

# [MS-MSRP]: Messenger Service Remote Protocol Specification

---

## Intellectual Property Rights Notice for Protocol Documentation

- This protocol documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you may make copies of it in order to develop implementations of the protocols, and may distribute portions of it in your implementations of the protocols or your documentation as necessary to properly document the implementation. This permission also applies to any documents that are referenced in the protocol documentation.
- Microsoft does not claim any trade secret rights in this documentation.
- Microsoft has patents that may cover your implementations of the protocols. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. If you are interested in obtaining a patent license, please contact [protocol@microsoft.com](mailto:protocol@microsoft.com).
- The names of companies and products contained in this documentation may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights.
- All other rights are reserved, and this notice does not grant any rights other than specifically described above, whether by implication, estoppel, or otherwise.

This protocol documentation is intended for use in conjunction with publicly available standard specifications, network programming art, and Microsoft Windows distributed systems concepts, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

A protocol specification does not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments you are free to take advantage of them.

## Revision Summary

Date	Revision History	Revision Class	Comments
05/11/2007	0.1		MCPD Milestone 4 Initial Availability
08/10/2007	1.0	Major	Updated and revised the technical content.
09/28/2007	2.0	Major	Made a change to the IDL.
10/23/2007	3.0	Major	Updated and revised the technical content.
11/30/2007	4.0	Major	Sections 2 and 3 modified for template conformance.

<b>Date</b>	<b>Revision History</b>	<b>Revision Class</b>	<b>Comments</b>
			Content was moved between section 2 and 3.
01/25/2008	4.0.1	Editorial	Revised and edited the technical content.

# Table of Contents

<b>1</b>	<b>Introduction .....</b>	<b>5</b>
1.1	Glossary .....	5
1.2	References .....	5
1.2.1	Normative References .....	5
1.2.2	Informative References.....	6
1.3	Protocol Overview (Synopsis).....	6
1.4	Relationship to Other Protocols.....	7
1.5	Prerequisites/Preconditions .....	7
1.6	Applicability Statement .....	7
1.7	Versioning and Capability Negotiation.....	7
1.8	Vendor-Extensible Fields .....	7
1.9	Standards Assignments.....	7
<b>2</b>	<b>Messages .....</b>	<b>9</b>
2.1	Transport .....	9
2.1.1	RPC Transport .....	9
2.1.2	Mailslots .....	9
2.1.3	SMB.....	9
2.2	Message Syntax .....	10
2.2.1	Constants .....	10
2.2.1.1	Windows Error Codes .....	10
2.2.2	Data Types .....	11
2.2.2.1	NET_API_STATUS.....	11
2.2.2.2	MSGSVCS_HANDLE .....	11
2.2.3	Structures .....	12
2.2.3.1	MSG_INFO_0 .....	12
2.2.3.2	MSG_INFO_1 .....	12
2.2.3.3	MSG_INFO_0_CONTAINER.....	12
2.2.3.4	MSG_INFO_1_CONTAINER.....	13
2.2.3.5	MSG_ENUM_STRUCT .....	13
2.2.3.6	MSG_INFO .....	14
2.2.4	SMB Message Delivery Protocol.....	14
2.2.4.1	SMB_COM_SEND_MESSAGE Request and Response Messages .....	14
2.2.4.1.1	SMB_COM_SEND_MESSAGE Request Message .....	14
2.2.4.1.2	SMB_COM_SEND_MESSAGE Response Message .....	16
2.2.4.2	SMB_COM_SEND_START_MB_MESSAGE Request and Response Messages.....	16
2.2.4.2.1	SMB_COM_SEND_START_MB_MESSAGE Request Message.....	16
2.2.4.2.2	SMB_COM_SEND_START_MB_MESSAGE Response Message.....	17
2.2.4.3	SMB_COM_SEND_TEXT_MB_MESSAGE Request and Response Messages.....	18
2.2.4.3.1	SMB_COM_SEND_TEXT_MB_MESSAGE Request Message.....	18
2.2.4.3.2	SMB_COM_SEND_TEXT_MB_MESSAGE Response Message.....	19
2.2.4.4	SMB_COM_SEND_END_MB_MESSAGE Request and Response Messages.....	19
2.2.4.4.1	SMB_COM_SEND_END_MB_MESSAGE Request Message.....	19
2.2.4.4.2	SMB_COM_SEND_END_MB_MESSAGE Response Message.....	20
<b>3</b>	<b>Protocol Details .....</b>	<b>21</b>
3.1	Name Management Protocol .....	21
3.1.1	Abstract Data Model .....	21
3.1.2	Timers .....	21
3.1.3	Initialization .....	21
3.1.4	Message Processing and Sequencing Rules .....	21
3.1.4.1	NetrMessageNameAdd (Opnum 0) .....	22

3.1.4.2	NetrMessageNameEnum (Opnum 1).....	22
3.1.4.3	NetrMessageNameGetInfo (Opnum 2) .....	23
3.1.4.4	NetrMessageNameDel (Opnum 3) .....	23
3.1.4.5	Sending NetrMessageNameAdd .....	24
3.1.4.6	Receiving NetrMessageNameAdd .....	24
3.1.4.7	Sending NetrMessageNameEnum.....	25
3.1.4.8	Receiving NetrMessageNameEnum.....	25
3.1.4.9	Sending NetrMessageNameGetInfo .....	26
3.1.4.10	Receiving NetrMessageNameGetInfo .....	26
3.1.4.11	Sending NetrMessageNameDel .....	26
3.1.4.12	Receiving NetrMessageNameDel .....	27
3.1.5	Timer Events.....	27
3.1.6	Other Local Events .....	27
3.2	Messaging Protocol.....	27
3.2.1	Abstract Data Model .....	28
3.2.2	Timers .....	28
3.2.3	Initialization .....	28
3.2.4	Message Processing and Sequencing Rules .....	28
3.2.4.1	NetrSendMessage (Opnum 0).....	28
3.2.4.2	Sending NetrSendMessage.....	29
3.2.4.3	Receiving NetrSendMessage.....	29
3.2.4.4	Sending Mailslot Messages or SMB Messages.....	29
3.2.4.5	Receiving Mailslot Messages or SMB Messages.....	30
3.2.5	Timer Events.....	31
3.2.6	Other Local Events .....	31
<b>4</b>	<b>Protocol Examples .....</b>	<b>32</b>
<b>5</b>	<b>Security .....</b>	<b>34</b>
5.1	Security Considerations for Implementers .....	34
5.2	Index of Security Parameters .....	34
<b>6</b>	<b>Appendix A: Full IDL .....</b>	<b>35</b>
6.1	Appendix A.1: msgsvcsend.idl.....	35
6.2	Appendix A.2: msgsvc.idl .....	35
<b>7</b>	<b>Appendix B: Windows Behavior .....</b>	<b>38</b>
<b>8</b>	<b>Index.....</b>	<b>41</b>

# 1 Introduction

This document specifies the Messenger Service Remote Protocol. The Messenger Service Remote Protocol is a set of **remote procedure call (RPC)** interfaces that instructs a **server** (referred to in this document as a "**message server**") to perform one or more of the following tasks:

- Deliver messages to a local or remote message server for display to a console user.
- Manage the names for which the message server receives messages.

The message server does not maintain **client** state information.

## 1.1 Glossary

The following terms are defined in [\[MS-GLOS\]](#):

**Access Control List (ACL)**  
**Client**  
**Endpoint**  
**Local Area Network Adapter (LANA)**  
**Mailslot**  
**Message Server**  
**NetBIOS Name**  
**Opnum**  
**Original Equipment Manufacturer (OEM) Character Set**  
**Remote Procedure Call (RPC)**  
**RPC Dynamic Endpoint**  
**RPC Protocol Sequence**  
**RPC Server**  
**RPC Transport**  
**Server**  
**Unicode**  
**Universally Unique Identifier (UUID)**

The following terms are specific to this document:

**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](mailto: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>

[CODEPG] Microsoft Corporation, "Code Pages", <http://www.microsoft.com/globaldev/reference/cphome.aspx>

If you have any trouble finding [CODEPG], please check [here](#).

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

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

[MS-MAIL] Microsoft Corporation, "[Remote Mailslot Protocol Specification](#)", March 2007.

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

[MS-SMB] Microsoft Corporation, "[Server Message Block \(SMB\) Protocol Specification](#)", July 2007.

[RFC768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980, <http://www.ietf.org/rfc/rfc768.txt>

[RFC1001] Network Working Group, "Protocol Standard for a NetBIOS Service on a TCP/UDP Transport: Concepts and Methods", RFC 1001, March 1987, <http://www.ietf.org/rfc/rfc1001.txt>

[RFC1002] Network Working Group, "Protocol Standard for a NetBIOS Service on a TCP/UDP Transport: Detailed Specifications", RFC 1002, March 1987, <http://www.ietf.org/rfc/rfc1002.txt>

[RFC1035] Mockapetris, R., "Domain Names - Implementation and Specification", RFC 1035, November 1987, <http://www.ietf.org/rfc/rfc1035.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>

### 1.2.2 Informative References

[MS330904] Microsoft Corporation, "Messenger Service Window That Contains an Internet Advertisement Appears", February 2007, <http://support.microsoft.com/kb/330904>

[PIPE] Microsoft Corporation, "Named Pipes", <http://msdn2.microsoft.com/en-us/library/aa365590.aspx>

### 1.3 Protocol Overview (Synopsis)

The Messenger Service Remote Protocol suite is designed to perform the following functions:

- Receive and display short text messages to the console user. (This function is referred to in this document as the "messaging protocol".)
- Manage the names for which a message server receives messages. (This function is referred to in this document as the "name management protocol".)

The name management protocol portion of the Messenger Service Remote Protocol is used to manage the set of names for which the message server accepts messages. The operations in this protocol are very simple, consisting of add, remove, and enumeration methods. The messaging protocol portion of the Messenger Service Remote Protocol actually has several forms and runs over **mailslots** over Server Message Block protocol, as specified in [\[MS-SMB\]](#) and **RPC dynamic endpoints** over User Datagram Protocol (UDP) (as specified in [\[RFC768\]](#)). For how the message client selects the transport that is used for the messaging protocol, see section [3.2](#).

Typically, the Messenger Service Remote Protocol is used to send a short text message from a server, such as a file server or print server, to a client machine; for example, to indicate that a print job has completed or that a file server is shutting down and all of its clients should save their work and disconnect. As such, the roles of client and server are reversed from typical protocols, with the

message server (recipient) of the messages often being the workstation machine, and the message client (sender) being a server-class machine.

## 1.4 Relationship to Other Protocols

The Messenger Service Remote Protocol suite is dependent on RPC (as specified in [\[C706\]](#)), the [Server Message Block \(SMB\) Protocol](#) (as specified in [\[MS-SMB\]](#)), and the mailslot datagram delivery service (as specified in [\[MS-MAIL\]](#)), which are its transports.

The Messenger Service Remote Protocol uses **NetBIOS names** (as specified in [\[RFC1001\]](#) section 14 and [\[RFC1002\]](#) section 4.1) to identify message recipients.

## 1.5 Prerequisites/Preconditions

The messenger service name management protocol is an RPC interface and, as a result, has the prerequisites specified in [\[MS-RPCE\]](#) as being common to RPC interfaces. Both the message client and the message server must have working RPC implementations.

The messenger service messaging protocol also uses the mailslot (as specified in [\[MS-MAIL\]](#)) datagram delivery mechanism and the [SMB Protocol](#) (as specified in [\[MS-SMB\]](#)) for delivering messages to remote machines, and, therefore, it depends on this mailslot delivery mechanism being operational before the messenger service begins operation. For mailslot operational requirements, see [\[MS-MAIL\]](#) section 1.5. For the mailslot delivery mechanism, see [\[MS-SMB\]](#) section 2.2.12.

## 1.6 Applicability Statement

The messenger service name management protocol is suitable only for managing simple NetBIOS names. The messenger service messaging protocol is suitable only for short, human-readable messages that require no security and have no delivery guarantees. [<1>](#)

## 1.7 Versioning and Capability Negotiation

This document covers versioning issues in the following areas:

**Supported Transports:** The Messenger Service Remote Protocol uses RPC over UDP (as specified in [\[MS-RPCE\]](#)), RPC over Named Pipes (for more information, see [\[PIPE\]](#)), SMB (as specified in [\[MS-SMB\]](#)), and mailslots (as specified in [\[MS-MAIL\]](#)) for its transports.

**Protocol Version:** This protocol's RPC interfaces have a version number of 1.0.

**Security and Authentication Methods:** See section [2.1](#).

**Capability Negotiation:** None.

## 1.8 Vendor-Extensible Fields

The Messenger Service Remote Protocol does not include any vendor-extensible fields.

## 1.9 Standards Assignments

There are no standards assignments directly associated with this protocol.

This protocol does depend on RPC and uses the following RPC **UUIDs**:

- 17FDD703-1827-4E34-79D4-24A55C53BB37 (for name management methods)

- 5A7B91F8-FF00-11D0-A9B2-00C04FB6E6FC (for the [NetrSendMessage](#) method)

This protocol does use NetBIOS for message delivery in some cases. If NetBIOS is used on a TCP/IP network, UDP port 138 may be used, and NetBIOS may need to perform other functions such as name resolution on other ports (as specified in [\[RFC1001\]](#) and [\[RFC1002\]](#)) to support this protocol.

When this protocol uses named pipes, the pipe name used is \PIPE\MSGSVCS.

When this protocol uses mailslots for message delivery, the mailslot name used is \\recipient name\MAILSLOT\MESSNGR where recipient name is the NetBIOS name of the intended recipient of the message.



## 2 Messages

The following sections specify how Messenger Service Remote Protocol messages are transported and Messenger Service Remote Protocol message syntax.

### 2.1 Transport

This protocol suite has a variety of transports, the use of which is detailed in the following sections. Implementations MAY use any one of the transports.

#### 2.1.1 RPC Transport

The Messenger Service Remote Protocol MUST use either the RPC over UDP protocol sequence (NCADG\_IP\_UDP), or the RPC over Named Pipes (NCACN\_NP) protocol sequence, as specified in [\[MS-RPCE\]](#), depending on the interface used. When RPC over Named Pipes is used as the **RPC protocol sequence**, the pipe name that MUST be used is \PIPE\MSGSVCSVC. For the NCADG\_IP\_UDP, see section [3.2.4](#). For the NCACN\_NP protocol, see section [3.1.4](#).

This protocol MUST use the following universally unique identifiers (UUIDs):

- 17FDD703-1827-4E34-79D4-24A55C53BB37 (for recipient name management methods)
- 5A7B91F8-FF00-11D0-A9B2-00C04FB6E6FC (for the [NetrSendMessage](#) method)

This protocol MUST use RPC dynamic endpoints for RPC over TCP/IP, as specified in [\[C706\]](#) part 4.

For each recipient name registered with the message server, the message server MUST register the NetBIOS name of the recipient (with NetBIOS name type 0x3, as specified in [\[RFC1001\]](#) and [\[RFC1002\]](#)) on each bound **Local Area Network Adapter (LANA)**.

This protocol allows any user to establish a connection to the **RPC server**. When using named pipes as the **RPC transport**, the protocol uses the underlying RPC protocol to retrieve the identity of the caller that made the method call, as specified in [\[MS-RPCE\]](#). The message server SHOULD use this identity to perform method-specific access checks, as specified in section [3.1.4.5](#). When using UDP as the RPC transport, the protocol does not perform authentication.

#### 2.1.2 Mailslots

This protocol MUST use the mailslot datagram delivery server, as specified in [\[MS-MAIL\]](#). Mailslot messages, specified in sections [3.2.4.4](#) and [3.2.4.5](#), MUST be sent to the following mailslot: \\recipient name\MAILSLOT\MESSNGR.

The recipient name MUST be the NetBIOS name of the intended recipient of the message.

The message server MUST create this mailslot for each recipient name that is registered with the message server before it can receive messages for that recipient.

When using mailslots to transport messages, the protocol does not perform authentication.

#### 2.1.3 SMB

The Messenger Service Remote Protocol MUST use the SMB server, as specified in [\[MS-SMB\]](#). SMB messages are specified in sections [3.2.4.4](#) and [3.2.4.5](#).

SMB messages MUST always be sent to the NetBIOS name of the intended recipient of the message.

When using SMB to transport messages, the protocol does not perform authentication.

## 2.2 Message Syntax

In addition to RPC base types, the following sections use the definition of [DWORD](#), as specified in [\[MS-DTYP\]](#).

### 2.2.1 Constants

#### 2.2.1.1 Windows Error Codes

Constant/value	Description
ERROR_SUCCESS 0x00000000	The operation completed successfully.
ERROR_ACCESS_DENIED 0x00000005	Access is denied.
ERROR_NOT_ENOUGH_MEMORY 0x00000008	Not enough storage is available to process this command.
ERROR_INVALID_PARAMETER 0x00000057	The parameter is incorrect.
ERROR_INVALID_NAME 0x0000007B	The file name, directory name, or volume label syntax is incorrect.
ERROR_INVALID_LEVEL 0x0000007C	The system call level is not correct.
NERR_Success 0x00000000	The operation completed successfully.
NERR_BufTooSmall 0x0000084B	The API return buffer is too small.
NERR_NetworkError 0x00000858	A general network error occurred.
NERR_InternalError 0x0000085C	An internal error occurred.
NERR_NoComputerName 0x000008DE	The computer name could not be added as a message alias. The name may already exist on the network.
NERR_NameNotFound 0x000008E1	The message alias could not be found on the network.
NERR_AlreadyExists 0x000008E4	This message alias already exists locally.
NERR_TooManyNames 0x000008E5	The maximum number of added message aliases has been exceeded.
NERR_DelComputerName 0x000008E6	The computer name could not be deleted.
NERR_GrpMsgProcessor	An error occurred in the domain message processor.

Constant/value	Description
0x000008E8	
NERR_PausedRemote 0x000008E9	The message was sent, but the recipient has paused the Messenger service.
NERR_BadReceive 0x000008EA	The message was sent but not received.
NERR_NameInUse 0x000008EB	The message alias is currently in use. Try again later.
NERR_NotLocalName 0x000008ED	The name is not on the local computer.
NERR_TruncatedBroadcast 0x000008F1	The broadcast message was truncated.
NERR_DuplicateName 0x000008F9	A duplicate message alias exists on the network.
NERR_DeleteLater 0x000008FA	This message alias will be deleted later.
NERR_IncompleteDel 0x000008FB	The message alias was not successfully deleted from all networks.

## 2.2.2 Data Types

### 2.2.2.1 NET\_API\_STATUS

NET\_API\_STATUS is a 32-bit value used as a return code.

```
#define DWORD NET_API_STATUS
```

### 2.2.2.2 MSGSVC\_HANDLE

**MSGSVC\_HANDLE** is a null-terminated string that MUST denote the NetBIOS name (as specified in [\[RFC1001\]](#) section 14 and [\[RFC1002\]](#) section 4.1) or the fully-qualified DNS (as specified in [\[RFC1035\]](#)) name of the remote computer on which the method is to execute. See ServerName parameter in [NetrMessageNameAdd \(Opnum 0\) \(section 3.1.4.1\)](#), [NetrMessageNameEnum \(Opnum 1\) \(section 3.1.4.2\)](#), [NetrMessageNameGetInfo \(Opnum 2\) \(section 3.1.4.3\)](#), and [NetrMessageNameDel \(Opnum 3\) \(section 3.1.4.4\)](#).

This type is declared as follows:

```
typedef [handle] wchar_t* MSGSVC_HANDLE;
```

## 2.2.3 Structures

### 2.2.3.1 MSG\_INFO\_0

**MSG\_INFO\_0** is a data structure that contains a string specifying the recipient name to which a message is to be sent.

```
typedef struct _MSG_INFO_0 {  
    [string] wchar_t* msgi0_name;  
} MSG_INFO_0,  
*PMSG_INFO_0,  
*LPMSG_INFO_0;
```

**msgi0\_name:** Pointer to a **Unicode** UTF-16 string that MUST denote the recipient (NetBIOS name type 0x3, as specified in [RFC1001](#) and [RFC1002](#)) to which a message is to be sent.

### 2.2.3.2 MSG\_INFO\_1

**MSG\_INFO\_1** is a data structure that contains a string specifying the recipient name to which a message is to be sent.

```
typedef struct _MSG_INFO_1 {  
    [string] wchar_t* msgi1_name;  
    DWORD msgi1_forward_flag;  
    [string] wchar_t* msgi1_forward;  
} MSG_INFO_1,  
*PMSG_INFO_1,  
*LPMSG_INFO_1;
```

**msgi1\_name:** Pointer to a Unicode UTF-16 string that MUST denote the recipient (NetBIOS name type 0x3, as specified in [RFC1001](#) and [RFC1002](#)) to which a message is to be sent.

**msgi1\_forward\_flag:** MUST be 0, and ignored on receipt.

**msgi1\_forward:** MUST be NULL, and ignored on receipt.

### 2.2.3.3 MSG\_INFO\_0\_CONTAINER

**MSG\_INFO\_0\_CONTAINER** is a container structure holding one or more [MSG\\_INFO\\_0](#) structures.

```
typedef struct _MSG_INFO_0_CONTAINER {  
    DWORD EntriesRead;  
    [size_is(EntriesRead)] LPMSG_INFO_0 Buffer;  
} MSG_INFO_0_CONTAINER,  
*PMSG_INFO_0_CONTAINER,  
*LPMSG_INFO_0_CONTAINER;
```

**EntriesRead:** A 32-bit value that MUST denote the number of entries in *Buffer*.

**Buffer:** Pointer to a buffer that MUST contain one or more **MSG\_INFO\_0** structures.

#### 2.2.3.4 MSG\_INFO\_1\_CONTAINER

**MSG\_INFO\_1\_CONTAINER** is a container structure holding one or more [MSG\\_INFO\\_1](#) structures.

```
typedef struct _MSG_INFO_1_CONTAINER {
    DWORD EntriesRead;
    [size_is(EntriesRead)] LPMSG_INFO_1 Buffer;
} MSG_INFO_1_CONTAINER,
*PMSG_INFO_1_CONTAINER,
*LPMSG_INFO_1_CONTAINER;
```

**EntriesRead:** A 32-bit value that MUST denote the number of entries in *Buffer*.

**Buffer:** A pointer to a variable-size buffer that MUST contain one or more **MSG\_INFO\_1** structures.

#### 2.2.3.5 MSG\_ENUM\_STRUCT

**MSG\_ENUM\_STRUCT** is a container structure holding either one [MSG\\_INFO\\_0\\_CONTAINER](#) container or one [MSG\\_INFO\\_1\\_CONTAINER](#) container. The structure also has a member to indicate what type of container it contains.

```
typedef struct _MSG_ENUM_STRUCT {
    DWORD Level;
    [switch_is(Level)] union _MSG_ENUM_UNION {
        [case(0)]
            LPMSG_INFO_0_CONTAINER Level0;
        [case(1)]
            LPMSG_INFO_1_CONTAINER Level1;
    } MsgInfo;
} MSG_ENUM_STRUCT,
*PMSG_ENUM_STRUCT,
*LPMSG_ENUM_STRUCT;
```

**Level:** A 32-bit enumerated number that MUST denote the type of structure contained in *MsgInfo*. It must be either 0 or 1.

**MsgInfo:** A pointer to a buffer that MUST contain a union that consists of either an **MSG\_INFO\_0\_CONTAINER** structure or an **MSG\_INFO\_1\_CONTAINER** structure.

**Level0:** If *Level* is 0, *MsgInfo* MUST contain an **MSG\_INFO\_0\_CONTAINER** named *Level0*.

**Level1:** If *Level* is 1, *MsgInfo* MUST contain an **MSG\_INFO\_1\_CONTAINER** named *Level1*.

### 2.2.3.6 MSG\_INFO

**MSG\_INFO** is a data structure that contains either an [MSG\\_INFO\\_0](#) or an [MSG\\_INFO\\_1](#) structure.

```
typedef
[switch_type(DWORD)]
union _MSG_INFO {
    [case(0)]
        LPMSG_INFO_0 MsgInfo0;
    [case(1)]
        LPMSG_INFO_1 MsgInfo1;
} MSG_INFO,
*PMSG_INFO,
*LPMSG_INFO;
```

**MsgInfo0:** A pointer to a variable-size buffer that MUST contain an **MSG\_INFO\_0** data structure.

**MsgInfo1:** A pointer to a variable-size buffer that MUST contain an **MSG\_INFO\_1** data structure.

### 2.2.4 SMB Message Delivery Protocol

Text messages MAY be delivered by SMB to a message server. The SMB messages used for text message delivery are defined in this section.

Each of these SMB messages MUST be preceded by an SMB header, as specified in [\[MS-SMB\]](#) section 2.2.1. These messages MAY be transported by the NetBIOS over UDP transport, the NetBIOS over IPX transport, or the NetBEUI transport, as specified in [\[RFC1001\]](#) and [\[MS-SMB\].<2>](#)

Unless otherwise specified, numerical fields in these messages are in little-endian byte order.

#### 2.2.4.1 SMB\_COM\_SEND\_MESSAGE Request and Response Messages

The following two sections describe how to implement and interpret SMB\_COM\_SEND\_MESSAGE request messages and response messages.

##### 2.2.4.1.1 SMB\_COM\_SEND\_MESSAGE Request Message

The [SMB\\_COM\\_SEND\\_MESSAGE](#) message is used to send an entire text message in which the length of the message is 128 bytes or less.

In the SMB header of these messages, the **Command** field MUST be set to 0xD0, as specified in [\[MS-SMB\]](#) section 2.2.1. In the response message, the header MAY contain a Status code, as specified in [\[MS-SMB\]](#) section 2.2.1. All other fields in the SMB header MUST be set to 0x00. [<3>](#)

The payload of the SMB\_COM\_SEND\_MESSAGE request message is specified as follows:

0	1	2	3	4	5	6	7	8	9	0 <sup>1</sup>	1	2	3	4	5	6	7	8	9	0 <sup>2</sup>	1	2	3	4	5	6	7	8	9	0 <sup>3</sup>	1	
WordCount									ByteCount															BufferFormat1								
OriginatorName (variable)																																
...																																
BufferFormat2									DestinationName (variable)																							
...																																
BufferFormat3									DataLength															Data (variable)								
...																																

**WordCount (1 byte):** An 8-bit value that MUST denote the number of two-byte word values between the WordCount and **ByteCount** values. WordCount MUST be zero for this message.

**ByteCount (2 bytes):** A 16-bit value that MUST denote the total size of all of the fields that follow, in bytes.

**BufferFormat1 (1 byte):** A constant that MUST denote the type of the next parameter. BufferFormat1 MUST be 0x04 in this message, indicating that the next parameter is a null-terminated ASCII string.

**OriginatorName (variable):** A null-terminated ASCII string that MUST denote the name of the sender of the message. OriginatorName MUST NOT be more than 15 characters (bytes) long, exclusive of the trailing null character (with the trailing null character, this field MAY be 16 bytes long).

**BufferFormat2 (1 byte):** An 8-bit value that MUST contain a constant that specifies the type of the next parameter. BufferFormat2 MUST be 0x04 in this message, indicating that the next parameter is a null-terminated ASCII string.

**DestinationName (variable):** A null-terminated ASCII string that MUST denote the name of the intended recipient of the message. DestinationName MUST NOT be more than 15 characters (bytes) long, exclusive of the trailing null character (with the trailing null character, this field MAY be 16 bytes long).

**BufferFormat3 (1 byte):** An 8-bit value that MUST contain a constant that specifies the type of the next parameter. BufferFormat3 MUST be 0x01 in this message, indicating that the next parameter is a length-prefixed buffer of bytes.

**DataLength (2 bytes):** A 16-bit value that MUST specify the length of the Data buffer. This value MUST NOT be greater than 128 (0x0080).

**Data (variable):** A null-terminated ASCII string that MUST contain the text of the message. Before the message is sent, the ASCII characters CR (0x0D) and LF (0x0A) MUST be

converted to the value 0x14. Pairs of these characters (CRLF or LF CR) SHOULD be converted into a single 0x14 character. This buffer MUST NOT be more than 128 bytes in size. <4>

The response message to SMB\_COM\_SEND\_MESSAGE is specified in section 2.2.4.1.2.

#### 2.2.4.1.2 SMB\_COM\_SEND\_MESSAGE Response Message

The payload of the [SMB\\_COM\\_SEND\\_MESSAGE](#) response message is specified as follows:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
WordCount										ByteCount																					

**WordCount (1 byte):** An 8-bit value that MUST be zero for this message.

**ByteCount (2 bytes):** A 16-bit value that MUST be zero for this message.

The request message to SMB\_COM\_SEND\_MESSAGE is specified in section 2.2.4.1.1.

#### 2.2.4.2 SMB\_COM\_SEND\_START\_MB\_MESSAGE Request and Response Messages

The following two sections describe how to implement and interpret SMB\_COM\_SEND\_START\_MB\_MESSAGE request messages and response messages.

##### 2.2.4.2.1 SMB\_COM\_SEND\_START\_MB\_MESSAGE Request Message

The [SMB\\_COM\\_SEND\\_START\\_MB\\_MESSAGE](#) message is used to signal that a new text message is being sent and to carry the strings containing the names of the sender and the intended recipient of the text message.

In the SMB header of this message, the **Command** field MUST be set to 0xD5, as specified in [\[MS-SMB\]](#) section 2.2.1. In the response message, the header MAY contain a Status code, as specified in [\[MS-SMB\]](#) section 2.2.1. All other fields in the SMB header MUST be set to 0x00. <5>

The payload of the SMB\_COM\_SEND\_START\_MB\_MESSAGE request message is specified as follows:



0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
WordCount								ByteCount																BufferFormat1							
OriginatorName (variable)																															
...																															
BufferFormat2								DestinationName (variable)																							
...																															

**WordCount (1 byte):** An 8-bit value that MUST specify the number of two-byte word values between the WordCount and **ByteCount** values. WordCount MUST be zero for this message.

**ByteCount (2 bytes):** A 16-bit value that MUST specify the size of the remainder of the message (not including ByteCount), in bytes.

**BufferFormat1 (1 byte):** An 8-bit value that MUST contain the type of the next parameter. BufferFormat1 MUST be 0x04 in this message, indicating that the next parameter is a null-terminated ASCII string.

**OriginatorName (variable):** A buffer that MUST contain a null-terminated ASCII string denoting the name of the sender of the message. OriginatorName MUST NOT be more than 15 characters (bytes) long, exclusive of the trailing null character (with the trailing null character, this field MAY be 16 bytes long).

**BufferFormat2 (1 byte):** An 8-bit value that MUST contain a constant that specifies the type of the next parameter. BufferFormat2 MUST be 0x04 in this message, indicating that the next parameter is a null-terminated ASCII string.

**DestinationName (variable):** A buffer that MUST contain a null-terminated ASCII string denoting the name of the intended recipient of the message. DestinationName MUST NOT be more than 15 characters (bytes) long (with the trailing null character, this field MAY be 16 bytes long).

The response message to SMB\_COM\_SEND\_START\_MB\_MESSAGE is specified in section [2.2.4.2.2](#).

#### 2.2.4.2.2 SMB\_COM\_SEND\_START\_MB\_MESSAGE Response Message

The payload of the SMB\_COM\_SEND\_START\_MB\_MESSAGE response message is specified as follows:

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
WordCount								MessageGroupId															ByteCount								
...																															

**WordCount (1 byte):** An 8-bit value that MUST be set to one (0x1) for this message.

**MessageGroupId (2 bytes):** A 16-bit value that MUST specify the NetBIOS session number on which this group of messages is to be delivered.

**ByteCount (2 bytes):** A 16-bit value that MUST be zero for this message.

The request message to [SMB\\_COM\\_SEND\\_START\\_MB\\_MESSAGE](#) is specified in section [2.2.4.2.1](#).

### 2.2.4.3 SMB\_COM\_SEND\_TEXT\_MB\_MESSAGE Request and Response Messages

The following two sections describe how to implement and interpret SMB\_COM\_SEND\_TEXT\_MB\_MESSAGE request messages and response messages.

#### 2.2.4.3.1 SMB\_COM\_SEND\_TEXT\_MB\_MESSAGE Request Message

The [SMB\\_COM\\_SEND\\_TEXT\\_MB\\_MESSAGE](#) message is used to transmit a block of text from a text message when the text message is larger than 128 bytes.

In the SMB header of this message, the **Command** field MUST be set to 0xD7, as specified in [\[MS-SMB\]](#) section 2.2.1. In the response message, the header MAY contain a Status code, as specified in [\[MS-SMB\]](#) section 2.2.1. All other fields in the SMB header MUST be set to 0x00. [<6>](#)

The payload of the SMB\_COM\_SEND\_TEXT\_MB\_MESSAGE request message is specified as follows:

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
WordCount									MessageGroupId														ByteCount								
...									BufferFormat							DataLength															
Data (variable)																															
...																															

**WordCount (1 byte):** An 8-bit value that MUST specify the number of two-byte word values between the WordCount and **ByteCount** values. WordCount MUST be one (0x1) for this message.

**MessageGroupId (2 bytes):** A 16-bit value that MUST specify the NetBIOS session number on which this group of messages is to be delivered.

- ByteCount (2 bytes):** A 16-bit value that MUST specify the size of all of the following fields, in bytes.
- BufferFormat (1 byte):** An 8-bit value that MUST contain a constant value that specifies the type of the next parameter. BufferFormat MUST be 0x01 in this message, indicating that the next parameter is a length-prefixed buffer of bytes.
- DataLength (2 bytes):** A 16-bit value that MUST specify the length of the Data buffer. This value MUST NOT be greater than 128.
- Data (variable):** A block of null-terminated ASCII message text. Before the message is sent, the ASCII characters CR (0x0D) and LF (0x0A) MUST be converted to the value 0x14. Pairs of these characters (CRLF or LFCR) MUST be converted into a single 0x14 character. This buffer MUST NOT be more than 128 bytes in size.<7>

The response message to SMB\_COM\_SEND\_TEXT\_MB\_MESSAGE is specified in section [2.2.4.3.2](#).

**2.2.4.3.2 SMB\_COM\_SEND\_TEXT\_MB\_MESSAGE Response Message**

The payload of the [SMB\\_COM\\_SEND\\_TEXT\\_MB\\_MESSAGE](#) response message is specified as follows:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
WordCount										ByteCount																					

- WordCount (1 byte):** An 8-bit value that MUST specify the number of two-byte word values between the **WordCount** and **ByteCount** values. **WordCount** MUST be zero for this message.
- ByteCount (2 bytes):** A 16-bit value that MUST be zero for this message.

The request message to SMB\_COM\_SEND\_TEXT\_MB\_MESSAGE is specified in section [2.2.4.3.1](#).

**2.2.4.4 SMB\_COM\_SEND\_END\_MB\_MESSAGE Request and Response Messages**

The following two sections describe how to implement and interpret SMB\_COM\_SEND\_END\_MB\_MESSAGE request messages and response messages.

**2.2.4.4.1 SMB\_COM\_SEND\_END\_MB\_MESSAGE Request Message**

The [SMB\\_COM\\_SEND\\_END\\_MB\\_MESSAGE](#) message is used to indicate that transmission of a multiblock text message is complete.

In the SMB header of this message, the **Command** field MUST be set to 0xD6, as specified in [\[MS-SMB\]](#) section 2.2.1. In the response message, the header MAY contain a Status code, as specified in [\[MS-SMB\]](#) section 2.2.1. All other fields in the SMB header MUST be set to 0x00.<8>

The payload of the SMB\_COM\_SEND\_END\_MB\_MESSAGE request message is specified as follows:

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1	
WordCount								MessageGroupId															ByteCount									
...																																

**WordCount (1 byte):** An 8-bit value that MUST specify the number of two-byte word values between the **WordCount** and **ByteCount** values. **WordCount** MUST be one (0x1) for this message.

**MessageGroupId (2 bytes):** A 16-bit value that MUST specify the NetBIOS session number on which this group of messages is to be delivered.

**ByteCount (2 bytes):** A 16-bit value that MUST be 0 for this message.

The response message to SMB\_COM\_SEND\_END\_MB\_MESSAGE is specified in section [2.2.4.4.2](#).

#### 2.2.4.4.2 SMB\_COM\_SEND\_END\_MB\_MESSAGE Response Message

The payload of the [SMB\\_COM\\_SEND\\_END\\_MB\\_MESSAGE](#) response message is specified as follows:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
WordCount								ByteCount																							

**WordCount (1 byte):** An 8-bit value that MUST specify the number of two-byte word values between the **WordCount** and **ByteCount** values. **WordCount** MUST be zero for this message.

**ByteCount (2 bytes):** A 16-bit value that MUST be zero for this message.

The request message to SMB\_COM\_SEND\_END\_MB\_MESSAGE is specified in section [2.2.4.4.1](#).

## 3 Protocol Details

As noted in section [1.3](#), there are two protocols that form the Messenger Service Remote Protocol suite. The first, the name management protocol, allows the invoker to control the names to which the message server responds. The second, the messaging protocol, contains the methods by which a message client can send a text message to the message server.

### 3.1 Name Management Protocol

The purpose of the name management protocol of the Messenger Service Remote Protocol suite is to manage the name table.

#### 3.1.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The organization is provided to explain 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 message server maintains a table (in memory) of NetBIOS names for which messages can be received. The message server maintains one such table per local area network adapter (LANA). The names typically include the name of the machine and the names of the users who are currently using the machine. Each name **MUST** be a valid NetBIOS name with a NetBIOS name type code. The message server **MUST** only listen for NetBIOS names of type 0x03. [<9>](#)

#### 3.1.2 Timers

Timers are used to retry some name management operations that might initially fail due to contention issues with a name table. This behavior is specified in section [3.1.4.6](#). Timers are used by RPC to implement resiliency to network outages, as specified in [\[MS-RPCE\]](#). Timers are present in NetBIOS name operations, as specified in [\[RFC1001\]](#) section 15 and [\[RFC1002\]](#) section 4.2.

#### 3.1.3 Initialization

The message server side **MUST** register the **endpoint** (as specified in section [2.1.1](#)) with RPC (as specified in [\[MS-RPCE\]](#)) using default security settings and dynamic endpoints.

#### 3.1.4 Message Processing and Sequencing Rules

Methods in RPC Opnum Order

Method	Description
<a href="#">NetrMessageNameAdd</a>	Opnum: 0
<a href="#">NetrMessageNameEnum</a>	Opnum: 1
<a href="#">NetrMessageNameGetInfo</a>	Opnum: 2
<a href="#">NetrMessageNameDel</a>	Opnum: 3

### 3.1.4.1 NetrMessageNameAdd (Opnum 0)

The **NetrMessageNameAdd (Opnum 0)** interface is used to configure the message server to listen for messages sent to an additional NetBIOS name.

```
NET_API_STATUS NET_API_FUNCTION NetrMessageNameAdd(  
    [in, string, unique] MSGSVC_HANDLE ServerName,  
    [in, string] wchar_t* MsgName  
);
```

**ServerName:** A pointer to a null-terminated string that MUST denote the NetBIOS name (as specified in [\[RFC1001\]](#) section 5.2) or the fully qualified DNS (as specified in [\[RFC1035\]](#)) name of the remote computer on which the function is to execute. There are no other constraints on the format of this string. The message server MUST ignore this parameter.

**MsgName:** A null-terminated Unicode string that MUST denote the recipient name to add. The name is not guaranteed to be unique or reachable by this method. The string MUST be represented using Unicode UTF-16.

**Return Values:** A [NET\\_API\\_STATUS](#) value specified in section [2.2.1.1](#).

### 3.1.4.2 NetrMessageNameEnum (Opnum 1)

The **NetrMessageNameEnum (Opnum 1)** interface is used to enumerate the NetBIOS names for which the message server is currently listening for messages.

```
NET_API_STATUS NET_API_FUNCTION NetrMessageNameEnum(  
    [in, string, unique] MSGSVC_HANDLE ServerName,  
    [in, out] LPMSG_ENUM_STRUCT InfoStruct,  
    [in] DWORD PrefMaxLen,  
    [out] LPDWORD TotalEntries,  
    [in, out, unique] LPDWORD ResumeHandle  
);
```

**ServerName:** A pointer to a null-terminated string that MUST denote the NetBIOS name (as specified in [\[RFC1001\]](#) section 5.2) or the fully qualified DNS (as specified in [\[RFC1035\]](#)) name of the remote computer on which the function is to execute. There are no other constraints on the format of this string. The message server MUST ignore this parameter.

**InfoStruct:** A pointer to a buffer that receives a variable-length data structure of type [MSG\\_ENUM\\_STRUCT](#). The buffer MUST be allocated, and the pointer MUST be assigned by the message server. On return, the structure MUST contain the list of names for which the message server is listening for messages.

**PrefMaxLen:** A 32-bit number that MUST denote the maximum number of bytes the message server should allocate for the buffer. If *PrefMaxLen* is set to 0xFFFFFFFF, the message server MUST always allocate a buffer that can hold all of the information available in a single **MSG\_ENUM\_STRUCT**.

**TotalEntries:** A pointer to a 32-bit number that, on return, MUST contain the total number of entries in *InfoStruct*.

**ResumeHandle:** A pointer to a 32-bit number that MUST contain the ordinal value of the name, in the message server's internal list, on which to start enumeration. This MAY be null.

**Return Values:** A [NET\\_API\\_STATUS](#) value specified in section [2.2.1.1](#).

### 3.1.4.3 NetrMessageNameGetInfo (Opnum 2)

The **NetrMessageNameGetInfo (Opnum 2)** interface is used to retrieve information from the message server on a NetBIOS name for which the message server is currently listening for messages.

```
NET_API_STATUS NET_API_FUNCTION NetrMessageNameGetInfo(  
    [in, string, unique] MSGSVC_HANDLE ServerName,  
    [in, string] wchar_t* MsgName,  
    [in] DWORD Level,  
    [out, switch_is(Level)] LPMSG_INFO InfoStruct  
);
```

**ServerName:** A pointer to a null-terminated string that MUST denote the NetBIOS name (as specified in [RFC1001](#) section 5.2) or the fully qualified DNS (as specified in [RFC1035](#)) name of the remote computer on which the function is to execute. There are no other constraints on the format of this string. The message server MUST ignore this parameter.

**MsgName:** A null-terminated Unicode UTF-16 string. It MUST denote the recipient name for which to get information. The name is not guaranteed to exist.

**Level:** A 32-bit number that MUST be either 0 or 1. It represents the type of structure contained in the InfoStruct [MSG\\_INFO](#) structure. If Level is 0, InfoStruct MUST contain an [MSG\\_INFO\\_0](#) data structure. If Level is 1, InfoStruct MUST contain an [MSG\\_INFO\\_1](#) data structure.

**InfoStruct:** A pointer to a structure of type **MSG\_INFO**.

**Return Values:** A [NET\\_API\\_STATUS](#) value specified in section [2.2.1.1](#).

### 3.1.4.4 NetrMessageNameDel (Opnum 3)

The **NetrMessageNameDel (Opnum 3)** interface is used to configure the message server to stop listening for messages for a particular NetBIOS name.

```
NET_API_STATUS NET_API_FUNCTION NetrMessageNameDel(  
    [in, string, unique] MSGSVC_HANDLE ServerName,  
    [in, string] wchar_t* MsgName  
);
```

**ServerName:** A pointer to a null-terminated string that MUST denote the NetBIOS name (as specified in [RFC1001](#) section 5.2) or the fully qualified DNS (as specified in [RFC1035](#)) name of the remote computer on which the function is to execute. There are no other constraints on the format of this string. The message server MUST ignore this parameter.

**MsgName:** A null-terminated Unicode UTF-16 string that MUST denote the recipient name to delete. It is limited in length to 16 characters. [<10>](#)

**Return Values:** A [NET\\_API\\_STATUS](#) value specified in section [2.2.1.1](#).

### 3.1.4.5 Sending NetrMessageNameAdd

The message client MUST select a message server for the Messenger Service Remote Protocol by means outside the protocol, and MUST set *ServerName* to the NetBIOS name or the DNS name of the message server.

The message client MUST select a recipient name by means outside the protocol, and MUST set *MsgName* to the NetBIOS name of the recipient to add. [<11>](#)

### 3.1.4.6 Receiving NetrMessageNameAdd

On receipt of this message, the message server SHOULD check an internal **access control list (ACL)** to determine whether the message client is authorized to access the name list. If the ACL authorization check is performed, and the message client is not authorized to perform the operation, the message server MUST return ERROR\_ACCESS\_DENIED. [<12>](#)

Next, the message server MUST convert *MsgName* to a valid NetBIOS name, as specified in [\[RFC1001\]](#) section 5.2 and [\[RFC1002\]](#) section 4.1. If *MsgName* is invalid, the message server MUST return ERROR\_INVALID\_NAME. The process for conversion is:

- Server MUST convert *MsgName* to an ASCII string.
- Server MUST truncate *MsgName* to 15 characters if *MsgName* is longer than 15 characters.
- Server MUST pad *MsgName* to 15 characters with the ASCII space character if *MsgName* is shorter than 15 characters.
- Server MUST remove the null terminator for *MsgName*, if present.
- Server MUST append the NetBIOS name type code 0x03 to the converted *MsgName*.

If *MsgName* is valid, the message server MUST check each Local Area Network Adapter (LANA) to see if *MsgName* is currently in the name table on that LANA.

- If *MsgName* is in the name table on any LANA, the message server MUST return NERR\_AlreadyExists.
- If *MsgName* is currently in a delete pending status on any LANA, the message server MUST wait five seconds and check again. At that time, if *MsgName* is still present in the name table on that LANA, the message server MUST return NERR\_AlreadyExists.
- If *MsgName* is not in the name table on any LANA, the message server MUST add *MsgName* to the name table on every LANA.

If the message server is unable to add *MsgName* to the name table on any LANA because the name table is full on that LANA, the message server MUST return NERR\_TooManyNames. [<13>](#)

If the message server fails to add *MsgName* to the name table on any LANA, the message server MUST attempt to delete *MsgName* from the name table on any LANAs where it had successfully added *MsgName*. If it fails to delete *MsgName* from any name tables on any LANA, the message server MUST return NERR\_InternalError.

If the message server successfully added *MsgName* to all LANAs, the message server MUST return ERROR\_SUCCESS.



### 3.1.4.7 Sending NetrMessageNameEnum

The message client MUST select a message server for this protocol by means outside the protocol, and MUST set *ServerName* to the NetBIOS name or the DNS name of the message server.

The message client MUST select the information **Level** that is wanted, either 0 or 1, by means outside the protocol, and MUST set **Level** accordingly.

Fulfilling a [NetrMessageNameEnum](#) request may require multiple calls to this interface. If this is the first call to the interface for a specific request, the message client MUST set *ResumeHandle* to zero. If this is not the first call in a sequence of calls for a specific request, the message client MUST set *ResumeHandle* to the value of *ResumeHandle* returned by the message server in the last call to **NetrMessageNameEnum**.

### 3.1.4.8 Receiving NetrMessageNameEnum

On receipt of this message, the message server SHOULD check an internal access control list (ACL) to determine whether the message client is authorized to access the name list. If the ACL authorization check is performed, and the message client is not authorized to perform the operation, the message server MUST return ERROR\_ACCESS\_DENIED. [<14>](#)

Next, the message server MUST validate that the level passed in *InfoStruct* is either 0 or 1. If the level is any other value, the message server MUST return ERROR\_INVALID\_LEVEL.

If the message client specifies *PrefMaxLen* of 0xFFFFFFFF, the message server MUST attempt to return all names in a single buffer. Otherwise, if the message client specifies *Level* 0, and *PrefMaxLen* is larger than 11,776 bytes, the message server MUST return ERROR\_INVALID\_PARAMETER. [<15>](#)

The message server MUST iterate through its name table in order, starting with the name in ordinal position *ResumeHandle*. If *ResumeHandle* is larger than the number of names in the name table, the message server MUST return NERR\_Success.

For each registered name, the message server MUST add a structure containing the name to a return buffer. The type of structure used MUST be based on the value of *Level*, and MUST be as specified in section [2.2.3.5](#).

If the message server fills its buffer before it has iterated through all registered names, it MUST:

- Return the buffer as *InfoStruct*.
- Set *ResumeHandle* to the ordinal value of the last copied name plus 1.
- Set *TotalEntries* to the number of names in the buffer.
- Return NERR\_BufTooSmall.

If the last registered name is copied into the buffer, the message server MUST:

- Copy the buffer to *InfoStruct*.
- Set *TotalEntries* to the number of names in the buffer.
- Return NERR\_Success.

#### 3.1.4.9 Sending NetrMessageNameGetInfo

The message client MUST select a message server for this protocol by means outside the protocol, and MUST set *ServerName* to the NetBIOS name or the DNS name of the message server.

The message client MUST select a recipient name for this protocol by means outside the protocol, and MUST set *MsgName* to that name. [<16>](#)

The message client MUST select the information *Level* that is wanted, either 0 or 1, by means outside the protocol, and MUST set *Level* accordingly.

#### 3.1.4.10 Receiving NetrMessageNameGetInfo

On receipt of this message, the message server SHOULD check an internal access control list (ACL) to determine whether the message client is authorized to access the name list. If the ACL authorization check is performed, and the message client is not authorized to perform the operation, the message server MUST return ERROR\_ACCESS\_DENIED. [<17>](#)

Before using *MsgName*, the message server MUST convert *MsgName* to a valid NetBIOS name, as specified in [\[RFC1001\]](#) section 5.2 and [\[RFC1002\]](#) section 4.1. If *MsgName* is invalid, the message server MUST return ERROR\_INVALID\_NAME. The process for conversion is:

- Server MUST convert *MsgName* to an ASCII string.
- Server MUST truncate *MsgName* to 15 characters if *MsgName* is longer than 15 characters.
- Server MUST pad *MsgName* to 15 characters with the ASCII space character if *MsgName* is shorter than 15 characters.
- Server MUST remove the null terminator for *MsgName*, if present.
- Server MUST append the NetBIOS name type code 0x03 to the converted *MsgName*.

Then the message server MUST check to see if *MsgName* is in the name table on any LANA. If *MsgName* is not in the name table on any LANA, the message server MUST return NERR\_NotLocalName.

Next, the message server MUST validate that the level passed in *InfoStruct* is either 0 or 1. If *Level* is any other value, the message server MUST return ERROR\_INVALID\_LEVEL.

The message server MUST allocate memory for a structure to return information on the name. If the message server fails to allocate a buffer, the message server MUST return ERROR\_NOT\_ENOUGH\_MEMORY.

The type of structure returned MUST be based on the value of *Level*; fields other than names in the structure MUST be populated as specified in section [2.2.3](#). The message server MUST return all names as Unicode UTF-16 strings.

If the structure is successfully allocated and populated, the message server MUST return NERR\_Success and return a pointer to the structure as *InfoStruct*.

#### 3.1.4.11 Sending NetrMessageNameDel

The message client MUST select a message server for this protocol by means outside the protocol, and MUST set *ServerName* to the NetBIOS name or the DNS name of the message server.

The message client MUST select a recipient name by means outside the protocol, and MUST set *MsgName* to the name of the recipient to delete. [<18>](#)

#### 3.1.4.12 Receiving NetrMessageNameDel

On receipt of this message, the message server SHOULD check an internal access control list (ACL) to determine whether the message client is authorized to access the name list. If the ACL authorization check is performed, and the message client is not authorized to perform the operation, the message server MUST return ERROR\_ACCESS\_DENIED. [<19>](#)

Before using *MsgName*, the message server MUST convert *MsgName* to a valid NetBIOS name, as specified in [\[RFC1001\]](#) section 5.2 and [\[RFC1002\]](#) section 4.1. The process for conversion is:

- Server MUST convert *MsgName* to an ASCII string.
- Server MUST truncate *MsgName* to 15 characters if *MsgName* is longer than 15 characters.
- Server MUST pad *MsgName* to 15 characters with the ASCII space character if *MsgName* is shorter than 15 characters.
- Server MUST remove the null terminator for *MsgName*, if present.
- Server MUST append the NetBIOS name type code 0x03 to the converted *MsgName*.

If conversion of *MsgName* to a NetBIOS name fails, the message server MUST return ERROR\_INVALID\_NAME.

If *MsgName* is the name of the computer, the message server MUST return NERR\_DelComputerName. [<20>](#)

The message server MUST check the name table on each LANA to verify whether *MsgName* is in the name table on that LANA. If *MsgName* is not in the name table on any LANA, the message server MUST return NERR\_NotLocalName.

The message server MUST iterate through each LANA and attempt to delete *MsgName* from the name table for that LANA. If *MsgName* is locked (for example, currently being added) in the name table on any LANA and cannot be deleted, the message server MUST return NERR\_NameInUse.

If *MsgName* is not locked in the name table for any LANA, and the message server attempted to delete *MsgName* from that name table, the message server MUST check the name table after the deletion to see if *MsgName* still exists. If the name still exists, the message server MUST return NERR\_IncompleteDel.

If the deletion is successful, the message server MUST return NERR\_Success.

#### 3.1.5 Timer Events

There are no timer events for this protocol.

#### 3.1.6 Other Local Events

There are no other local events for this protocol.

### 3.2 Messaging Protocol

The messaging protocol is used for the actual transmission of the text messages from the message client to the message server.

### 3.2.1 Abstract Data Model

The messaging protocol relies on the name tables maintained by the name management protocol, as specified in section [3.1.1](#).

### 3.2.2 Timers

There are no timers in the transmission of the text messages. The messages are unreliable when sent by datagram. They are reliable when sent by SMB, as specified in [\[MS-SMB\]](#).

### 3.2.3 Initialization

The message server MUST register the RPC endpoint for messaging, as specified in section [2.1.1](#). Also, the message server MUST register the names from the name table on each LANA as NetBIOS names for mailslot delivery.

The message server SHOULD also register the NetBIOS name of the computer on which it resides as a NetBIOS name of type Messenger (NetBIOS name type 0x03) in the name table on each LANA to which it listens.

### 3.2.4 Message Processing and Sequencing Rules

Methods in RPC Opnum Order

Method	Description
<a href="#">NetrSendMessage</a>	Opnum: 0

#### 3.2.4.1 NetrSendMessage (Opnum 0)

The **NetrSendMessage (Opnum 0)** method is used to send a text message to a message server.

```
NET_API_STATUS NetrSendMessage(  
    [in] handle_t hRpcBinding,  
    [in, string] LPSTR From,  
    [in, string] LPSTR To,  
    [in, string] LPSTR Text  
);
```

**hRpcBinding:** An RPC primitive binding handle, which MUST be as specified in [\[C706-Ch4InterfaceDef\]](#) and [\[C706-Ch5Stubs\].<21>](#)

**From:** A null-terminated string that MUST denote the name of the sender of the message. The name is not guaranteed to be unique or reachable by this method. The string MUST be expressed in the **original equipment manufacturer (OEM) character set**, as specified in [\[CODEPG\]](#), of the invoker of this method.

**To:** A null-terminated string that MUST represent the name of the intended recipient of the message. The name is not guaranteed to be unique or reachable by this method. The string is expressed in the OEM character set, as specified in [\[CODEPG\]](#), of the invoker of this method.

**Text:** A null-terminated string that MUST contain the message that is being sent to the recipient in the *To* parameter. The string is expressed in the OEM character set, as specified in [\[CODEPG\]](#).

**Return Values:** A 32-bit return code.

#### 3.2.4.2 Sending NetrSendMessage

The message client **MUST** select a message server for this protocol by means outside the protocol. Once the name of the message server is selected, the message client **MUST** compose a primitive binding handle, as specified in [\[C706\]](#), for RPC over UDP. [<22>](#)

#### 3.2.4.3 Receiving NetrSendMessage

When the message server receives a [NetrSendMessage](#) message, it **MUST** check the table of names it is maintaining for each LANA to see if the name supplied in the *To* parameter matches one of the names in the table. If there is a match, the message server **SHOULD** display the message in the *Text* parameter. The method of displaying the message is implementation-specific. A message server **MAY** impose security or other policies that control whether the message is displayed, the maximum length of a message, and so on. [<23>](#)

#### 3.2.4.4 Sending Mailslot Messages or SMB Messages

Although this protocol has no methods for constructing or sending mailslot messages or SMB messages, the message server **MUST** be able to receive such messages. Therefore, it is necessary to discuss how such messages are constructed.

The sender **MUST** select a recipient name (called *To*) for the message. *To* **MUST** be a valid NetBIOS name of type 0x03. *To* **MUST** not begin with an asterisk character.

The sender **SHOULD** provide a *From* name field, indicating the sender of the message. If no sender name is available, the sender **SHOULD** use the NetBIOS name of the local computer as the *From* field.

The sender **SHOULD** provide textual content for the message (called *Text*). *Text* **MUST** be represented in the OEM character set, and **MUST** be 652 bytes or less in length.

The sender **MAY** send the message to the mailslot \\recipient name\MAILSLOT\MESSNGR on all LANAs.

The sender **MAY** send the message as a directed SMB on each LANA, as defined below.

If the message text is 128 bytes or less in length, the sender **SHOULD** send the message as an [SMB\\_COM\\_SEND\\_MESSAGE](#) request message. The SMB\_COM\_SEND\_MESSAGE request message **MUST** be constructed as follows:

- *From* **MUST** be a valid NetBIOS name.
- *From* **MUST** be converted to a null-terminated ASCII string and be placed in the **OriginatorName** field.
- *To* **MUST** be a valid NetBIOS name.
- *To* **MUST** be converted to a null-terminated ASCII string and be placed in the **DestinationName** field.
- *Text* **MUST** be placed in the **Data** buffer.

If the message text is more than 128 bytes in length, the sender **SHOULD** break the message text into 128-byte segments. In this case:

- The sender MUST send the first segment as an [SMB\\_COM\\_SEND\\_START\\_MB\\_MESSAGE](#) request message. The SMB\_COM\_SEND\_START\_MB\_MESSAGE request message MUST be constructed as follows:
  - *From* MUST be a valid NetBIOS name.
  - *From* MUST be converted to a null-terminated ASCII string and be placed in the **OriginatorName** field.
  - *To* MUST be a valid NetBIOS name.
  - *To* MUST be converted to a null-terminated ASCII string and be placed in the **DestinationName** field.
- The sender SHOULD wait for the acknowledgment, an SMB\_COM\_SEND\_START\_MB\_MESSAGE response message, before proceeding.
- The sender MUST send each additional segment as an [SMB\\_COM\\_SEND\\_TEXT\\_MB\\_MESSAGE](#) request message. The SMB\_COM\_SEND\_TEXT\_MB\_MESSAGE request message MUST be constructed as follows:
  - The **Data** field of each of these messages MUST contain sequential, contiguous segments of *Text*.
  - Every segment of *Text* except the last SHOULD be exactly 128 bytes in length.
  - The sender SHOULD NOT send additional segments until acknowledgment is received that the previous segment is received in the form of an SMB\_COM\_SEND\_TEXT\_MB\_MESSAGE response message.
- After sending the last segment of *Text*, the sender MUST send an [SMB\\_COM\\_SEND\\_END\\_MB\\_MESSAGE](#) request message. <24>

### 3.2.4.5 Receiving Mailslot Messages or SMB Messages

When the message server receives a mailslot message, it MUST check the table of names it is maintaining for each LANA to see whether the name supplied in the *To* parameter matches one of the names in the table. If there is a match, the message server SHOULD display the message in the *Text* parameter. The method of displaying the message is implementation-specific. A message server MAY impose security or other policies that control whether the message is displayed, the maximum length of a message, and so on. <25>

When the message server receives an SMB message, it MUST validate the name supplied in the *To* parameter against the table of names that it is maintaining for each LANA. If there is a match, the message server SHOULD display the message in the *Text* parameter. The method of displaying the message is implementation-specific. A message server MAY impose security or other policies that control whether the message is displayed, the maximum length of a message, and so on. On receiving and successfully processing any SMB request message, the message server MUST send the corresponding SMB response message back to the message client. <26>

When the message server receives an SMB message, it MUST validate the SMB message, and, if the message is invalid, it MUST return an appropriate SMB error code. <27>

Validation test	SMB error code to return if validation fails
Message size is greater than or equal to the size of a valid SMB header structure.	2
SMB protocol prefix = 0xFF.	3
SMB protocol = S M B.	4
SMB command matches expected function code.	5
SMB has the correct number of parameters for the function performed.	6
SMB header length is less than or equal to the length of the buffer containing the SMB.	7
Variable-length data block fields in the SMB payload are prefixed by \001 or \005.	8
Null-terminated dialect strings in the SMB payload are prefixed by \002.	9
Null-terminated path strings in the SMB payload are prefixed by \003.	10
Null-terminated strings in the SMB payload are prefixed by \004.	11
Total length of SMB message is less than or equal to the buffer used to contain the message.	12

When the message server receives an SMB message, if it is unable to buffer the message, it MUST return SMB\_ERR\_NO\_ROOM to the message client in an appropriate SMB reply message. If it is able to successfully buffer the message, it MUST return SMB\_ERR\_SUCCESS to the message client in an appropriate SMB reply message.

### 3.2.5 Timer Events

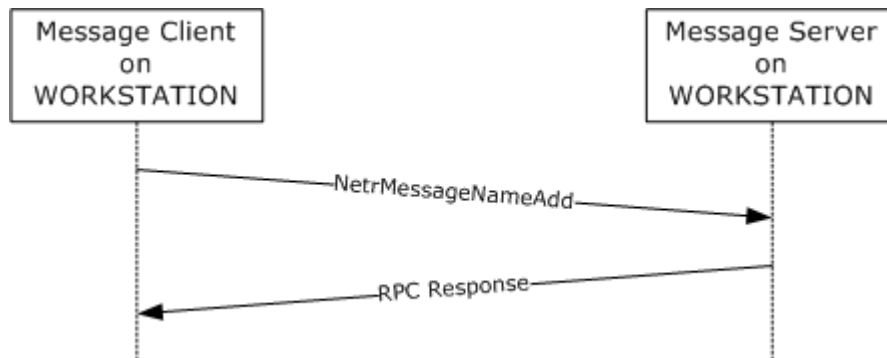
There are no timer events specific to this protocol.

### 3.2.6 Other Local Events

There are no other local events specific to this protocol.

## 4 Protocol Examples

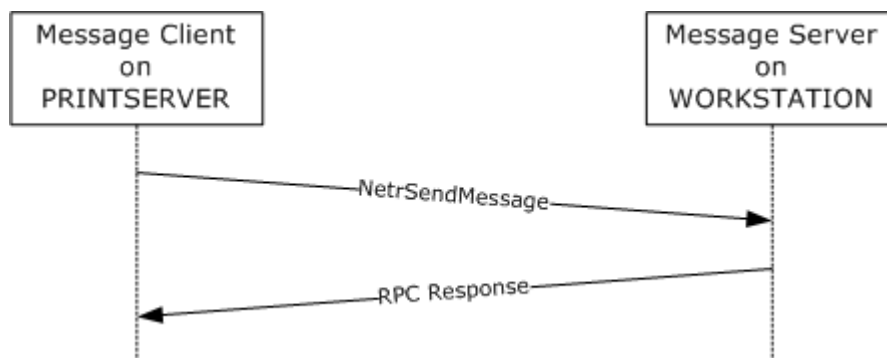
Consider two computers, PRINTSERVER (a print server) and WORKSTATION (a user's desktop). WORKSTATION has only one network interface card, which is on a physical network that is remote to PRINTSERVER, accessible by way of TCP/IP.



**Figure 1: Name management protocol flow diagram**

Example 1. The message server on WORKSTATION, during boot, registers the NetBIOS name "WORKSTATION [03]" with its sole LANA. The component that caused the name to be registered did so using the [NetrMessageNameAdd](#) method of the name management protocol. The flow of this protocol is illustrated in Figure 1.

Example 2. A user "ALICE" logs on to WORKSTATION. The message server on WORKSTATION registers the NetBIOS name "ALICE [03]" with the LANA. The component that caused the name to be registered did so using the **NetrMessageNameAdd** method of the name management protocol. The flow of this protocol is illustrated in Figure 1.



**Figure 2: Message sending protocol diagram**

Example 3. Subsequently, Alice uses some program to print a document. Through means unrelated to this example, the print job is delivered to the print server. On finishing its work, the print server needs to notify Alice. The print server must select among the three protocols available to it for message delivery. Because mailslots and SMB use broadcast mechanisms, and, therefore, are only suitable for machines in the same collision domain, the print server selects the [NetrSendMessage](#) method of the message-sending protocol.

The print server constructs the message, setting *To* to Alice's NetBIOS name, *From* to its own NetBIOS name, and *Text* to the text Print Job Completed. The print server invokes the



**NetrSendMessage** method that delivers the message to the message service on Alice's machine. The message service causes the text of the message to display on the console. The flow of this protocol is illustrated in Figure 2.

## 5 Security

Some interfaces in this protocol do not support remote authentication or authorization, and can potentially be abused to interfere with or cause spurious message delivery.

### 5.1 Security Considerations for Implementers

Because some interfaces of this protocol are unauthenticated and do not perform authorization, a service that receives and acts on messages in this protocol can be used by an attacker to cause unwanted notifications to a console user. This occurred in the Windows implementation of this protocol (for more information, see [\[MS330904\]](#)) and was not due to a code defect in the message server that acted on the protocol messages. It is recommended that this protocol not be implemented due to the lack of security features in the protocol.

### 5.2 Index of Security Parameters

There are no security parameters associated with this protocol.

## 6 Appendix A: Full IDL

The Messenger Service Remote protocol uses two IDL files, msgsvcsend.idl and msgsvc.idl.

### 6.1 Appendix A.1: msgsvcsend.idl

The protocol uses two IDL files, msgsvcsend.idl and msgsvc.idl.

The msgsvcsend.idl file appears as follows:

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

[ uuid (5a7b91f8-ff00-11d0-a9b2-00c04fb6e6fc),
  version(1.0),
  pointer_default(unique)
]
interface msgsvcsend
{
    NET_API_STATUS
    NetrSendMessage(
        [in]          handle_t hRpcBinding,
        [in, string] LPSTR      From,
        [in, string] LPSTR      To,
        [in, string] LPSTR      Text
    );
}
```

### 6.2 Appendix A.2: msgsvc.idl

The protocol uses two IDL files, msgsvcsend.idl and msgsvc.idl.

The msgsvc.idl file appears as follows:

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

#if ( _MSC_VER >= 800) || defined(_STDCALL_SUPPORTED)
    #define NET_API_FUNCTION    __stdcall
#else
    #define NET_API_FUNCTION
#endif

[ uuid(17FDD703-1827-4E34-79D4-24A55C53BB37),
  version(1.0),
  ms_union,
  pointer_default(unique)
]
interface msgsvc
{

    typedef [handle] wchar_t*  MSGSVC_HANDLE;
```

```

typedef struct _MSG_INFO_0 {
    [string] wchar_t* msgi0_name;
}MSG_INFO_0, *PMSG_INFO_0, *LPMSG_INFO_0;

typedef struct _MSG_INFO_1 {
    [string] wchar_t* msgi1_name;
    DWORD msgi1_forward_flag;
    [string] wchar_t* msgi1_forward;
}MSG_INFO_1, *PMSG_INFO_1, *LPMSG_INFO_1;

typedef struct _MSG_INFO_0_CONTAINER {
    DWORD EntriesRead;
    [size_is(EntriesRead)] LPMSG_INFO_0 Buffer;
} MSG_INFO_0_CONTAINER, *PMSG_INFO_0_CONTAINER,
 *LPMSG_INFO_0_CONTAINER;

typedef struct _MSG_INFO_1_CONTAINER {
    DWORD EntriesRead;
    [size_is(EntriesRead)] LPMSG_INFO_1 Buffer;
} MSG_INFO_1_CONTAINER, *PMSG_INFO_1_CONTAINER,
 *LPMSG_INFO_1_CONTAINER;

typedef struct _MSG_ENUM_STRUCT {
    DWORD Level;
    [switch_is(Level)] union _MSG_ENUM_UNION {
        [case(0)] LPMSG_INFO_0_CONTAINER Level0;
        [case(1)] LPMSG_INFO_1_CONTAINER Level1;
    } MsgInfo;
} MSG_ENUM_STRUCT, *PMSG_ENUM_STRUCT, *LPMSG_ENUM_STRUCT;

typedef [switch_type(DWORD)] union _MSG_INFO {
    [case(0)] LPMSG_INFO_0 MsgInfo0;
    [case(1)] LPMSG_INFO_1 MsgInfo1;
} MSG_INFO, *PMSG_INFO, *LPMSG_INFO;

NET_API_STATUS NET_API_FUNCTION
NetrMessageNameAdd (
    [in,string,unique] MSGSVCS_HANDLE ServerName,
    [in,string] wchar_t* MsgName
);

NET_API_STATUS NET_API_FUNCTION
NetrMessageNameEnum (
    [in,string,unique] MSGSVCS_HANDLE ServerName,
    [in,out] LPMSG_ENUM_STRUCT InfoStruct,
    [in] DWORD PrefMaxLen,
    [out] LPDWORD TotalEntries,
    [in,out,unique] LPDWORD ResumeHandle
);

NET_API_STATUS NET_API_FUNCTION
NetrMessageNameGetInfo (
    [in,string,unique] MSGSVCS_HANDLE ServerName,
    [in,string] wchar_t* MsgName,
    [in] DWORD Level,

```

```

        [out, switch_is(Level)] LPMSG_INFO  InfoStruct
    );

NET_API_STATUS NET_API_FUNCTION
NetrMessageNameDel (
    [in,string,unique]  MSGSVC_HANDLE ServerName,
    [in,string]         wchar_t*      MsgName
);
}

```

## 7 Appendix B: Windows Behavior

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

- Windows 2000
- Windows XP
- Windows Server 2003

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

[<1> Section 1.6:](#) The Messenger Service Remote Protocol has been present in Windows operating system products since before the release of Windows NT 4.0. In Windows 2000, the RPC over UDP delivery option was introduced. The protocol specified in this document was present through Windows Server 2003. Support for the protocol was disabled by default (but could be enabled by administrator action) for new installations of Windows Server 2003 SP1 and Windows XP SP2. The protocol was removed from the Windows product line starting with Windows Vista and is not present in Windows Server 2008.

[<2> Section 2.2.4:](#) Windows uses whatever transport is bound to a particular local area network adapter (LANA), and supports all of these transports.

[<3> Section 2.2.4.1.1:](#) Windows returns a 32-bit error code in the **Status** field when an SMB error occurs.

[<4> Section 2.2.4.1.1:](#) Windows always converts ASCII CR, LF, CRLF, and LFCR into a single 0x14 character.

[<5> Section 2.2.4.2.1:](#) Windows returns a 32-bit error code in the Status field when an SMB error occurs.

[<6> Section 2.2.4.3.1:](#) Windows returns a 32-bit error code in the Status field when an SMB error occurs.

[<7> Section 2.2.4.3.1:](#) Windows always converts ASCII CR, LF, CRLF, and LFCR into a single 0x14 character.

[<8> Section 2.2.4.4.1:](#) Windows returns a 32-bit error code in the Status field when an SMB error occurs.

[<9> Section 3.1.1:](#) The Windows implementation supports a table of up to 256 names for each LANA. Windows does not permit the deletion of the computer name from any LANA while the message server is running.

[<10> Section 3.1.4.4:](#) Windows XP and Windows Server 2003 both limit the *MsgName* parameter to 16 characters, including the terminating null character, when calling *NetrMessageNameDel()*. If the *MsgName* is longer than 16 characters, the server returns *ERROR\_INVALID\_PARAMETER*. Windows 2000 does not check the length of the *MsgName* parameter when calling *NetrMessageNameDel()*, but will only read in the first 255 characters.

[<11> Section 3.1.4.5:](#) Windows XP and Windows Server 2003 both limit the *MsgName* parameter to 16 characters, including the terminating null character, when calling [NetrMessageNameAdd\(\)](#). If the *MsgName* is longer than 16 characters, the server returns *ERROR\_INVALID\_PARAMETER*.

Windows 2000 does not check the length of the *MsgName* parameter when calling [NetrMessageNameAdd\(\)](#), but will only read in the first 255 characters.

<12> [Section 3.1.4.6:](#) Windows implementations require that the message client be an administrator or a local user of the machine.

<13> [Section 3.1.4.6:](#) The name table for each LANA on Windows can hold up to 256 entries.

<14> [Section 3.1.4.8:](#) Windows implementations require that the message client be an administrator or a local user of the machine.

<15> [Section 3.1.4.8:](#) If the message client specifies *Level 0*, and *PrefMaxLen* is larger than 11,776 bytes, a Windows 2000 Server message server returns ERROR\_SUCCESS.

<16> [Section 3.1.4.9:](#) Windows XP and Windows Server 2003 both limit the *MsgName* parameter to 16 characters, including the terminating null character, when calling [NetrMessageNameGetInfo\(\)](#). If the *MsgName* is longer than 16 characters, the server returns ERROR\_INVALID\_PARAMETER. Windows 2000 does not check the length of the *MsgName* parameter when calling [NetrMessageNameGetInfo\(\)](#), but will only read in the first 255 characters.

<17> [Section 3.1.4.10:](#) Windows implementations require that the message client be an administrator or a local user of the machine.

<18> [Section 3.1.4.11:](#) Windows XP and Windows Server 2003 both limit the *MsgName* parameter to 16 characters, including the terminating null character, when calling [NetrMessageNameDel\(\)](#). If the *MsgName* is longer than 16 characters, the server returns ERROR\_INVALID\_PARAMETER. Windows 2000 does not check the length of the *MsgName* parameter when calling [NetrMessageNameDel\(\)](#), but will only read in the first 255 characters.

<19> [Section 3.1.4.12:](#) Windows implementations require that the message client be an administrator or a local user of the machine.

<20> [Section 3.1.4.12:](#) If *MsgName* is the name of the computer, a Windows Server 2003 message server returns ERROR\_SUCCESS.

<21> [Section 3.2.4.1:](#) *hRpcBinding* is an interface field, not a wire field, and was implicit rather than explicit in the interface prior to Windows XP. In Windows XP the interface was changed to add the explicit RPC binding handle. The RPC binding handle will never be seen in messages on the network.

<22> [Section 3.2.4.2:](#) Windows-based clients format the name specified in the *To* parameter as the NetBIOS name of the recipient (with NetBIOS name type 0x03) and use that to form the binding handle. For how the name may be truncated or padded, see section [3.1.4.1](#).

<23> [Section 3.2.4.3:](#) When the message server receives a message for a registered recipient name, the message server causes the message to be displayed on the console of the interactive user of the machine. The maximum message length that displays is 4,095 characters.

<24> [Section 3.2.4.4:](#) Windows sends messages of up to 128 bytes using [SMB\\_COM\\_SEND\\_MESSAGE](#) and uses the multiblock SMB sequence for messages larger than 128 bytes. When a message is more than 128 bytes, Windows breaks the messages into 128-byte segments prior to transmission. Where a behavior is defined as SHOULD, Windows implements that behavior.

The Windows implementation of the message client for the protocols defined in this document rejects messages sent to the recipient name \*.

For recipient names that end in an asterisk (called domain names), Windows attempts to deliver the message through mailslots.

For recipient names that do not end in an asterisk, Windows attempts to deliver the message by way of SMB.

If message delivery over SMB is attempted and fails, Windows attempts to deliver the message by way of a call to [NetrSendMessage](#).

[<25> Section 3.2.4.5:](#) When the message server receives a mailslot message for a registered recipient name, the message server causes the message to be displayed on the console of the interactive user of the machine. The maximum message length that displays is 4,095 characters.

[<26> Section 3.2.4.5:](#) When the message server receives an SMB message for a registered recipient name, the message server causes the message to be displayed on the console of the interactive user of the machine. The maximum message length that displays is 4,095 characters.

[<27> Section 3.2.4.5:](#) If the message client is unable to validate the SMB, it returns NERR\_NetworkError.



## 8 Index

### A

Abstract data model  
    [messaging](#)  
    [name management](#)  
[Applicability](#)

### C

[Capability negotiation](#)  
[Constants](#)

### D

Data model - abstract  
    [messaging](#)  
    [name management](#)  
[Data types](#)

### E

[ERROR\\_ACCESS\\_DENIED](#)  
[ERROR\\_INVALID\\_LEVEL](#)  
[ERROR\\_INVALID\\_NAME](#)  
[ERROR\\_INVALID\\_PARAMETER](#)  
[ERROR\\_NOT\\_ENOUGH\\_MEMORY](#)  
[ERROR\\_SUCCESS](#)  
[Examples](#)

### F

[Fields - vendor-extensible](#)  
Full IDL ([section 6](#), [section 6.1](#), [section 6.2](#))

### G

[Glossary](#)

### I

IDL ([section 6](#), [section 6.1](#), [section 6.2](#))  
[Implementer - security considerations](#)  
[Index of security parameters](#)  
[Informative references](#)  
Initialization  
    [messaging](#)  
    [name management](#)  
[Introduction](#)

### L

Local events  
    [messaging](#)  
    [name management](#)  
[LPMSG\\_ENUM\\_STRUCT](#)  
[LPMSG\\_INFO\\_0](#)  
[LPMSG\\_INFO\\_0\\_CONTAINER](#)  
[LPMSG\\_INFO\\_1](#)

[LPMSG\\_INFO\\_1\\_CONTAINER](#)

### M

Mailslots  
    [receiving messages](#)  
    [sending messages](#)  
    [transport](#)  
Messages  
    [overview](#)  
    [syntax](#)  
    [transport](#)  
Messaging  
    [abstract data model](#)  
    [initialization](#)  
    [overview](#)  
    [timer events](#)  
    [timers](#)  
[MSG\\_ENUM\\_STRUCT structure](#)  
[MSG\\_INFO\\_0 structure](#)  
[MSG\\_INFO\\_0\\_CONTAINER structure](#)  
[MSG\\_INFO\\_1 structure](#)  
[MSG\\_INFO\\_1\\_CONTAINER structure](#)

### N

Name management  
    [abstract data model](#)  
    [initialization](#)  
    [local events](#)  
    [overview](#)  
    [timer events](#)  
    [timers](#)  
[NERR\\_AlreadyExists](#)  
[NERR\\_BadReceive](#)  
[NERR\\_BufTooSmall](#)  
[NERR\\_DelComputerName](#)  
[NERR\\_DeleteLater](#)  
[NERR\\_DuplicateName](#)  
[NERR\\_GrpMsgProcessor](#)  
[NERR\\_IncompleteDel](#)  
[NERR\\_InternalError](#)  
[NERR\\_NameInUse](#)  
[NERR\\_NameNotFound](#)  
[NERR\\_NetworkError](#)  
[NERR\\_NoComputerName](#)  
[NERR\\_NotLocalName](#)  
[NERR\\_PausedRemote](#)  
[NERR\\_Success](#)  
[NERR\\_TooManyNames](#)  
[NERR\\_TruncatedBroadcast](#)  
[NET\\_API\\_STATUS](#)  
NetrMessageNameAdd ([section 3.1.4.5](#), [section 3.1.4.6](#))  
[NetrMessageNameAdd method](#)  
NetrMessageNameDel ([section 3.1.4.11](#), [section 3.1.4.12](#))  
[NetrMessageNameDel method](#)

[NetrMessageNameEnum](#) ([section 3.1.4.7](#), [section 3.1.4.8](#))  
[NetrMessageNameEnum method](#)  
[NetrMessageNameGetInfo](#) ([section 3.1.4.9](#), [section 3.1.4.10](#))  
[NetrMessageNameGetInfo method](#)  
[NetrSendMessage](#) ([section 3.2.4.2](#), [section 3.2.4.3](#))  
[NetrSendMessage method](#)  
[Normative references](#)

## O

[Overview \(synopsis\)](#)

## P

[Parameters - security index](#)  
[PMSG\\_ENUM\\_STRUCT](#)  
[PMSG\\_INFO\\_0](#)  
[PMSG\\_INFO\\_0\\_CONTAINER](#)  
[PMSG\\_INFO\\_1](#)  
[PMSG\\_INFO\\_1\\_CONTAINER](#)  
[Preconditions](#)  
[Prerequisites](#)

## R

[References](#)  
[informative](#)  
[normative](#)  
[overview](#)  
[Relationship to other protocols](#)  
[RPC transport](#)

## S

[Security](#)  
[implementer considerations](#)  
[overview](#)  
[parameter index](#)  
[SMB](#)  
[receiving messages](#)  
[sending messages](#)  
[transport](#)  
[SMB message delivery protocol](#) ([section 2.2.4](#), [section 2.2.4.1](#), [section 2.2.4.2](#), [section 2.2.4.3](#), [section 2.2.4.4](#))  
[SMB\\_COM\\_SEND\\_END\\_MB\\_MESSAGE packet](#)  
[SMB\\_COM\\_SEND\\_END\\_MB\\_MESSAGE Response packet](#)  
[SMB\\_COM\\_SEND\\_MESSAGE packet](#)  
[SMB\\_COM\\_SEND\\_MESSAGE Response packet](#)  
[SMB\\_COM\\_SEND\\_START\\_MB\\_MESSAGE packet](#)  
[SMB\\_COM\\_SEND\\_START\\_MB\\_MESSAGE Response packet](#)  
[SMB\\_COM\\_SEND\\_TEXT\\_MB\\_MESSAGE packet](#)  
[SMB\\_COM\\_SEND\\_TEXT\\_MB\\_MESSAGE Response packet](#)  
[Standards assignments](#)  
[Structures](#)  
[Syntax](#)

## T

[Timer events](#)  
[messaging](#)  
[name management](#)  
[Timers](#)  
[messaging](#)  
[name management](#)  
[Transport](#)

## V

[Vendor-extensible fields](#)  
[Versioning](#)

## W

[Windows behavior](#)