parameter in the corresponding call to **S_DSValidateServer**. This parameter provides a way for the receiver to correlate the callback with the receiver's in-progress call to **S_DSValidateServer**.

pServerbuff: MUST be set by the caller to point to a buffer that contains the output_token from the GSS_Accept_sec_context, as specified in [RFC2743].

dwServerBuffSize: MUST be set by the caller to the length, in bytes, of the output_token within *pServerBuff*.

dwClientBuffMaxSize: MUST be set by the caller to the size, in bytes, of the buffer to be returned in *pClientBuff*.

pClientBuff: MUST be set by the caller to point to a buffer to hold the returned token. MUST be set by the receiver to the output_token from a call to GSS_Init_sec_context. The buffer length

MUST NOT exceed the value specified by *dwClientBuffMaxSize*. If the negotiated token is larger than the supplied buffer, the server MUST return MQ_ERROR_USER_BUFFER_TOO_SMALL (0xC00E0028).

pdwClientBuffSize: MUST be set by the receiver to the actual size, in bytes, of the token in *pClientBuff*.

Return Values: If the method succeeds, and the negotiation is complete, the return value is 0. If the method succeeds, and the negotiation is not complete, the return value is SEC_I_CONTINUE_NEEDED (0x00090312). If the method fails, the return value is an implementation-specific error code.

ERROR_SUCCESS (0x00000000)
SEC_I_CONTINUE_NEEDED (0x00090312)

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

The caller MUST supply an input_token in *pServerBuff* computed through a call to GSS_Accept_sec_context. The receiver MUST process this input_token through a call to GSS_Init_sec_context, generating an output_token that MUST be returned in *pServerBuff*.

If GSS_Init_sec_context returns GSS_S_CONTINUE_NEEDED, this is a signal that the negotiation is not complete. The receiver MUST return SEC_I_CONTINUE_NEEDED (0x00090312).

If GSS_Init_sec_context returns GSS_S_COMPLETE, the negotiation is complete. The receiver MUST save the output context handle in the GSS security context state associated with the *dwContext* parameter. The receiver MUST return SEC_E_OK (0x00000000).

3.2.5 Timer Events

There are no timer events.

3.2.6 Other Local Events

3.2.6.1 Initialize List of Known Directory Service Servers Event

This event is performed when the client needs to refresh its list of known directory service servers.<37> After completion of this event, the client will have a list of known directory service servers for use in future [MS-MQDS] operations.

1. To begin processing this event, the client MUST create an RPC binding to a directory service server from its current list of known directory service servers.

2. Once an RPC binding is established, the client creates a temporary variable temp_server_list, which is an empty list of server names and addresses.

3. The client issues an **S_DSLookupBegin** method call with query parameters:

   Restriction: rel=PRGT, prop=PROPID_QM_SERVICE, prval=1

   Restriction: rel=PREQ, prop=PROPID_QM_SITE_ID, prval=client's site identifier

   Column: PROPID_QM_PATHNAME_DNS

   Column: PROPID_QM_PATHNAME

Column: PROPID_QM_ADDRESS

4. The client MUST call the **S_DSLookupNext** method. If the method returns success the output parameter will include the name and address of known directory service servers for the client's site. The client must add this to temp_server_list. The client MUST repeat the above process starting from step 4.

5. If the method fails, the client MUST call the **S_DSLookupEnd** method.

6. The client MUST then copy temp_server_list to store the list of known directory service servers for future operations using the [MS-MQDS] protocol.

3.3 dscomm2 Server Details

3.3.1 Abstract Data Model

3.3.1.1 PCONTEXT_HANDLE_DELETE_TYPE RPC Context Handle

This is an RPC context handle that represents an in-progress directory object delete notification.

A protocol client acquires a **PCONTEXT_HANDLE_DELETE_TYPE** RPC context handle through a call to the **S_DSBeginDeleteNotification** method. Each instance of this RPC context handle represents internal state that the server needs to maintain. The server SHOULD register a rundown method to close these context handles in the event that the client fails to call the **S_DSEndDeleteNotification** method.

The server MUST maintain the following state in the delete notification table.

For deleting queues, the internal state that the server MUST retain includes:

- The object type of the object being deleted (either MQDS_QUEUE or MQDS_MACHINE).
- The object pathname.
- The object identifier of the owner site.<38>

Also, if the object being deleted is an MQDS_QUEUE, the server MUST retain the following information:

- The object identifier of the queue manager that owns the queue being deleted.
- A flag indicating if this queue is a foreign queue.

The server MUST retain this information until the client closes the context handle through a call to the **S_DSEndDeleteNotification** method.

3.3.2 Timers

No protocol timers are required beyond those used internally by RPC to implement resiliency to network outages. See [MS-RPCE].

3.3.3 Initialization

Parameters necessary to initialize the RPC protocol are specified in section 2.1.

The server MAY<39> register static RPC ports for the server. If it does, it MUST return those ports in a call to the **S_DSGetServerPort** method.

3.3.4 Message Processing Events and Sequencing Rules

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

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

The following methods comprise the dscomm2 server interface:

Methods in RPC Opnum Order

Method	Description
S_DSGetComputerSites	Returns information on the sites of which a computer is a member. Opnum: 0
S_DSGetPropsEx	Returns extended properties associated with a directory object specified by a pathname. Opnum: 1
S_DSGetPropsGuidEx	Returns extended properties associated with a directory object specified by an object identifier. Opnum: 2
S_DSBeginDeleteNotification	Begins a delete notification. Opnum: 3
S_DSNotifyDelete	Notifies the server that a machine or queue has been deleted by the client. Opnum: 4
S_DSEndDeleteNotification	Ends a delete notification. Opnum: 5
S_DSIsServerGC	Returns a value that indicates whether a server is a global catalog server. Opnum: 6
Opnum7NotUsedOnWire	Reserved for local use. Opnum: 7
S_DSGetGCListInDomain	Returns a list of global catalog servers in the specified domain. Opnum: 8

In the preceding table, the phrase "Reserved for local use" means that the client MUST NOT send the opnum, and that the server behavior is undefined because it does not affect interoperability.<40>

3.3.4.1 S_DSGetComputerSites (Opnum 0)

The **S_DSGetComputerSites** method returns the site identifier for every site of which the specified computer is a member.

```
HRESULT S_DSGetComputerSites(
  [in] handle_t hBind,
  [in, unique] const wchar_t* pwcsPathName,
  [out] DWORD* pdwNumberOfSites,
  [out, size_is(,*pdwNumberOfSites), length_is(,*pdwNumberOfSites)]
    GUID** ppguidSites,
  [in] PCONTEXT_HANDLE_SERVER_AUTH_TYPE phServerAuth,
  [out, size_is(*pdwServerSignatureSize)]
    unsigned char* pbServerSignature,
  [in, out] DWORD* pdwServerSignatureSize
);
```

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

pwcsPathName: MUST be set by the client to a pointer to a NULL-terminated **UNICODE_STRING** that contains the pathname of the object in the Directory Service. The pathname MUST be the pathname of an object of type MQDS_MACHINE.<41>

pdwNumberOfSites: MUST be set by the server to the number of site identifiers returned in the *ppguidSites* parameter.

ppguidSites: MUST be set by the server to an array of pointers to the **GUID** site identifiers of the sites of which the computer is a member. Each **GUID** referenced by the array MUST be the site identifier for a site that the machine object specified by *pwcsPathName* is a member of. These are obtained from the PROPID_QM_SITE_IDS property of the MQDS_MACHINE object. The array MUST contain the site identifier for every site to which the object specified by *pwcsPathName* is a member.

phServerAuth: A **PCONTEXT_HANDLE_SERVER_AUTH_TYPE** RPC context handle acquired from the *pphServerAuth* parameter in a previous call to **S_DSValidateServer**. The server MUST use this parameter as a key to locate the GSS security context used to compute the signature returned in *pbServerSignature*. See section 3.1.4.2.

pbServerSignature: MUST be set by the server to a buffer that contains a signed hash over the array of site **GUIDs** returned in *ppguidSites* calculated by using the MD5 algorithm (as specified in [RFC1321]) and the GSS security context as specified by the following pseudocode:

```
Initialize an MD5 Hash context
Add to the hash context the DWORD value 0x00000001
Add to the hash value the DWORD value of VT_BLOB
SET cbSize to pdwNumberOfSites times the size in bytes of a GUID
Add to the hash value the DWORD value of cbSize
Add to the hash context the contents of ppGuidSites where the length
    for the data is cbSize
Call GSS_Wrap using the output context handle from GSS
     security context and the computed MD5 hash
SET pbServerSignature to the wrapped MD5 hash
SET *pdwServerSignatureSize to the size of the wrapped MD5 hash
```

pdwServerSignatureSize: MUST be set by the client to point to a **DWORD** that contains the maximum length of the server signature in bytes to return, and MUST be set by the server to contain the actual length in bytes of the server signature on output. If the server signature is

larger than the supplied buffer, the server MUST return MQ_ERROR_USER_BUFFER_TOO_SMALL (0xC00E0028).

Return Values: If the method succeeds, the return value is MQ_OK (0x00000000). If the method fails, the return value is an implementation-specific error code.

MQ_OK (0x00000000)

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

3.3.4.2 S_DSGetPropsEx (Opnum 1)

The **S_DSGetPropsEx** method returns the properties associated with the object specified by a pathname. This method differs from **S_DSGetProps** in that it supports a restricted set of properties that pertain only to queue or machine object security.

```
HRESULT S_DSGetPropsEx(
  [in] handle_t hBind,
  [in, range(1, 58)] DWORD dwObjectType,
  [in] const wchar_t* pwcsPathName,
  [in, range(1,128)] DWORD cp,
  [in, size_is(cp)] PROPID aProp[],
  [in, out, size_is(cp)] PROPVARIANT apVar[],
  [in] PCONTEXT_HANDLE_SERVER_AUTH_TYPE phServerAuth,
  [out, size_is(*pdwServerSignatureSize)]
    unsigned char* pbServerSignature,
  [in, out] DWORD* pdwServerSignatureSize
);
```

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

dwObjectType: Specifies the type of object for which properties are to be retrieved. MUST be set to one of the object types, as specified in section 2.2.9.

pwcsPathName: MUST be set by the client to a pointer to a **UNICODE_STRING** that contains the pathname of the object in the Directory Service from which to retrieve the properties.

cp: MUST be set to the size (in elements) of the arrays *aProp* and *apVar*, which for this method MUST be one (0x00000001). The arrays *aProp* and *apVar* MUST have an identical number of elements, and MUST each contain exactly one element.

aProp: An array of identifiers of properties to retrieve from the object. Each element MUST specify a value from the property identifiers table for the object type specified in *dwObjectType*. Each element MUST specify the property identifier for the corresponding property value at the same element index in *apVar*. The array MUST contain exactly one element.

apVar: MUST be set by the client to an array that holds the property values retrieved from the object. Each element MUST be set by the server to the property value for the corresponding property identifier at the same element index in *aProp*. The array MUST contain exactly one element.

phServerAuth: A **PCONTEXT_HANDLE_SERVER_AUTH_TYPE** RPC context handle acquired from the *pphServerAuth* parameter in a previous call to **S_DSValidateServer**. The server

MUST use this parameter as a key to locate the GSS security context used to compute the signature returned in pbServerSignature. See section 3.1.4.2.

pbServerSignature: MUST be set by the server to a buffer that contains a signed hash over the returned property values. See the *pbServerSignature* parameter description in section 3.1.4.7.

pdwServerSignatureSize: MUST be set by the client to point to a **DWORD** that contains the maximum length of the server signature in bytes to return, and MUST be set by the server to contain the actual length in bytes of the server signature on output. If the server signature is larger than the supplied buffer, the server MUST return MQ_ERROR_USER_BUFFER_TOO_SMALL (0xC00E0028).

Return Values: If the method succeeds, the return value is MQ_OK (0x00000000). If the method fails, the return value is an implementation-specific error code.

MQ_OK (0x00000000)

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

The server MUST verify that the property identifiers supplied in the *aProp* parameter are valid properties and MUST return an error if not.

PROPID_Q_OBJ_SECURITY (1102)

PROPID_QM_OBJ_SECURITY (234)

PROPID_QM_ENCRYPT_PKS (238)

PROPID_QM_SIGN_PKS (239)

For each property identifier in *aProp*, the server MUST verify that the property identifier is valid for the object type specified in *dwObjectType*, and that the corresponding variant in *apVar* is of the type defined for the property identifier, as specified in the tables in section 2.2.10.1.

3.3.4.3 S_DSGetPropsGuidEx (Opnum 2)

The **S_DSGetPropsGuidEx** method returns the properties for the object specified by object identifier. This method differs from **S_DSGetPropsGuid** in that it supports a restricted set of properties that pertain only to queue or machine object security.

```
HRESULT S_DSGetPropsGuidEx(
  [in] handle_t hBind,
  [in, range(1, 58)] DWORD dwObjectType,
  [in, unique] const GUID* pGuid,
  [in, range(1,128)] DWORD cp,
  [in, size_is(cp)] PROPID aProp[],
  [in, out, size_is(cp)] PROPVARIANT apVar[],
  [in] PCONTEXT_HANDLE_SERVER_AUTH_TYPE phServerAuth,
  [out, size_is(*pdwServerSignatureSize)]
    unsigned char* pbServerSignature,
  [in, out] DWORD* pdwServerSignatureSize
);
```

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

dwObjectType: Specifies the type of object for which properties are to be retrieved. MUST be set to one of the object types, as specified in section 2.2.9.

pGuid: MUST specify a pointer to the object identifier of the object for which properties are to be retrieved.

cp: MUST be set to the size (in elements) of the arrays *aProp* and *apVar*, which for this method MUST be one (0x00000001). The arrays *aProp* and *apVar* MUST have an identical number of elements, and MUST each contain exactly one element.

aProp: An array of identifiers of properties to retrieve from the object. Each element MUST specify a value from the property identifiers for the object type specified in *dwObjectType*. Each element MUST specify the property identifier for the corresponding property value at the same element index in *apVar*. The array MUST contain exactly one element.

apVar: MUST be set by the client to an array that holds the property values retrieved from the object. Each element MUST be set by the server to the property value for the corresponding property identifier at the same element index in *aProp*. The array MUST contain exactly one element.

phServerAuth: A **PCONTEXT_HANDLE_SERVER_AUTH_TYPE** RPC context handle acquired from the *pphServerAuth* parameter in a previous call to **S_DSValidateServer**. The server MUST use this parameter as a key to locate the GSS security context used to compute the signature returned in *pbServerSignature*. See section 3.1.4.2.

pbServerSignature: MUST be set by the server to a buffer that contains a signed hash over the returned property values. See the *pbServerSignature* parameter description in section 3.1.4.7.

pdwServerSignatureSize: MUST be set by the client to point to a **DWORD** that contains the maximum length of the server signature in bytes to return, and MUST be set by the server to contain the actual length in bytes of the server signature on output. If the server signature is larger than the supplied buffer, the server MUST return MQ_ERROR_USER_BUFFER_TOO_SMALL (0xC00E0028).

Return Values: If the method succeeds, the return value is MQ_OK (0x00000000). If the method fails, the return value is an implementation-specific error code.

MQ_OK (0x00000000)

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

The server MUST verify that the property identifiers supplied in the *aProp* parameter are valid properties and MUST return an error if not.

PROPID_Q_OBJ_SECURITY (1102)

PROPID_QM_OBJ_SECURITY (234)

PROPID_QM_ENCRYPT_PKS (238)

PROPID_QM_SIGN_PKS (239)

For each property identifier in *aProp*, the server MUST verify that the property identifier is valid for the object type specified in *dwObjectType*, and that the corresponding variant in *apVar* is of the type defined for the property identifier, as specified in the tables in section 2.2.10.1.

3.3.4.4 S_DSBeginDeleteNotification (Opnum 3)

The **S_DSBeginDeleteNotification** method begins a delete notification and returns an RPC context handle associated with the delete notification.

```
HRESULT S_DSBeginDeleteNotification(
  [in] handle_t hBind,
  [in] const wchar_t* pwcsPathName,
  [out] PPCONTEXT_HANDLE_DELETE_TYPE pHandle,
  [in] PCONTEXT_HANDLE_SERVER_AUTH_TYPE phServerAuth
);
```

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

pwcsPathName: MUST be set by the client to a pointer to a null-terminated **UNICODE_STRING** that contains the pathname for an object of type MQDS_MACHINE or MQDS_QUEUE.<42>

pHandle: MUST be set by the server to a pointer to a unique RPC context_handle representing the delete notification. This handle is used by the client in subsequent calls to **S_DSNotifyDelete**.

phServerAuth: A **PCONTEXT_HANDLE_SERVER_AUTH_TYPE** RPC context handle acquired from the *pphServerAuth* parameter in a previous call to **S_DSValidateServer**. The server MUST use this parameter as a key to locate the GSS security context used to compute the signature returned in *pbServerSignature*. See section 3.1.4.2.

Return Values: If the method succeeds, the return value is MQ_OK (0x00000000). If the method fails, the return value is an implementation-specific error code.<43>

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

The server MUST add an entry to the delete notification table as follows:

- Set the object type to the type of object identified by pwcsPathName.<44>
- Set the name to *pwcsPathName*.
- Set the owner site to the object identifier of the site owning the queue object.<45>

Also, if the object type is MQDS_QUEUE, the server MUST:

- Set the owner queue manager object identifier to the queue manager object identifier for the queue.
- Set the flag indicating if this queue is a foreign queue.

The server MUST retain this information until the client calls the **S_DSEndDeleteNotification** method or until the PCONTEXT_HANDLE_DELETE_TYPE rundown event occurs.

3.3.4.5 S_DSNotifyDelete (Opnum 4)

The **S_DSNotifyDelete** method instructs the server to notify the computer that owns the deleted object about the deletion.

```
HRESULT S_DSNotifyDelete(
  [in] handle_t hBind,
  [in] PCONTEXT_HANDLE_DELETE_TYPE Handle
);
```

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

Handle: MUST be set by the client to a pointer to an RPC context_handle acquired from a previous call to **S_DSBeginDeleteNotification**. This RPC context handle MUST NOT have been used in a previous call to **S_DSEndDeleteNotification**.

Return Values: If the method succeeds, the return value is MQ_OK (0x00000000). If the method fails, the return value is an implementation-specific error code.

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

The server MUST perform the following processing:

- Look up the handle in the delete notification table. If not found, return an implementation-specific error code.
- If the object type in the delete notification table entry is MQDS_MACHINE, the server MAY notify the Primary Site Controller of the owner site that the machine object was deleted by the client. This specification does not mandate how this notification is performed.<46>
- If the object type in the delete notification table entry is MQDS_QUEUE, the server MAY notify the Primary Site Controller of the owner site that the queue object was deleted by the client. This specification does not mandate how this notification is performed.<47>
- If the object type in the delete notification table entry is MQDS_QUEUE, and if the queue is not foreign, the server MUST notify the owner queue manager that the queue object was deleted by the client. This specification does not mandate how this notification is performed.

3.3.4.6 S_DSEndDeleteNotification (Opnum 5)

The **S_DSEndDeleteNotification** method closes the RPC context handle acquired from a previous call to **S_DSBeginDeleteNotification**.

```
void S_DSEndDeleteNotification(
  [in] handle_t hBind,
  [in, out] PPCONTEXT_HANDLE_DELETE_TYPE pHandle
);
```

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

pHandle: MUST be set by the client to a pointer to an RPC context_handle returned by a previous call to **S_DSBeginDeleteNotification**. The RPC context handle MUST NOT have

been used in a previous call to **S_DSEndDeleteNotification**. The server MUST set this parameter to NULL.

Return Values: None.

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

The server MUST perform the following processing:

- Look up the handle in the delete notification table. If not found, return an implementation-specific error code.
- Remove the entry from the delete notification table.
- Set *pHandle* to NULL.

3.3.4.7 S_DSIsServerGC (Opnum 6)

The **S_DSIsServerGC** method returns a value that indicates if that server is a **Global Catalog Server**.

```
long S_DSIsServerGC(
  [in] handle_t hBind
);
```

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

Return Values: This method returns TRUE (0x00000001) if the Directory Service server is also a Global Catalog Server; otherwise, it returns FALSE (0x00000000).

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

3.3.4.8 S_DSGetGCListInDomain (Opnum 8)

The **S_DSGetGCListInDomain** method returns the list of Global Catalog Servers in the specified domain.

```
HRESULT S_DSGetGCListInDomain(
  [in] handle_t hBind,
  [in, ptr] const wchar_t* lpwszComputerName,
  [in, ptr] const wchar_t* lpwszDomainName,
  [out, string] wchar_t** lplpwszGCList,
  [in] PCONTEXT_HANDLE_SERVER_AUTH_TYPE phServerAuth,
  [out, size_is(*pdwServerSignatureSize)]
    unsigned char* pbServerSignature,
  [in, out] DWORD* pdwServerSignatureSize
);
```

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

lpwszComputerName: MUST be set by the client to NULL. MUST be ignored by the server.

lpwszDomainName: MUST be set by the client to the domain name of the domain to query.

lplpwszGCList: MUST be set by the server to the list of Global Catalog Servers. The format of the list is a Server List String.

phServerAuth: A **PCONTEXT_HANDLE_SERVER_AUTH_TYPE** RPC context handle acquired from the *pphServerAuth* parameter in a previous call to **S_DSValidateServer**. The server MUST use this parameter as a key to locate the GSS security context used to compute the signature returned in *pbServerSignature*.

pbServerSignature: MUST be set by the server to a buffer that contains a signed hash over the returned list of Global Catalog Servers (*lplpwszGCList*) calculated by using the MD5 algorithm (as specified in [RFC1321]) and the GSS security context, as specified by the following pseudocode.

```
Initialize an MD5 Hash context
Add to the hash context the DWORD value 0x00000001
Add to the hash value the DWORD value of VT_BLOB
SET cbSize to the size in bytes of the Server List string in
    lplpwszGCList
Add to the hash value the DWORD value cbSize
Add to the hash context the contents of lplpwszGCList where the
      length for the data is cbSize
Call GSS_Wrap using the output_context_handle from GSS
      security context and the computed MD5 hash
SET pbServerSignature to the wrapped MD5 hash
SET *pdwServerSignatureSize to the size of the wrapped MD5 hash
```

pdwServerSignatureSize: MUST be set by the client to point to a **DWORD** that contains the maximum size in bytes of the server signature to return, and MUST be set by the server to contain the actual size in bytes of the server signature on output.

Return Values: If the method succeeds, the return value is MQ_OK (0x00000000). If the method fails, the return value is an implementation-specific error code.

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

3.3.5 Timer Events

There are no timer events.

3.3.6 Other Local Events

3.3.6.1 PCONTEXT_HANDLE_DELETE_TYPE Rundown

This event occurs when a **PCONTEXT_HANDLE_DELETE_TYPE** RPC context handle has been established between a client and server through a call to **S_DSBeginDeleteNotification**, and the RPC connection between the client and server is severed before the context handle has been closed via a call to **S_DSEndDeleteNotification**.

The server MUST use the context handle supplied as an event argument as a key to locate the delete notification entry in the delete notification table. The server MUST remove the delete notification entry from the table.

3.4 dscomm2 Client Details

3.4.1 Abstract Data Model

No abstract data model is required.

3.4.2 Timers

No protocol timers are required beyond those used internally by RPC to implement resiliency to network outages. See [MS-RPCE].

3.4.3 Initialization

Initialization is as specified in section 3.2.3.

3.4.4 Message Processing Events and Sequencing Rules

3.4.4.1 Send an Object Deleted Notification

To notify the server of an object deleted, the client MUST follow this sequence:

- The client MUST call the **S_DSBeginDeleteNotification** method with *pwcsPathName* set to the pathname of the directory object to delete, and with *phServerAuth* set to the context handle previously acquired from a call to **S_DSValidateServer**.
- The client MUST call the **S_DSNotifyDelete** method with the handle set to the context handle acquired from the previous call to **S_DSBeginDeleteNotification**.
- The client MUST call the **S_DSEndDeleteNotification** method with the handle set to the context handle acquired from the previous call to **S_DSBeginDeleteNotification**.

3.4.5 Timer Events

There are no client timer events for the **dscomm2** interface.

3.4.6 Other Local Events

There are no other local events for the **dscomm2** interface.

4 Protocol Examples

The following sections describe several methods used to allow the client and server to mutually authenticate.

4.1 S_DSValidateServer and S_InitSecCtx

Collectively, the **S_DSValidateServer** and **S_InitSecCtx** methods allow the client and server to mutually authenticate. These methods are used to tunnel GSS security context negotiation through the RPC interface. This protocol exchange is initiated when the client calls into its security provider via GSS to create a new security context. The security context returned is passed as the *pSecurityBuffer* parameter to **S_DSValidateServer**.

The server calls into its security provider via GSS to accept this security buffer. The server then calls the client's **S_InitSecCtx** method, passing the security buffer. The client then calls its own security provider via GSS to accept and possibly modify the security buffer. If both client and server signal that they need another round of exchanges by returning SEC_I_CONTINUE_NEEDED, the process repeats with the contents of the security buffer being passed to (and potentially modified by) both client and server security providers in turn.

If either client or server signals an error via an error return code, the protocol exchange halts and the **S_DSValidateServer** method returns that error code to the initial caller.

When both client and server security providers signal completion via a success return code from their respective GSS calls, the protocol exchange is complete and the **S_DSValidateServer** method returns successfully.

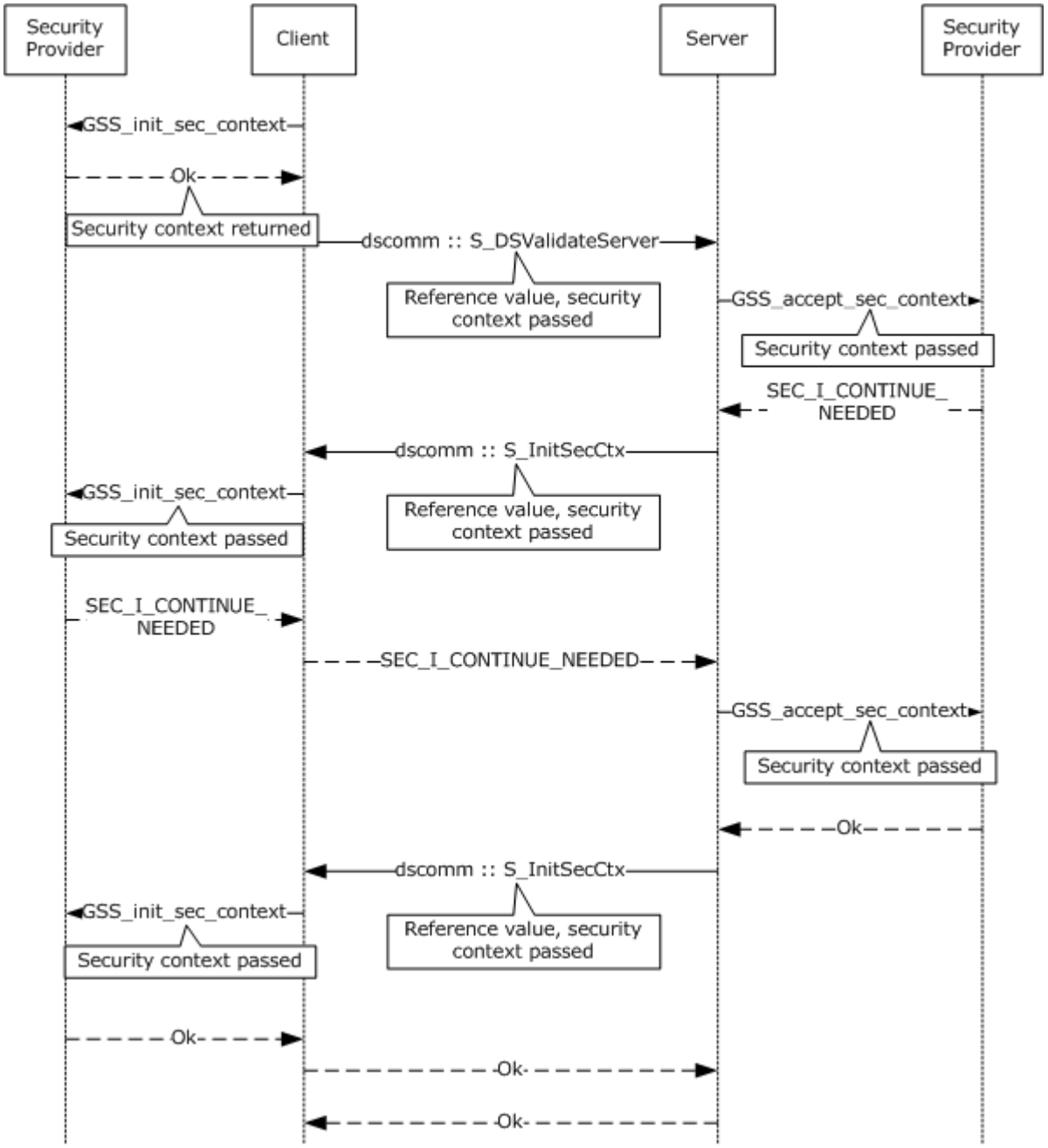


Figure 4: Protocol exchange complete

4.2 S_DSQMGetObjectSecurity and S_DSQMGetObjectSecurityChallengeResponceProc

Collectively, the **S_DSQMGetObjectSecurity** and **S_DSQMGetObjectSecurityChallengeResponceProc** methods allow the server to authenticate the client in the context of retrieving security information related to a directory service object. The client makes the initial call to **S_DSQMGetObjectSecurity** and includes a correlation ID in the

dwContext parameter. The server generates a random challenge via a call into its security provider. The server then invokes the **S_DSQMGetObjectSecurityChallengeResponceProc** on the client, passing both the random challenge and the correlation ID from the *dwContext* parameter in the initial call to **S_DSQMGetObjectSecurity**.

The client then calculates the MD5 hash (as specified in [RFC1321]) of the random challenge bytes and signs this hash by using the private key corresponding to the public key associated with the directory object as registered in the directory service. If the directory service object for which the security identifier is retrieved is a queue, the private key MUST be the private key associated with the machine that owns the queue; otherwise, the private key MUST be the private key associated with the directory service object. This signed hash is returned to the server in the *abCallengeResponce* parameter.

The server then calculates the MD5 hash (as specified in [RFC1321]) of the random challenge bytes and verifies that the appropriate public key verifies the signed hash. If the directory service object for which the security identifier is retrieved is a queue, the public key MUST be the public key associated with the machine that owns the queue; otherwise, the public key MUST be the public key associated with the directory service object.

If either client or server signals an error via an error return code, the protocol exchange halts and the **S_DSQMGetObjectSecurity** method returns that error code to the initial caller.

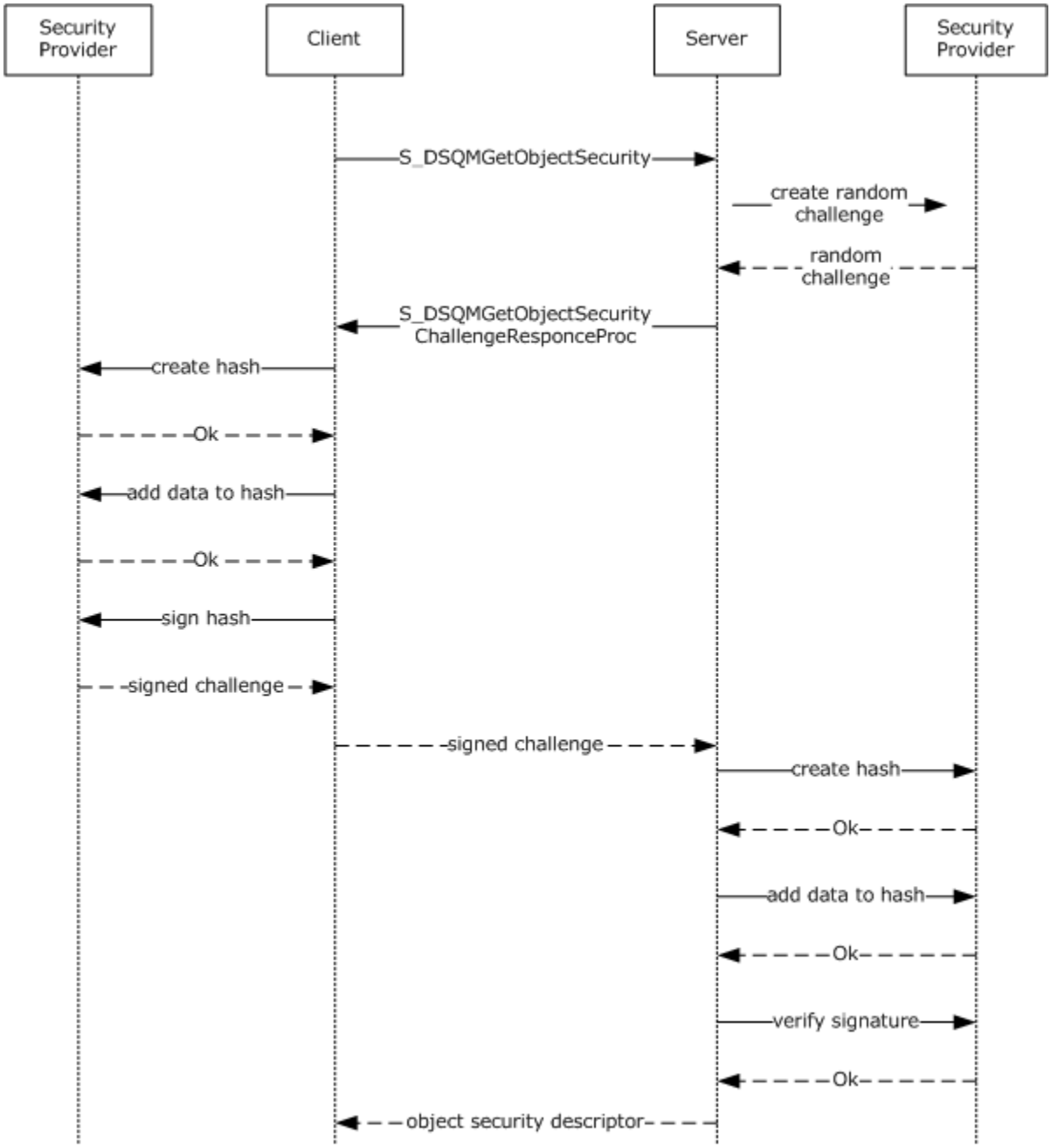


Figure 5: The S_DSQMGetObjectSecurity method

4.3 S_DSLookupBegin, S_DSLookupNext, and S_DSLookupEnd

Collectively, the methods **S_DSLookupBegin**, **S_DSLookupNext**, and **S_DSLookupEnd** allow the client to perform ad-hoc queries against the directory service.

This protocol exchange begins when the client calls **S_DSLookupBegin** with a set of property restrictions, a set of properties to be returned, and a sort key. The server responds with an RPC

context handle that represents the in-progress query. The client then calls **S_DSLookupNext** repeatedly, passing the RPC context handle each time; the server returns either an error code or the specified properties for the next directory service object for each such call. When there are no more objects in the query, the server returns a success result code with 0 properties. When the client has completed processing all results, the client calls **S_DSLookupEnd**, passing the RPC context handle.

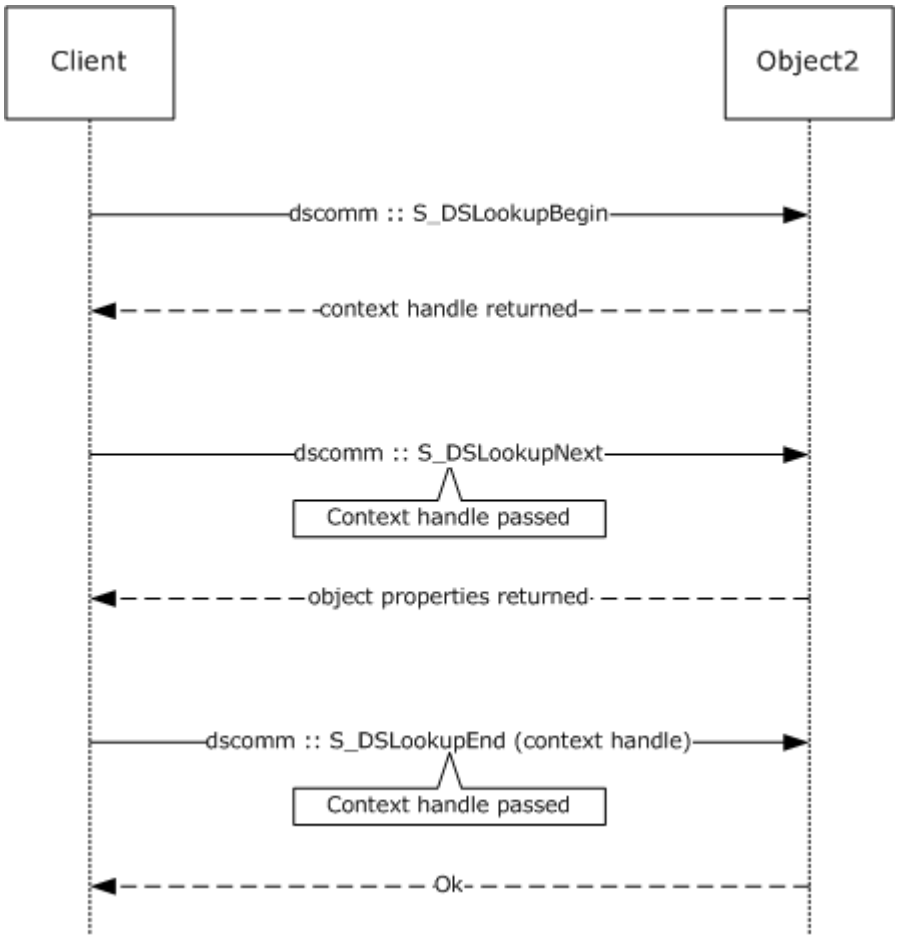


Figure 6: Calling the S_DSLookupEnd method

4.4 S_DSBeginDeleteNotification, S_DSNotifyDelete, and S_DSEndDeleteNotification

Collectively, the **S_DSBeginDeleteNotification**, **S_DSNotifyDelete**, and **S_DSEndDeleteNotification** methods are used by the client to notify the server that a queue has been deleted by the client.

This protocol exchange begins when the client calls **S_DSBeginDeleteNotification** specifying either the machine or queue to be deleted. The server obtains information on this machine or queue and retains it for later use, associating it with a delete context handle. The server returns the delete context handle to the client. The client calls **S_DSNotifyDelete** (specifying the delete context handle) after it has deleted the machine or queue. The client then calls **S_DSEndDeleteNotification** (specifying the delete context handle), which indicates to the server that it is to clean up the state retained from the call to **S_DSBeginDeleteNotification**.

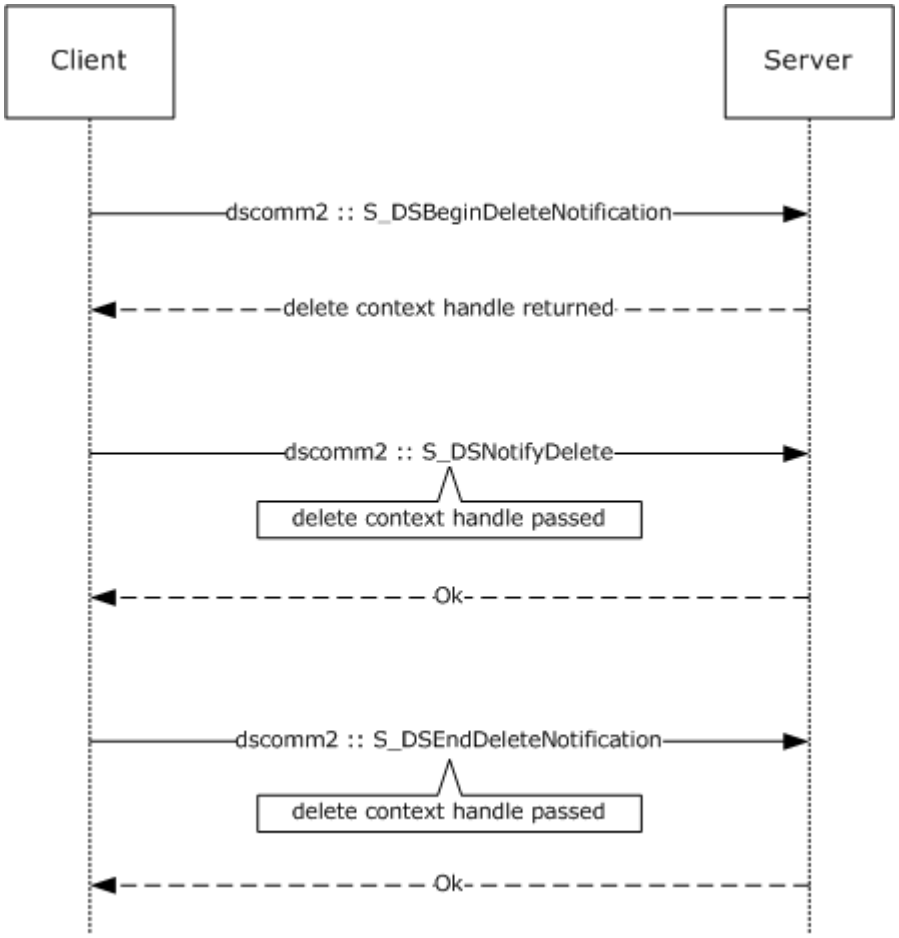


Figure 7: The S_DSBeginDeleteNotification method

5 Security

The following sections specify security considerations for all server implementations.

5.1 Security Considerations for Implementers

As specified in section 2.1, this protocol allows any user to connect to the server. Therefore, any security bug in the server implementation could be exploitable. The server implementation should enforce security on each method.

5.2 Index of Security Parameters

Security Parameter	Section
Authentication Protocol	2.1

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, and "ms-mqmq.idl" is the IDL specified in [MS-MQMQ] Appendix A.

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

const unsigned long PRLT        = 0;
const unsigned long PRLE        = 1;
const unsigned long PRGT        = 2;
const unsigned long PRGE        = 3;
const unsigned long PREQ        = 4;
const unsigned long PRNE        = 5;

const unsigned long PRRE        = 6;
const unsigned long PRAllBits   = 7;
const unsigned long PRSomeBits  = 8;
const unsigned long PRAll       = 0x100;
const unsigned long PRAny       = 0x200;

typedef struct  tagMQPROPERTYRESTRICTION
{
        unsigned long rel;
        unsigned long prop;
        PROPVARIANT   prval;
} MQPROPERTYRESTRICTION;

typedef struct  tagMQRESTRICTION
{
        unsigned long cRes;
        [size is(cRes)] MQPROPERTYRESTRICTION  * paPropRes;

}       MQRESTRICTION;

typedef struct  tagMQCOLUMNSET
{
    unsigned long cCol;
    [size is(cCol)] PROPID  *aCol;
} MQCOLUMNSET;

const unsigned long QUERY SORTASCEND     = 0;
const unsigned long QUERY SORTDESCEND    = 1;

typedef struct  tagMQSORTKEY
{
    unsigned long propColumn;
    unsigned long dwOrder;
} MQSORTKEY;

typedef struct  tagMQSORTSET
{
    unsigned long cCol;
    [size is(cCol)] MQSORTKEY  *aCol;
} MQSORTSET;

[
version(1.0),
uuid(77df7a80-f298-11d0-8358-00a024c480a8),
pointer default(unique)
]
```

```
interface dscomm
{
/*================================================
 Structures
================================================*/

typedef struct SERVER AUTH STRUCT tag {
  void* pvReserved;
  unsigned long ulReserved1;
  unsigned long ulReserved2;
} SERVER AUTH STRUCT;

/*================================================
 RPC Context Handles
================================================*/
typedef [context handle] void * PCONTEXT HANDLE TYPE;
typedef [ref] PCONTEXT HANDLE TYPE * PPCONTEXT HANDLE TYPE;

typedef [context handle] SERVER AUTH STRUCT
                         *PCONTEXT_HANDLE_SERVER_AUTH_TYPE;
typedef [ref] PCONTEXT HANDLE SERVER AUTH TYPE
              *PPCONTEXT HANDLE SERVER AUTH TYPE;

typedef [context handle] void * PCONTEXT HANDLE DELETE TYPE;
typedef [ref] PCONTEXT_HANDLE_DELETE_TYPE
              *PPCONTEXT_HANDLE_DELETE_TYPE;

/*============================================================================
 MQDS API
============================================================================*/

HRESULT
S_DSCreateObject(
    [in] handle t hBind,
    [in] unsigned long dwObjectType,
    [in, string, unique] const wchar t * pwcsPathName,
    [in] unsigned long dwSDLength,
    [in, size_is(dwSDLength), unique] char * SecurityDescriptor,
    [in] unsigned long cp,
    [in, size is(cp)] unsigned long aProp[],
    [in, size is(cp)] PROPVARIANT apVar[],
    [in, out, unique] GUID* pObjGuid
);

HRESULT
S DSDeleteObject(
    [in] handle t hBind,
    [in] unsigned long dwObjectType,
    [in, string] const wchar_t * pwcsPathName
);

HRESULT
S DSGetProps(
    [in] handle t hBind,
    [in] unsigned long dwObjectType,
    [in, string] const wchar_t * pwcsPathName,
    [in] unsigned long cp,
    [in, size is(cp)] unsigned long aProp[],
    [in, out, size is(cp)] PROPVARIANT apVar[],
    [in] PCONTEXT HANDLE SERVER AUTH TYPE phServerAuth,
    [out, size is(*pdwServerSignatureSize)]
          unsigned char * pbServerSignature,
    [in, out] unsigned long * pdwServerSignatureSize
);
```

```
HRESULT
S_DSSetProps(
    [in] handle_t hBind,
    [in] unsigned long dwObjectType,
    [in, string] const wchar t * pwcsPathName,
    [in] unsigned long cp,
    [in, size is(cp)] unsigned long aProp[],
    [in, size_is(cp)] PROPVARIANT apVar[]
);

HRESULT
S DSGetObjectSecurity(
    [in] handle t hBind,
    [in] unsigned long dwObjectType,
    [in, string] const wchar_t * pwcsPathName,
    [in] unsigned long SecurityInformation,
    [out, size is(nLength)] unsigned char* pSecurityDescriptor,
    [in] unsigned long nLength,
    [out] unsigned long* lpnLengthNeeded,
    [in] PCONTEXT_HANDLE_SERVER_AUTH_TYPE phServerAuth,
    [out, size is(*pdwServerSignatureSize)]
          unsigned char * pbServerSignature,
    [in, out] unsigned long * pdwServerSignatureSize
);

HRESULT
S_DSSetObjectSecurity(
    [in] handle t hBind,
    [in] unsigned long dwObjectType,
    [in, string] const wchar t * pwcsPathName,
    [in] unsigned long SecurityInformation,
    [in, unique, size_is(nLength)] unsigned char* pSecurityDescriptor,
    [in] unsigned long nLength
);

HRESULT
S DSLookupBegin(
    [in] handle_t hBind,
    [out] PPCONTEXT_HANDLE_TYPE pHandle,
    [in, unique, string] wchar t * pwcsContext,
    [in, unique] MQRESTRICTION* pRestriction,
    [in, ref] MQCOLUMNSET* pColumns,
    [in, unique] MQSORTSET* pSort,
    [in] PCONTEXT_HANDLE_SERVER_AUTH_TYPE phServerAuth
);

HRESULT
S DSLookupNext(
    [in] handle_t hBind,
    [in] PCONTEXT_HANDLE_TYPE Handle,
    [in] unsigned long* dwSize,
    [out] unsigned long* dwOutSize,
    [out, size is(*dwSize), length is(*dwOutSize)] PROPVARIANT pbBuffer[],
    [in] PCONTEXT HANDLE SERVER AUTH TYPE phServerAuth,
    [out, size_is(*pdwServerSignatureSize)]
          unsigned char * pbServerSignature,
    [in, out] unsigned long * pdwServerSignatureSize
);

HRESULT
S DSLookupEnd(
    [in] handle_t hBind,
    [in, out] PPCONTEXT_HANDLE_TYPE phContext
);
```

```
void Opnum9NotUsedOnWire(void);

HRESULT
S DSDeleteObjectGuid(
    [in] handle t hBind,
    [in] unsigned long dwObjectType,
    [in] const GUID* pGuid
 );

HRESULT
S DSGetPropsGuid(
    [in] handle t hBind,
    [in] unsigned long dwObjectType,
    [in, unique] const GUID* pGuid,
    [in] unsigned long cp,
    [in, size is(cp)] unsigned long aProp[],
    [in, out, size is(cp)] PROPVARIANT apVar[],
    [in] PCONTEXT HANDLE SERVER AUTH TYPE phServerAuth,
    [out, size is(*pdwServerSignatureSize)]
          unsigned char * pbServerSignature,
    [in, out] unsigned long * pdwServerSignatureSize
);

HRESULT
S_DSSetPropsGuid(
    [in] handle_t hBind,
    [in] unsigned long dwObjectType,
    [in] const GUID * pGuid,
    [in] unsigned long cp,
    [in, size is(cp)] unsigned long aProp[],
    [in, size is(cp)] PROPVARIANT apVar[]
);

HRESULT
S DSGetObjectSecurityGuid(
    [in] handle t hBind,
    [in] unsigned long dwObjectType,
    [in] const GUID * pGuid,
    [in] unsigned long SecurityInformation,
    [out, size is(nLength)] unsigned char* pSecurityDescriptor,
    [in] unsigned long nLength,
    [out] unsigned long* lpnLengthNeeded,
    [in] PCONTEXT_HANDLE_SERVER_AUTH_TYPE phServerAuth,
    [out, size_is(*pdwServerSignatureSize)]
          unsigned char * pbServerSignature,
    [in, out] unsigned long * pdwServerSignatureSize
);

HRESULT
S_DSSetObjectSecurityGuid(
    [in] handle t hBind,
    [in] unsigned long dwObjectType,
    [in] const GUID * pGuid,
    [in] unsigned long SecurityInformation,
    [in, unique, size_is(nLength)]
         unsigned char* pSecurityDescriptor,
    [in] unsigned long nLength
);

void Opnum15NotUsedOnWire(void);

void Opnum16NotUsedOnWire(void);

void Opnum17NotUsedOnWire(void);
```

```
void Opnum18NotUsedOnWire(void);

HRESULT
S DSQMSetMachineProperties(
    [in] handle t hBind,
    [in, string] const wchar t * pwcsPathName,
    [in] unsigned long cp,
    [in, size_is(cp)] unsigned long aProp[],
    [in, size_is(cp)] PROPVARIANT apVar[],
    [in] unsigned long dwContext
);

HRESULT
S_DSCreateServersCache(
    [in] handle_t hBind,
    [in, out] unsigned long * pdwIndex,
    [in, out, ptr, string] wchar t * * lplpSiteServers,
    [in] PCONTEXT HANDLE SERVER AUTH TYPE phServerAuth,
    [out, size is(*pdwServerSignatureSize)]
         unsigned char * pbServerSignature,
    [in, out] unsigned long * pdwServerSignatureSize
);

[callback]
HRESULT
S_DSQMSetMachinePropertiesSignProc(
    [in, size_is(dwCallengeSize)] byte *abChallenge,
    [in] unsigned long dwCallengeSize,
    [in] unsigned long dwContext,
    [in, out, size is(dwSignatureMaxSize), length is(*pdwSignatureSize)]
         byte * abSignature,
    [in, out] unsigned long * pdwSignatureSize,
    [in] unsigned long dwSignatureMaxSize
);

HRESULT
S DSQMGetObjectSecurity(
    [in] handle_t hBind,
    [in] unsigned long dwObjectType,
    [in] const GUID * pGuid,
    [in] unsigned long SecurityInformation,
    [out, size is(nLength)] unsigned char* pSecurityDescriptor,
    [in] unsigned long nLength,
    [out] unsigned long* lpnLengthNeeded,
    [in] unsigned long dwContext,
    [in] PCONTEXT HANDLE SERVER AUTH TYPE phServerAuth,
    [out, size is(*pdwServerSignatureSize)]
         unsigned char * pbServerSignature,
    [in, out] unsigned long * pdwServerSignatureSize
);

[callback]
HRESULT
S DSQMGetObjectSecurityChallengeResponceProc(
    [in, size_is(dwCallengeSize)] byte * abChallenge,
    [in] unsigned long dwCallengeSize,
    [in] unsigned long dwContext,
    [in, out, size is(dwCallengeResponceMaxSize),
              length is(*pdwCallengeResponceSize)]
              byte * abCallengeResponce,
    [in, out] unsigned long *pdwCallengeResponceSize,
    [in] unsigned long dwCallengeResponceMaxSize
);

[callback]
```

```
HRESULT
S_InitSecCtx(
    [in] unsigned long dwContext,
    [in, size is(dwServerBuffSize)] unsigned char * pServerbuff,
    [in] unsigned long dwServerBuffSize,
    [in] unsigned long dwClientBuffMaxSize,
    [out, size is(dwClientBuffMaxSize), length is(*pdwClientBuffSize)]
          unsigned char * pClientBuff,
    [out] unsigned long * pdwClientBuffSize
);

HRESULT
S DSValidateServer(
    [in] handle_t hBind,
    [in] const GUID * pguidEnterpriseId,
    [in] long fSetupMode,
    [in] unsigned long dwContext,
    [in] unsigned long dwClientBuffMaxSize,
    [in, size is(dwClientBuffMaxSize), length is(dwClientBuffSize)]
         unsigned char * pClientBuff,
    [in] unsigned long dwClientBuffSize,
    [out] PPCONTEXT HANDLE SERVER AUTH TYPE pphServerAuth
);

HRESULT
S_DSCloseServerHandle(
    [in, out] PPCONTEXT_HANDLE_SERVER_AUTH_TYPE pphServerAuth
);

void Opnum24NotUsedOnWire(void);

void Opnum25NotUsedOnWire(void);

void Opnum26NotUsedOnWire(void);

unsigned long
S DSGetServerPort(
    [in] handle_t hBind,
    [in] unsigned long fIP
);

}

[
    version(1.0),
    uuid(708cca10-9569-11d1-b2a5-0060977d8118),
    pointer default(unique)
]
interface dscomm2
{
HRESULT S DSGetComputerSites(
[in] handle t hBind,
    [in, unique] const wchar t * pwcsPathName,
    [out] DWORD * pdwNumberOfSites,
    [out, size_is(,*pdwNumberOfSites), length_is(,*pdwNumberOfSites)]
          GUID ** ppguidSites,
    [in] PCONTEXT HANDLE SERVER AUTH TYPE phServerAuth,
    [out, size is(*pdwServerSignatureSize)]
          unsigned char * pbServerSignature,
    [in, out] DWORD * pdwServerSignatureSize
);

HRESULT S_DSGetPropsEx(
[in] handle t hBind,
[in, range(1, 58)] DWORD dwObjectType,
```

```
[in] const wchar t * pwcsPathName,
[in, range(1,128)] DWORD cp,
[in, size_is(cp)] PROPID aProp[],
[in, out, size is(cp)] PROPVARIANT apVar[],
[in] PCONTEXT HANDLE SERVER AUTH TYPE phServerAuth,
[out, size is(*pdwServerSignatureSize)]
      unsigned char * pbServerSignature,
[in, out] DWORD * pdwServerSignatureSize
);

HRESULT S DSGetPropsGuidEx(
[in] handle t hBind,
[in, range(1, 58)] DWORD dwObjectType,
[in, unique] const GUID * pGuid,
[in, range(1,128)] DWORD cp,
[in, size is(cp)] PROPID aProp[],
[in, out, size is(cp)] PROPVARIANT apVar[],
[in] PCONTEXT HANDLE SERVER AUTH TYPE phServerAuth,
[out, size is(*pdwServerSignatureSize)]
      unsigned char * pbServerSignature,
[in, out] DWORD * pdwServerSignatureSize
);

HRESULT S DSBeginDeleteNotification(
[in] handle_t hBind,
[in] const wchar_t * pwcsPathName,
[out] PPCONTEXT_HANDLE_DELETE_TYPE pHandle,
[in] PCONTEXT HANDLE SERVER AUTH TYPE phServerAuth
);

HRESULT S DSNotifyDelete(
[in] handle_t hBind,
[in] PCONTEXT_HANDLE_DELETE_TYPE Handle
);

void S DSEndDeleteNotification(
[in] handle t hBind,
[in, out] PPCONTEXT_HANDLE_DELETE_TYPE pHandle
);

long S DSIsServerGC(
[in] handle t hBind
);

void Opnum7NotUsedOnWire(void);

HRESULT S DSGetGCListInDomain(
[in] handle t hBind,
[in, ptr] const wchar_t * lpwszComputerName,
[in, ptr] const wchar_t * lpwszDomainName,
[out, string] wchar t ** lplpwszGCList,
[in] PCONTEXT HANDLE SERVER AUTH TYPE phServerAuth,
[out, size is(*pdwServerSignatureSize)]
      unsigned char * pbServerSignature,
[in, out] DWORD * pdwServerSignatureSize
);
}
```

7 Appendix B: Windows Behavior

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

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

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

<1> Section 1.4: Windows NT Server contains a directory service that implements the server side of the Message Queuing (MSMQ): Directory Service Protocol. All other versions of Windows Server use Active Directory and implement the server side of this protocol as a pass-through to Active Directory.

Windows NT Server, Windows 2000, Windows XP, and Windows Server 2003 implement the client side of this protocol. Windows XP and Windows Server 2003 use [MS-ADTS] to Active Directory when it is available; otherwise, they use this protocol. Windows Vista and Windows Server 2008 always use [MS-ADTS].

<2> Section 1.6: Windows NT Server contains a directory service that implements the server side of the Message Queuing (MSMQ): Directory Service Protocol. All other versions of Windows Server use Active Directory and implement the server side of this protocol as a pass-through to Active Directory.

Windows NT Server, Windows 2000, Windows XP, and Windows Server 2003 implement the client side of this protocol. Windows XP and Windows Server 2003 use [MS-ADTS] to Active Directory when it is available; otherwise, they use this protocol. Windows Vista and Windows Server 2008 always use [MS-ADTS].

<3> Section 1.7: Windows NT Server,Windows 2000 Server , and Windows Server 2003 implement the client and server side of the dscomm interface specified in section 3.1. Windows 2000 Server and Windows Server 2003 implement the client and server side of the dscomm2 interface specified in section 3.3.

Windows NT Server, Windows 2000, Windows XP, and Windows Server 2003 implement the client side of the dscomm interface specified in section 3.1. Windows 2000 Server, Windows XP, and Windows Server 2003 implement the client side of the dscomm2 interface specified in section 3.3.

<4> Section 2.1: All versions of Windows can be configured via the registry to use a static port. The server uses registry key HKLM\SOFTWARE\Microsoft\MSMQ\Parameters\UseDSPredefinedEP to determine if the predefined endpoint should be used instead of a dynamic endpoint. Setting this key to a nonzero **DWORD** value causes the predefined endpoint to be used. The endpoint port number defaults to 2879 but can be configured by setting registry key HKLM\SOFTWARE\Microsoft\MSMQ\Parameters\MsmqDSRpcIpPort to a **DWORD** value equal to the preferred endpoint port number.

<5> Section 2.1: Windows NT 4.0 and Windows 2000 support RPC over SPX. No other versions of Windows support RPC over SPX.

<6> Section 2.1: Windows NT supports only NTLM. All other versions of Windows support both NTLM and Kerberos.

<7> Section 2.2.20: Other key BLOB types may be supported but are untested. All versions of Windows use the PUBLICKEYBLOB BLOB type.

<8> Section 2.2.20: Other key algorithms may be supported but are untested. All versions of Windows use only CALG_RSA_KEYX or CALG_RSA_SIGN algorithms.

<9> Section 3.1.3: Section 3.1.3: All versions of Windows support both dynamic and static endpoints. The server uses registry key HKLM\SOFTWARE\Microsoft\MSMQ\Parameters\UseDSPredefinedEP to determine if the predefined endpoint should be used instead of a dynamic endpoint. Setting this key to a nonzero **DWORD** value causes the predefined endpoint to be used. The endpoint port number defaults to 2879, but can be configured by setting registry key HKLM\SOFTWARE\Microsoft\MSMQ\Parameters\MsmqDSRpcIpPort to a **DWORD** value equal to the preferred endpoint port number.

<10> Section 3.1.4: Gaps in the opnum numbering sequence apply to Windows as follows.

Opnum	Description
9	For Windows NT 4.0, this method is not used. For all other versions of Windows, this method returns MQ_ERROR_FUNCTION_NOT_SUPPORTED. It is never used.
15	For Windows NT 4.0, this method is not used. For all other versions of Windows, this method returns MQ_ERROR_FUNCTION_NOT_SUPPORTED. It is never used.
16	For Windows NT 4.0, this method is not used. For all other versions of Windows, this method returns MQ_ERROR_FUNCTION_NOT_SUPPORTED. It is never used.
17	For Windows NT 4.0, this method is not used. For all other versions of Windows, this method returns MQ_ERROR_FUNCTION_NOT_SUPPORTED. It is never used.
18	This method is deprecated and not used.
24	For Windows NT 4.0, this method is not used. For all other versions of Windows, this method returns MQ_ERROR_FUNCTION_NOT_SUPPORTED. It is never used.
25	For Windows NT 4.0, this method is not used. For all other versions of Windows, this method returns MQ_ERROR_FUNCTION_NOT_SUPPORTED. It is never used.
26	For Windows NT 4.0, this method is not used. For all other versions of Windows, this method returns MQ_ERROR_FUNCTION_NOT_SUPPORTED. It is never used.

<11> Section 3.1.4.1: Windows NT and Windows 2000 support IPX bindings; no other versions of Windows support IPX bindings.

<12> Section 3.1.4.2: If the *fSetup* parameter is nonzero, Windows NT verifies that the *pguidEnterpriseId* parameter matches the **GUID** of the enterprise owning the site to which the server belongs. All other versions of Windows ignore this parameter.

<13> Section 3.1.4.2: Windows NT uses this parameter to determine if the *pguidEnterpriseId* parameter should be validated. All other versions of Windows ignore this parameter.

<14> Section 3.1.4.4: Windows NT allows the creation of MQDS_ENTERPRISE objects. All other versions of Windows disallow the creation of MQDS_ENTERPRISE objects and return MQ_ERROR_ILLEGAL_ENTERPRISE_OPERATION (0xC00E0071) instead.

<15> Section 3.1.4.4: If *dwObjectType* is MQDS_USER, and *SecurityDescriptor* is not NULL, Windows NT ignores *SecurityDescriptor* while all other versions of Windows return MQ_ERROR_ILLEGAL_PROPERTY_VALUE (0xC00E0018).

<16> Section 3.1.4.4: If *dwObjectType* is not MQDS_QUEUE, and *SecurityDescriptor* is not NULL, all versions of Windows except Windows NT return MQ_ERROR_ILLEGAL_PROPERTY_VALUE (0xC00E0018).

<17> Section 3.1.4.4: Windows NT ignores the supplied security descriptor if the *SecurityDescriptor* parameter is not NULL. All other versions of Windows return MQ_ERROR_ILLEGAL_PROPERTY_VALUE (0xC00E0018) if the *SecurityDescriptor* parameter is not NULL.

<18> Section 3.1.4.4: Windows NT uses the supplied security descriptor if the *SecurityDescriptor* parameter is not NULL. All other versions of Windows return MQ_ERROR_ILLEGAL_PROPERTY_VALUE (0xC00E0018) if the *SecurityDescriptor* parameter is not NULL.

<19> Section 3.1.4.8: All versions of Windows return MQ_ERROR_ILLEGAL_PROPID (0xC00E0039) in this case.

<20> Section 3.1.4.9: All versions of Windows return MQ_ERROR_ILLEGAL_PROPID (0xC00E0039) in this case.

<21> Section 3.1.4.10: All versions of Windows return MQ_ERROR_ILLEGAL_PROPID (0xC00E0039) in this case.

<22> Section 3.1.4.11: Other key BLOB types may be supported but are untested. All versions of Windows use only RSA_PUBLIC_KEY BLOBs.

<23> Section 3.1.4.12: Other key BLOB types may be supported but are untested. All versions of Windows use only RSA_PUBLIC_KEY BLOBs.

<24> Section 3.1.4.13: Other key BLOB types may be supported but are untested. All versions of Windows use only RSA_PUBLIC_KEY BLOB.

<25> Section 3.1.4.14: Other key BLOB types may be supported but are untested. All versions of Windows use only RSA_PUBLIC_KEY BLOBs.

<26> Section 3.1.4.15: This method is called by Windows NT 4.0. No other version of Windows calls this method.

<27> Section 3.1.4.15: Windows NT Server, Windows 2000, Windows XP, and Windows Server 2003 invoke the **S_DSQMGetObjectSecurityChallengeResponceProc** callback method; no other version of Windows invokes this method.

<28> Section 3.1.4.16: The **S_DSQMSetMachineProperties** method is called by Windows NT 4.0. No other version of Windows calls this method.

<29> Section 3.1.4.16: Windows NT Server, Windows 2000, Windows XP, and Windows Server 2003 invoke the **S_DSQMSetMachinePropertiesSignProc** callback method; no other version of Windows invokes this method.

<30> Section 3.1.4.16: All versions of Windows return MQ_ERROR_ILLEGAL_PROPID (0xC00E0039) in this case.

<31> Section 3.2.1: All versions of Windows clients store their site identifiers in registry key HKLM\Software\Microsoft\MSMQ\Parameters\MachineCache\SiteId.

<32> Section 3.2.1: Windows NT and Windows 2000 clients get the initial known directory service server from the user during the setup process and store it as both the current server and as a list of one known server. No other Windows client uses this protocol.

<33> Section 3.2.1: Windows NT and Windows 2000 clients store the list of known directory service servers in registry key HKLM\Software\Microsoft\MSMQ\Parameters\MachineCache\MQISServer, and store the current server in registry key HKLM\Software\Microsoft\MSMQ\Parameters\MachineCache\CurrentMQISServer. No other Windows client uses this protocol.

<34> Section 3.2.3: Windows NT and Windows 2000 protocol clients call the **S_DSGetServerPort** method; no other versions of Windows call this method.

<35> Section 3.2.4.1: Windows NT calls this method. All other versions of Windows return MQ_ERROR_NOT_SUPPORTED (0xC00E03EB).

<36> Section 3.2.4.2: Windows NT calls this method. All other versions of Windows return MQ_ERROR_NOT_SUPPORTED (0xC00E03EB).

<37> Section 3.2.6.1: Windows NT and Windows 2000 clients perform this event during initialization. No other Windows client uses this event.

<38> Section 3.3.1.1: The object identifier of the site is not required if the owner is Active Directory.

<39> Section 3.3.3: All versions of Windows support both dynamic and static endpoints. The server uses registry key HKLM\SOFTWARE\Microsoft\MSMQ\Parameters\UseDSPredefinedEP to determine if the predefined endpoint should be used instead of a dynamic endpoint. Setting this key to a nonzero **DWORD** value causes the predefined endpoint to be used. The endpoint port number defaults to 2879 but can be configured by setting registry key HKLM\SOFTWARE\Microsoft\MSMQ\Parameters\MsmqDSRpcIpPort to a **DWORD** value equal to the preferred endpoint port number.

<40> Section 3.3.4: Gaps in the opnum numbering sequence apply to Windows as follows.

Opnum	Description
7	This method is deprecated and not used.

<41> Section 3.3.4.1: For Windows 2000 and Windows XP, the server returns an error if a NULL value is specified for *pwcsPathName*.

<42> Section 3.3.4.4: All versions of Windows assume that *pwcsPathName* specifies a queue name if it contains a backslash "\"; otherwise, *pwcsPathName* specifies a machine name.

<43> Section 3.3.4.4: If the named object is a queue and the queue is owned by a Windows NT PSC, Windows 2000 and Windows XP return MQ_INFORMATION_QUEUE_OWNED_BY_NT4_PSC (0x400E03EB). If the named object is a machine, and the machine is owned by a Windows NT PSC,

Windows 2000 and Windows XP return MQ_INFORMATION_MACHINE_OWNED_BY_NT4_PSC (0x400E03EB).

<44> Section 3.3.4.4: All versions of Windows assume that *pwcsPathName* specifies a queue name if it contains a backslash "\"; otherwise, *pwcsPathName* specifies a machine name.

<45> Section 3.3.4.4: The object identifier of the site is not required if the owner is the Active Directory.

<46> Section 3.3.4.5: For Windows 2000 and Windows XP, the server notifies the Primary Site Controller if the queue is owned by the Primary Site Controller. Windows Server 2003, Windows Vista, and Windows Server 2008 do not notify the Primary Site Controller.

<47> Section 3.3.4.5: For Windows 2000 and Windows XP, the server notifies the Primary Site Controller if the queue is owned by the Primary Site Controller. Windows Server 2003, Windows Vista, and Windows Server 2008 do not notify the Primary Site Controller.

8 Index

L

M

N

O

P

Q

R

S

T

U

V

W