[MS-RPCL]:
Remote Procedure Call Location Services Extensions

Revision Summary

Date	Revision History	Revision Class	Comments
10/22/2006	0.01		MCPP Milestone 1 Initial Availability
01/19/2007	1.0		MCPP Milestone 1
03/02/2007	1.1		Monthly release
04/03/2007	1.2		Monthly release
05/11/2007	1.3		Monthly release

Date	Revision History	Revision Class	Comments
07/03/2007	1.3.1	Editorial	Revised and edited the technical content.
07/20/2007	1.3.2	Editorial	Revised and edited the technical content.
08/10/2007	1.3.3	Editorial	Revised and edited the technical content.
09/28/2007	1.3.4	Editorial	Revised and edited the technical content.
10/23/2007	1.3.5	Editorial	Revised and edited the technical content.
11/30/2007	1.3.6	Editorial	Revised and edited the technical content.
01/25/2008	1.3.7	Editorial	Revised and edited the technical content.

Table of Contents

1 Introduction

The Remote Procedure Call (RPC) Location Services Extensions is a set of Microsoft-proprietary extensions/restrictions to the DCE Remote Procedure Call (RPC) Location Services specified in [C706]. These extensions add new capabilities to the DCE RPC Location services protocol.

This document specifies a set of extensions and restrictions to the DCE Remote Procedure Call Location Services specification as specified in [C706].

1.1 Glossary

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

Active Directory (AD)
Binding
Client Locator
Domain
Domain Controller (DC)
Endpoint
Fully Qualified Distinguished Name (FQDN)
Interface Definition Language (IDL)
Locator
Master Locator
Mailslot
Microsoft Interface Definition Language (MIDL)
Name Service Entry
Object UUID
Opnum
Profile Element
Relative Distinguished Name (RDN)
Remote Procedure Call (RPC)
Remote Procedure Call Name Service (RPC Name Service)
RPC Protocol Sequence (Protocol Sequence)
RPC Transfer Syntax (Transfer Syntax)
RPC Transport (Transport)
Server Locator
Universally Unique Identifier (UUID) or Globally Unique Identifier (GUID)
Well-Known Endpoint

The following terms are specific to this document:

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as described in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT. 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 tem MAY implies that Windows does not follow prescription.

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.

[C705] The Open Group, "DCE 1.1: Directory Services", C705, August 1997, http://www.opengroup.org/public/pubs/catalog/c705.htm

[C706] The Open Group, "DCE 1.1: Remote Procedure Call", C706, August 1997, http://www.opengroup.org/public/pubs/catalog/c706.htm

[MS-ADA1] Microsoft Corporation, "Active Directory Schema Attributes A-L", June 2007.

[MS-ADA3] Microsoft Corporation, "Active Directory Schema Attributes N-Z", June 2007.

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

[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.

[NETBEUI] IBM Corporation, "LAN Technical Reference: 802.2 and NetBIOS APIs", 1986, http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/BOOKS/BK8P7001/CCONTENTS

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

[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

[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

[RFC4120] Neuman, C., Yu, T., Hartman, S., and Raeburn, K., "The Kerberos Network Authentication Service (V5)", RFC 4120, July 2005, http://www.ietf.org/rfc/rfc4120.txt

[RFC4178] Zhu, L., Leach, P., Jaganathan, K., and Ingersoll, W., "The Simple and Protected Generic Security Service Application Program Interface (GSS-API) Negotiation Mechanism", RFC 4178, October 2005, http://www.ietf.org/rfc/rfc4178.txt

1.2.2 Informative References

1.3 Protocol Overview (Synopsis)

This specification extends the DCE Remote Procedure Call (RPC) Location Services specification defined in section "Name Service Interface", in Part 2 of [C706]. These extensions add new capabilities to the DCE RPC Location Services Protocol and, in some cases, place additional restrictions upon the DCE RPC Location Services Protocol. This specification adheres to the abstract data model as specified in [C706], Part 2, but an implementation of this specification will not interoperate with an implementation of [C706] Part 2.

This document includes the following:

- An extension to provide RPC Location Service functionality in an environment where a centrally accessible directory service like **AD** is not available.
- An extension defining the implementation of the RPC Location Services specification in an AD environment.
- An extension enabling interoperable RPC Location Service functionality between locators running outside an AD environment, and locators running inside an AD environment.
- An extension of the syntax for **name service entries** to include a **domain** name.
- A restriction requiring profile, group and server attributes to be defined on separate name service entries. These attributes are as specified in section "Name Service Attributes" in [C706].
- A restriction requiring clients to be in joined to a domain role as described in [MS-DSSR] in order to support persistently storing exported name service entries.
- A restriction requiring clients to be members of an AD domain to support persistently storing exported name service entries.
- A restriction requiring clients to be members of an AD domain to support profile and group attributes.

1.3.1 Roles

A **locator** conceptually operates in the following three roles. Over the course of a given **protocol sequence**, a given locator may simultaneously occupy more than one of these roles.

1. **Server Locator**: A locator running on a computer on which a given name service entry is exported.
2. **Client Locator**: A locator running on a computer on which a given name service entry is looked up.
3. **Master Locator**: A locator that facilitates communication between client locators and server locators.

1.3.2 Modes

A locator MUST run in exactly one of the following modes.

Non-Directory Mode: In this mode, a locator supports lookup and export of server entries without support for persistently stored name service entries and, therefore, it does not rely on an AD store. Functionalities related to profile and group entries are not supported.

Directory Mode: In this mode, a locator supports persistently storing all name service entries by relying on an AD store. A locator in this mode MUST be running on a computer joined to an AD domain. Locators in this mode interoperate with locators in non-directory mode in the following ways.

- Client locators in this mode support lookups of server entries from server locators running in non-directory mode.
- Server locators in this mode support lookups of server entries from client locators running in non-directory mode.

Directory-Only Mode: In this mode, a locator supports persistently storing all name service entries by relying on an AD store. A locator in this mode MUST be running on a computer joined to an AD domain. Locators in this mode do not interoperate with locators running in non-directory mode.

In this mode, a locator MUST NOT do the following:

1. Respond or listen to **mailslot** requests.
2. Initiate any mailslot requests.
3. Forward a lookup request that originated locally to the master locator.

1.3.3 Name Service Entries in AD

In AD domain environments, this specification persistently stores **RPC name service** entries in the AD store. The following schema elements are used to implement persistent storing, as specified in section 2.2.5.

Schema class	Description
rpcServer	Represents a server entry. This instance MUST contain the **object UUIDs** exported to the server entry. Interfaces exported by the server MUST be represented as child elements of this instance of type rpcServerElement.
rpcGroup	Represents a group entry. This instance MUST contain group members of the Group Entry.
rpcProfile	Represents a profile entry. **Profile elements** in this profile entry MUST be represented as child elements of this instance of type rpcProfileElement.
rpcServerElement	Represents a single interface exported to the server entry represented by the parent container.
rpcProfileElement	Represents a profile element exported to the profile entry represented by the parent container. An entry with an interface identifier specification of null **GUID** represents the default profile element of the profile entry.

The following diagram shows the layout of the RPC name service entries in the AD store.

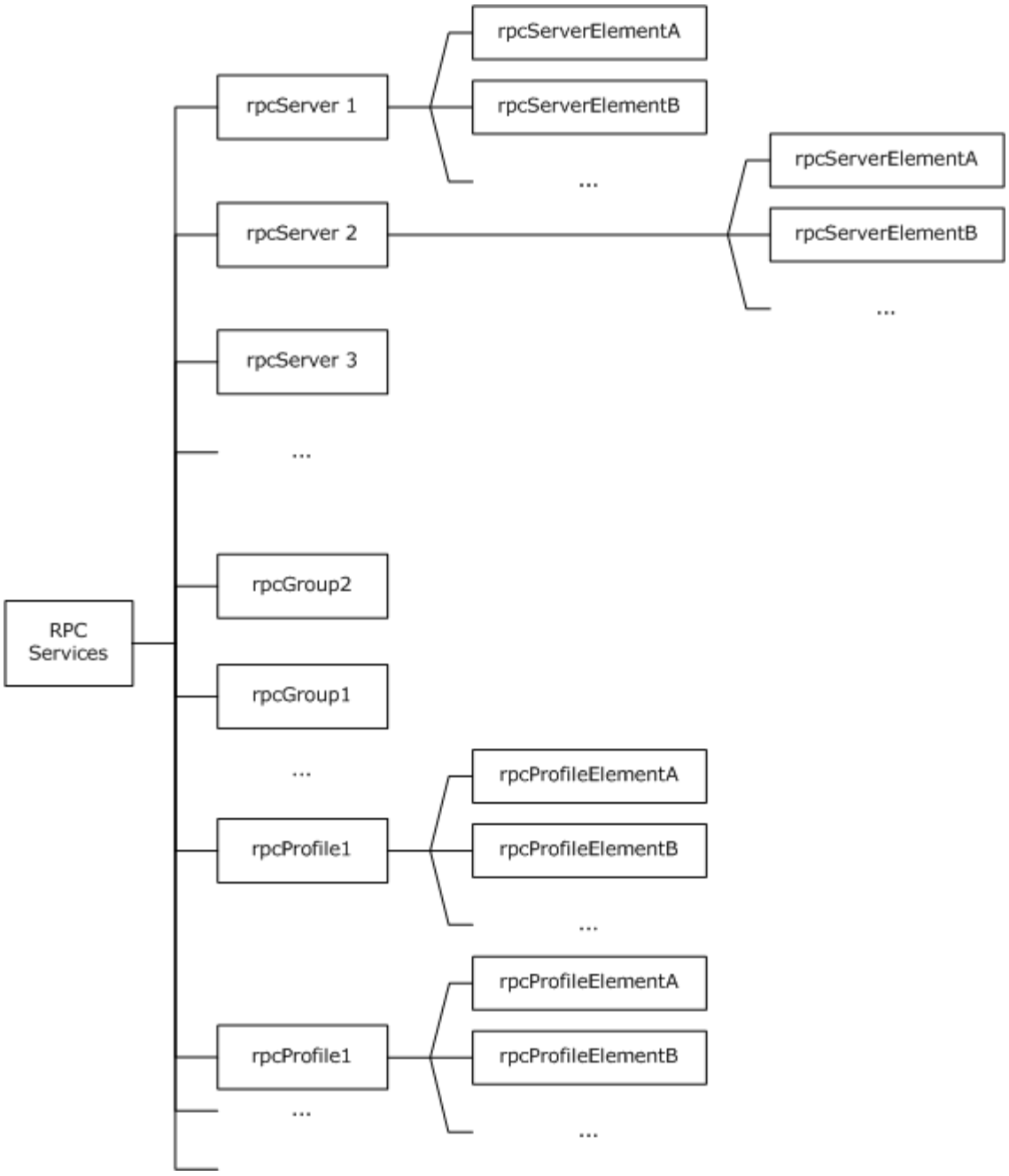


Figure 1: AD layout

1.4 Relationship to Other Protocols

The Remote Procedure Call Location Services Extensions depend on the following protocols.

1. Locators depend on the **DC** and domain discovery mechanism [MS-ADTS] to obtain information about domains and to select their mode of operation, as specified in section 3.1.3.1.

2. Client locators depend on Remote Procedure Call Protocol Extensions [MS-RPCE] for forwarding requests to the master locator, as specified in section 3.3.1.4.

3. Master locators depend on the Remote Mailslot Protocol [MS-MAIL] to broadcast requests for the queries they receive, as specified in section 3.4.1.5.1. Server locators depend on the Remote Mailslot Protocol for responding to broadcast requests, as specified in section 3.2.1.5.

4. Client locators depend on the Remote Mailslot Protocol to dynamically discover the master locator, as specified in sections 3.3.1.4.3 and 3.4.1.5.2.

5. Server locators depend on LDAP [MS-ADTS] for persistently storing name service entries in the AD store, as specified in section 3.2.2.4.

6. Client locators depend on LDAP [MS-ADTS] to look up persistently stored entries in AD for the name service entries, as specified in section 3.2.2.4.

No other protocols have a dependency on this protocol.

1.5 Prerequisites/Preconditions

Elements of these protocol extensions incorporate **RPC** interfaces and, as a result, inherit the prerequisites identified in [MS-RPCE] that are common to RPC interfaces.

A locator MUST be able to determine its role and mode by discovering the following:

Whether it is operating on a computer joined to an AD domain.

Whether it is running on the domain controller for a domain. A locator running on a domain controller runs as a master locator.

When operating as a member of an AD domain, a locator MUST be able to access the AD during initialization to support AD supported functionalities as specified in section 1.3.2.

1.6 Applicability Statement

The RPC Location Services Extensions do not restrict the applicability of [C706]; rather, they extend its applicability to environments where computers have no centrally accessible directory service.

1.7 Versioning and Capability Negotiation

Supported Transports

The client locator communicates with the master locator using the RPC over Server Message Block (SMB) Protocol sequence (ncacn_np). For more information, see section 2.1.

The client locator uses the Remote Mailslot Protocol [MS-MAIL] to discover master locators. For more information, see section 3.3.1.4.3.

The master locator uses the Remote Mailslot Protocol [MS-MAIL] to broadcast requests to client locators and receive their responses. For more information, see section 3.4.1.5.

When operating on a domain-joined computer, the locator uses the LDAP protocol to issue queries and updates to AD in its domain. For more information, see sections 3.2.2.4 and 3.3.2.4.

Capability Negotiation<1>

Protocol Version: This protocol's RPC interface has a single version number of 1.0. The RPC versioning and capability negotiation in this situation is as specified in [C706] and in [MS-RPCE] section 1.7.

Security and Authentication Methods

RPC Interfaces: The RPC interfaces defined by these extensions use the default security settings for RPC over SMB and do not register any additional security providers ([MS-RPCE] section 3.3.3.3). Default security is used for the RPC interfaces of these extensions. More information on security used by the RPC is specified in [MS-RPCE].

LDAP: When **binding** through LDAP, the [GSS]-SPNEGO profile for SASL is selected. The [GSS]-SPNEGO profile uses an implementation specified in [RFC4178] and will result in an actual security mechanism being selected. Typically, this mechanism is Kerberos [RFC4120] but others are possible. If the [GSS]-Kerberos profile is selected, then Kerberos is used. If Kerberos is used, the name passed in for authentication is "LDAP/hostname-of-ldap-server." More information on LDAP, see [MS-ADTS].

1.8 Vendor-Extensible Fields

This protocol does not define any vendor-extensible fields.

1.9 Standards Assignments

Parameter	Value	Reference
LocToLoc RPC Interface UUID	UUID: e33c0cc4-0482-101a-bc0c-02608c6ba218	As specified in 2.1
LocToLoc RPC Interface End Point	Pipe Name: \pipe\Locator	As specified in 2.1
Master Locator Discovery Request Mailslot	\Mailslot\Resp_s	As specified in 2.1
Master Locator Discovery Response Mailslot	\Mailslot\Resp_c	As specified in 2.1
Broadcast Lookup Request Mailslot	\Mailslot\RpcLoc_s	As specified in 2.1
Broadcast Lookup Response Mailslot	\Mailslot\RpcLoc_c	As specified in 2.1

2 Messages

The following sections specify how the RPC Location Services Extensions messages are encapsulated on the wire, and common data types.

2.1 Transport

Lookup Forwarding: The client locators forward lookup requests to master locators over the LocToLoc RPC interface. The RPC interface uses the RPC over SMB protocol sequence, as specified in [MS-RPCE].

This protocol uses the following **well-known endpoint**.

- \pipe\Locator

This **endpoint** is a pipe name for RPC over SMB, as specified in [MS-RPCE].

This protocol MUST use the UUID specified in section 1.9. The RPC Version number is 1.0.

Broadcast Lookup: Master locators broadcast requests for Server Entries by using the Remote Mailslot Protocol with the following destination and address.

- Destination may be either all reachable computers on the network or all computers in a domain
- Address: String literal "\Mailslot\RpcLoc_s"

Server locators respond to broadcast requests from master locators by using the Remote Mailslot Protocol with the following destination and address.

- Mailslot Destination: The machine that sent the broadcast request that this is a response to.
- Address: String literal "\Mailslot\RpcLoc_c"

Master Locator Discovery: Client locators discover the master locator by using the Remote Mailslot Protocol with the following destination and address.

- Mailslot Destination: All reachable computers on the network
- Address: String literal "\Mailslot\Resp_s"

Master locators respond to discovery requests by using the Remote Mailslot Protocol with the following destination and address.

- Mailslot Destination: The machine that sent the broadcast request that this is a response to.
- Address: String literal "\Mailslot\Resp_c"

AD Lookup: A locator on a domain-joined machine uses LDAP to determine whether AD is accessible and to read and write data from AD in the computer's domain. For more information, see section 3.1.3.1. For more information on LDAP, see [MS-ADTS].

2.2 Common Data Types

This section specifies common data types.

2.2.1 Constants

Return value(code)	Description
RPC_C_NS_SYNTAX_DCE(3)	Specifies the syntax of the entry nName, as specified in section 2.2.2.
NSI_S_OK(0)	Used to indicate that the LocToLoc method call executed successfully.

2.2.2 Extensions to the Name Service Entry Name Syntax

All name service entries MUST be identified by an entry name. The syntax of the entry name is specified by the constant RPC_C_NS_SYNTAX_DCE defined in the preceding section. This syntax allows specification of a domain name, which is an extension of the syntax specified in section "DCE Name Syntax" in [C705] part 1.

An entry name is a case-insensitive, null-terminated Unicode [UNICODE] string. The entry name MUST be less than 256 characters. Entry names used in the LocToLoc RPC methods are further restricted so that the maximum length of the entry name (including the terminating NULL character) MUST be less than or equal to 100 characters. Entry names MUST be in one of the following forms.

Local Specification:

/.:/name

Domain Specification:

/.../domainname/name

name: Specifies an identifier for the entry. This field MAY contain a slash (/) character. When operating in directory or directory-only mode, this field MUST NOT contain any characters that are disallowed in the Relative Distinguished Name (**RDN**) of an AD object, as specified in [MS-ADTS]. This is a restriction on the syntax specified in section "DCE Name Syntax" in [C705] part 1.

domainname: Specifies the name of the domain. This field MUST NOT contain the delimiting slash (/) character.

2.2.3 LocToLoc RPC Interface Types

This RPC interface defines data types in addition to the RPC base types and definitions specified in [C706] and [MS-RPCE].

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

Data Type
STRING_T
NSI_UUID_P_T
NSI_UUID_VECTOR_T
NSI_UUID_VECTOR_P_T
NSI_NS_HANDLE_T
NSI_STRING_BINDING_T

Data Type
NSI_BINDING_T
NSI_BINDING_VECTOR_T
NSI_BINDING_VECTOR_P_T

2.2.3.1 STRING_T

The **STRING_T** type defines a string of Unicode [UNICODE] characters.

This type is declared as follows:

```
typedef [string, unique] wchar_t* STRING_T;
```

2.2.3.2 NSI_UUID_P_T

The **NSI_UUID_P_T** type defines a pointer to a GUID structure.

This type is declared as follows:

```
typedef [unique] GUID* NSI_UUID_P_T;
```

2.2.3.3 NSI_UUID_VECTOR_T

The **NSI_UUID_VECTOR_T** type defines an array of **NSI_UUID_P_T** structures.

```
typedef struct _NSI_UUID_VECTOR_T {
  unsigned long count;
  [size_is(count)] NSI_UUID_P_T uuid[*];
} NSI_UUID_VECTOR_T;
```

count: MUST specify the number of **NSI_UUID_P_T** elements in the **uuid** member.

uuid: An array of **NSI_UUID_P_T** entries.

2.2.3.4 NSI_UUID_VECTOR_P_T

The **NSI_UUID_VECTOR_P_T** type defines a pointer to the **NSI_UUID_VECTOR_T** structure.

This type is declared as follows:

```
typedef [unique] NSI_UUID_VECTOR_T* NSI_UUID_VECTOR_P_T;
```

2.2.3.5 NSI_NS_HANDLE_T

The **NSI_NS_HANDLE_T** type defines an opaque pointer that is used to represent a context handle, as specified in [C706] and [MS-RPCE]. It is returned from the server to the client.

This type is declared as follows:

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

2.2.3.6 NSI_STRING_BINDING_T

The **NSI_STRING_BINDING_T** type defines a [UNICODE] string that is used to represent binding information, and which MAY optionally contain endpoint information.<2>

This type is declared as follows:

```
typedef [string] wchar_t* NSI_STRING_BINDING_T;
```

2.2.3.7 NSI_BINDING_T

The **NSI_BINDING_T** type defines an association of a binding with a server entry.

```
typedef struct _NSI_BINDING_T {
  NSI_STRING_BINDING_T string;
  unsigned long entry_name_syntax;
  STRING_T entry_name;
} NSI_BINDING_T;
```

string: A [UNICODE] string that contains a string Binding. For more information, see section "String Bindings" in [C706] Part 2.

entry_name_syntax: An unsigned 32-bit integer specifying the syntax of the entry_name field. This value MUST be RPC_C_NS_SYNTAX_DCE.

entry_name: A [UNICODE] string specifying the entry name of the name service entry, using the syntax identified by the entry_name_syntax parameter as specified in section 2.2.2.

2.2.3.8 NSI_BINDING_VECTOR_T

The NSI_BINDING_VECTOR_T type is defined to hold an array of binding information entries.

```
typedef struct _NSI_BINDING_VECTOR_T {
  unsigned long count;
  [size_is(count)] NSI_BINDING_T binding[*];
} NSI_BINDING_VECTOR_T;
```

count: MUST specify the number of **NSI_BINDING_T** elements in the binding array.

binding: An array of binding information entries.

2.2.3.9 NSI_BINDING_VECTOR_P_T

The **NSI_BINDING_VECTOR_P_T** type defines a pointer to a BINDING_VECTOR_T structure.

This type is declared as follows:

```
typedef [unique] NSI_BINDING_VECTOR_T* NSI_BINDING_VECTOR_P_T;
```

2.2.4 Mailslot Structures

This section specifies structures sent and received by using the Remote Mailslot Protocol for the following operations.

- Broadcast Lookup (section 2.2.4.2)
- Master Locator Discovery (section 2.2.4.3)

2.2.4.1 Common Details

This section specifies the syntax for attributes common to the definitions of several objects in this protocol.

2.2.4.1.1 Mailslot Sender

Mailslot requests and responses, as specified in sections 2.2.4.2.1, 2.2.4.3.1, and 2.2.4.3.2, include information about the sender.

The sender information MUST be a null-terminated string of the following form.

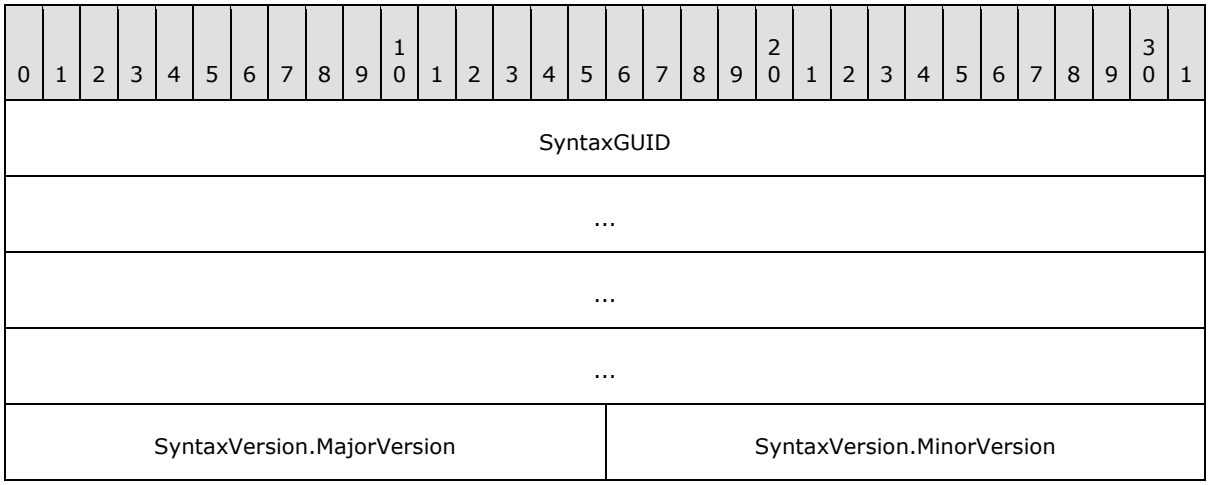
```
SenderName = \\ComputerName
```

ComputerName MUST be the NetBIOS name of the computer where the mailslot originated. For more information on NetBIOS, see [NETBEUI], [RFC1001], and [RFC1002].

2.2.4.1.2 RPC_SYNTAX_IDENTIFIER

This structure MUST contain a GUID and version information ([MS-RPCE] section 2.2.2.7) and is identical to the RPC_SYNTAX_IDENTIFIER structure used in the LocToLoc interface in section 3.1.4. This structure is used to represent the following:

- Identifier and version of an interface.
- Identifier and version of **transfer syntax** for an interface.



SyntaxGUID (16 bytes): As specified in [MS-RPCE] section 2.2.2.7.

SyntaxVersion.MajorVersion (2 bytes): As specified in [MS-RPCE] section 2.2.2.7.

SyntaxVersion.MinorVersion (2 bytes): As specified in [MS-RPCE] section 2.2.2.7.

2.2.4.2 Broadcast Lookup

A master locator broadcasts a request for information by using the Remote Mailslot Protocol when it receives a query as specified in section 3.4.1.5.1.

- This query MUST be sent over the mailslot by using the QueryPacket structure specified in section 2.2.4.2.1
- Server locators MUST respond to the request with the QueryReply structure specified in section 2.2.4.2.2.

2.2.4.2.1 QueryPacket

The **QueryPacket** structure defines the format of the messages sent by the master locator.

```
typedef struct {
  RPC_SYNTAX_IDENTIFIER Interface;
  GUID Object;
  WCHAR WkstaName[20];
  WCHAR EntryName[100];
} QueryPacket;
```

Interface: Optionally MUST specify the identifier and version for the interface being queried. MUST be filled with zeros to indicate that no interface identifier is specified. The type of the structure is specified in section 2.2.4.1.2.

Object: Optionally MUST specify the UUID for the object being queried. MUST be filled with zeros to indicate that no object UUID is specified.

WkstaName: MUST be a Mailslot sender as specified in section 2.2.4.1.1. This parameter is limited to 20 characters including the null terminator.

EntryName: MUST specify the name service entry being looked for. This parameter MUST conform to the RPC_C_NS_SYNTAX_DCE syntax as specified in section 2.2.2. MUST be filled with all zeros to indicate that no name service entry is specified.

2.2.4.2.2 QueryReply

The **QueryReply** structure defines the response of a server locator to a master locator Broadcast Lookup Query.

```
typedef struct {
  WCHAR Domain[20];
  char Buffer[1000];
} QueryReply;
```

Domain: MUST be a null-terminated, fixed-length buffer that MUST contain the NetBIOS domain name of the computer on which the server locator is running. Information on NetBIOS is specified in [NETBEUI], [RFC1001], and [RFC1002].

Buffer: A buffer, which MAY have a maximum of 1000 bytes, which MUST contain the response from the server locator. The buffer MUST be an array of Reply Buffer structures terminated by 4 zero-initialized bytes. Reply buffers are specified in section 2.2.4.2.2.2.

2.2.4.2.2.1 MAILSLOT_ENTRY_TYPE

The **MAILSLOT_ENTRY_TYPE** enumeration defines the type of response being sent as a response to the master locator request.

```
typedef  enum
{
  MailslotServerEntryType = 1
} MAILSLOT_ENTRY_TYPE;
```

MailslotServerEntryType: Server entry is contained in this response.

2.2.4.2.2.2 ReplyBuffer

The **ReplyBuffer** structure specifies the layout of the response in the QueryReply structure.

```
typedef struct {
  fixed_part_of_reply fpr;
  wchar_t entryName[fpr.EntryNameLength];
  long objListSize;
  DWORD unused;
  GUID objUUID[objListSize];
  wchar_t binding[fpr.BindingLength];
} ReplyBuffer;
```

fpr: MUST contain the fixed part of the reply. For more information, see section 2.2.4.2.2.3.

entryName: A null-terminated **Unicode** buffer that MUST contain the name of the name service entry as specified by the RPC_C_NS_SYNTAX_DCE syntax. The size (in characters) of this buffer, including the terminating null character, MUST be fpr.EntryNameLength.

objListSize: MUST contain the number of object UUIDs present in the objUUID array.

unused: Senders MUST set this to zero and receivers MUST ignore it.

objUUID: An array of object UUIDs exported on the name service entry. The number of object UUIDs in this buffer MUST be equal to the objListSize. The size of this buffer MUST be the number of object UUIDs in this buffer.

binding: A null-terminated Unicode buffer, that MUST contain a string binding exported to the name service entry. The size (in characters) of this buffer, including the terminating null character, MUST be fpr.BindingLength.

2.2.4.2.2.3 fixed_part_of_reply

The **fixed_part_of_reply** structure defines the layout of the **Buffer** field in the QueryReply structure that forms the server locator's response to the master locator's query.

```
typedef struct {
  MAILSLOT_ENTRY_TYPE type;
  DWORD unused1[5];
  unsigned long unused2;
  unsigned long unused3;
  RPC_SYNTAX_IDENTIFIER Interface;
  RPC_SYNTAX_IDENTIFIER XferSyntax;
  unsigned long BindingLength;
  DWORD unused4;
  unsigned long EntryNameLength;
  DWORD unused5;
} fixed_part_of_reply;
```

type: MUST specify the type of response. This MUST contain MailslotServerEntryType as specified in section 2.2.4.2.2.1.

unused1: Senders MUST set this to zero and receivers MUST ignore it.

unused2: Senders MUST set this to zero and receivers MUST ignore it.

unused3: Senders MUST set this to zero and receivers MUST ignore it.

Interface: Specifies the interface being returned. The structure MUST be as specified in section 2.2.4.1.2.

XferSyntax: Specifies the transfer syntax for the interface being returned. The structure is specified in section 2.2.4.1.2.

BindingLength: Specifies the number of characters (including the terminating null) in the string binding that appears in the binding field of the Reply Buffer structure that contains this fixed_part_of_reply structure.

unused4: Senders MUST set this to zero and receivers MUST ignore it.

EntryNameLength: MUST specify the number of characters (including the terminating null) in the entry name that appears in the **entryName** field of the Reply Buffer structure that contains this fixed_part_of_reply structure.

unused5: Senders MUST set this to zero and receivers MUST ignore it.

2.2.4.3 Master Locator Discovery

Client locators broadcast requests to find master locators, as specified in section 3.3.1.4.3.

- These requests are sent over mailslot by using the QUERYLOCATOR structure.
- Master locators respond to the request over mailslot by using the QUERYLOCATORREPLY structure.

2.2.4.3.1 QUERYLOCATOR

The **QUERYLOCATOR** structure defines the structure that is sent by using the Remote Mailslot Protocol when the client locator is looking for a master locator.

```
typedef struct {
  unsigned long MessageType;
  unsigned long SenderOsType;
  wchar_t RequesterName[18];
} QUERYLOCATOR;
```

MessageType: This defines the type of the message being sent. It MUST be the following value.

Value	Meaning
QUERY_MASTER_LOCATOR 0x01	Query for an existing master locator.

SenderOsType: An identifier indicating the type of operating system running on the computer of the sender locator. This MUST be the following value.

Value	Meaning
OS_NTWKGRP 0x04	The operating system is Windows NT 4.0or later.

RequesterName: The mailslot sender as specified in section 2.2.4.1.1. This parameter is limited to 18 characters including the null terminator.

2.2.4.3.2 QUERYLOCATORREPLY

The **QUERYLOCATORREPLY** structure represents the data that is sent back by a master locator in response to a master locator discovery request.

```
typedef struct {
  unsigned long unused;
  unsigned long Hint;
  unsigned long Uptime;
  unsigned short SenderName[18];
} QUERYLOCATORREPLY;
```

unused: Clients MUST set this to zero and servers MUST ignore it.

Hint: A hint representing the type of responding locator. It MUST be the following value.

Value	Meaning
REPLY_MASTER_LOCATOR 0x01	This locator is a master locator.

Uptime: SHOULD contain the number of elapsed seconds since the sending computer started up.<3>

SenderName: MUST contain the mailslot sender as specified in section 2.2.4.1.1. This parameter is limited to 18 characters including the null terminator.

2.2.5 Active Directory Schema Specifications

The following sections specify the schemas of objects relevant to this protocol. For more details, see [MS-ADTS], [MS-ADA1], and [MS-ADA3].

2.2.5.1 Common Details

This section specifies the syntax for attributes common to the definitions of several objects.

2.2.5.1.1 Name Service Entry RDN

The relative distinguished name (RDN) attribute of an object specifies the identifier for the object relative to its AD path. For a name service entry, this attribute MUST be identical to the name component of the object's corresponding name service entry name as specified by using RPC_C_NS_SYNTAX_DCE (section 2.2.2).<4>

2.2.5.1.2 Reference Attributes

The Reference Attribute specifies a reference to a name service entry in AD. The value of the attribute MUST be a modified LDAP URL for an object in AD that represents the referenced name service entry. This attribute MUST be identical to the object's LDAP URL without the URL scheme ("ldap:"). For more information on LDAP URLs, see [MS-ADTS].

In addition to being a valid LDAP URL, the Reference Attribute MUST adhere to the following format. This format is defined by using the extended BNF form specified in [C706].

Reference Attribute Value = "//" Domain "/cn=" Entry "," RestOfLDAPURL

Domain: MUST be a valid DNS name of the domain.

Entry: MUST be identical to the name component of the object's name service entry name specified by using the syntax described in section 2.2.2.

RestOfLDAPURL: MUST be the rest of the LDAP URL and MUST conform to the LDAP URL syntax specified in [MS-ADTS], without the domain and URL scheme ("ldap:").

2.2.5.1.3 RPC Syntax Identifier Attribute

An RPC Syntax Identifier Attribute represents an RPC_SYNTAX_IDENTIFIER structure. This attribute specifies either of the following properties.

- Identifier and version of an interface.
- Identifier and version of transfer syntax for an interface.

This structure MUST be specified as a string in the following format. The syntax of the format is according to extended BNF as specified in [C706].

```
UUIDAndVersion   =UUID "." Version
Version          =<5>*Digit "." <5>*Digit
Digit   = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

The UUID MUST be encoded as the string representation of the interface UUID, as specified in "Universal Unique Identifier" section in Part 4 of [C706]. The numeric value of each of the <5>*Digit productions in the preceding version MUST be less than or equal to 2^{16}-1.

2.2.5.2 rpcContainer Class

The rpcContainer class MUST represent the container in which all the RPC name service entries are created in AD. More information on the "Class rpcContainer" can be found in [MS-ADTS]. The following attributes on the rpcContainer class are accessed by this protocol.

```
unsigned long    nameServiceFlags
                 Optional
```

nameServiceFlags: If the value is nonzero, all locators in the domain MUST run in directory mode. If the value is 0, all locators in the domain MUST run in directory-only mode. The value is treated as a nonzero value when not set. More information on the Attribute nameServiceFlags see [MS-ADA3].

2.2.5.3 rpcServer Class

The rpcServer class MUST represent a RPC name service server entry in AD. See also "Class rpcServer" in [MS-ADTS]. The rpcServer class contains child elements of class rpcServerElement (section 2.2.5.6) that represent individual RPC interfaces exported to the corresponding server entry. The following attributes on the rpcServer class are accessed by the RPC Location Services Protocol.

```
string    rpcNsObjectID
```

```
          Optional, MultiValued
string    description
          Optional
string    RDN
          Mandatory
```

rpcNsObjectID: MUST be the list of object UUIDs exported to the corresponding Server Entry. Each object UUID MUST be stored in a string form encoded as defined in the "Universal Unique Identifier" section in Part 4 of [C706]. See also "Attribute rpcNsObjectID" in [MS-ADA3].

description: An implementation-specific informative text string for the name service entry. See also "Attribute description" in [MS-ADA1].

RDN: The RDN of the entry as specified in Name Service Entry RDN (section 2.2.5.1.1).

2.2.5.4 rpcProfile Class

The rpcProfile class MUST represent a RPC name service profile entry in AD. See also "Class rpcProfile" in [MS-ADTS]. The class contains child elements of class rpcProfileElement (section 2.2.5.7). The following attributes on this class are accessed by the RPC Location Services Extensions Protocol.

```
string    RDN
          Mandatory
```

RDN: The RDN of the entry as specified in Name Service Entry RDN (section 2.2.5.1.1).

2.2.5.5 rpcGroup Class

The rpcGroup class MUST represent a RPC name service group entry in AD. See also "Class rpcGroup" in [MS-ADTS]. The following attributes on this class are accessed by the RPC Location Services Extensions Protocol.

```
string    rpcNsGroup
          Optional, Multivalued
string    RDN
          Mandatory
```

rpcNsGroup: MUST be a set of references for the entries that are members of this group. Each reference is specified in Reference Attributes (section 2.2.5.1.2). For more information, see "Attribute rpcNsGroup" in [MS-ADA3]. The values in this attribute are unordered.

RDN: The RDN of the entry, MUST be as specified in Name Service Entry RDN (section 2.2.5.1.1).

2.2.5.6 rpcServerElement Class

The rpcServerElement class MUST represent a single RPC interface in a given RPC server entry in AD. See also "Class rpcServerElement" in [MS-ADTS]. Every instance of this class MUST be the child of an instance of class rpcServer. The following attributes on this class are accessed by the RPC Location Services Extensions Protocol.

```
string      rpcNsBindings
            Mandatory, Multivalued
string      rpcNsInterfaceID
            Mandatory
string      rpcNsTransferSyntax
            Mandatory
string      RDN
            Mandatory
```

rpcNsBindings: An array of one or more string bindings for this RPC interface. See also "Attribute rpcNsBindings" in [MS-ADA3]. The string bindings MAY optionally contain endpoint information. The format is described in "String Bindings" in Part 2 of [C706].

rpcNsInterfaceID: A string that encodes the interface identifier and version of this RPC interface, MUST be as specified in RPC Syntax Identifier Attribute (section 2.2.5.1.3). See also "Attribute rpcNsInterfaceID" in [MS-ADA3].

rpcNsTransferSyntax: A string that encodes the transfer syntax for this RPC interface, MUST be as specified in RPC Syntax Identifier Attribute (section 2.2.5.1.3). See also "Attribute rpcNsTransferSyntax" in [MS-ADA3].

RDN: Name of the entry. RDN MUST be the same as the rpcNsInterfaceID.

2.2.5.7 rpcProfileElement Class

The rpcProfileElement class represents a single entry in a given RPC profile in AD. See also "Class rpcProfileElement" in [MS-ADTS]. Every instance of this class must be the child of an instance of class rpcProfile. The following attributes on this class are accessed by the RPC Location Services Extensions Protocol.

```
string           rpcNsInterfaceID
                 Mandatory
unsigned long    rpcNsPriority
                 Mandatory
string           rpcNsAnnotation
                 Optional
string           rpcNsProfileEntry
                 Optional
```

rpcNsInterfaceID: A string that encodes the Interface Identifier and version of this RPC interface, as specified in RPC Syntax Identifier Attribute (section 2.2.5.1.3). See also "Attribute rpcNsInterfaceID" in [MS-ADA3].

rpcNsPriority: An integer that MUST represent the priority of the profile element as specified in "rpc_ns_profile_elt_add" in [C706] part 2. See also "Attribute rpcNsPriority" in [MS-ADA3].

rpcNsAnnotation: An optional informative text string for the entry. See also "Attribute rpcNsAnnotation" in [MS-ADA3]. This attribute MUST be ignored if set to an empty string.

rpcNsProfileEntry: MUST be a reference to the entry corresponding to this profile element. This attribute is specified in Reference Attributes (section 2.2.5.1.2). See also "Attribute rpcNsProfileEntry" in [MS-ADA3].

RDN: The RDN of the entry that MUST be the same as the RDN of the referred entry. The RDN of the entry MUST be as specified in Name Service Entry RDN (section 2.2.5.1.1).

3 Protocol Details

The relationship between various server, group, and profile entries is specified in "Name Service Attributes", [C706] section 2. This specification preserves those relationships in all respects except where explicitly stated otherwise.

The search algorithm used for lookup of bindings is defined in "Search Algorithm," in [C706] section 2. This specification maintains that algorithm as specified in all respects.

3.1 LocToLoc Common Details

This section specifies the details that are common to different locator roles.

3.1.1 Abstract Data Model

This section specifies a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The organization helps 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 the behavior described in this document.

3.1.1.1 RPC Services Container

In directory and directory-only modes, locators rely on an AD store in the domain of the computer for persistently storing and looking up name service entries represented as AD objects. These AD objects MUST reside under the CN=RpcServices, CN=System location under the Domain Naming Context of the computer domain. This container MUST be of class rpcContainer and MAY have direct children as instances of classes rpcServer, rpcGroup, and rpcProfile. Correspondence between name service entries and AD schema classes is specified in Name Service Entries in AD (section 1.3.3).

Entry FQDN: The **FQDN** of a given AD entry representation MUST be defined by concatenating the RDN of the entry (Name Service Entry RDN (section 2.2.5.1.1)) with the FQDN of the RPC services container.

Note The above conceptual data can be implemented using a variety of techniques. An implementation is at liberty to implement the data in any way it pleases.

3.1.2 Timers

No common timers are required across all locator roles.

3.1.3 Initialization

3.1.3.1 Mode Initialization

Any implementation of the RPC Location Services Protocol Extensions MUST determine its mode using the following algorithm.

1. The locator issues an LDAP query to retrieve the **nameServiceFlags** attribute of the AD RPC services container. Information from AD MUST be queried in the context of the security principal of the computer.

2. If the attribute is not retrieved successfully, the locator MUST set its mode to non-directory mode.<5>

3. If the attribute is retrieved successfully and is unspecified or specifies a nonzero value, the locator MUST set its mode to directory mode.<6>

4. Otherwise, if the attribute is retrieved successfully and is specified as zero, the locator MUST set its mode to directory-only.

3.1.3.2 Master and Non-Master Locator Initialization

Any implementation of the RPC Location Services Protocol Extensions MUST determine its role using the following algorithm.

1. The locator determines if the computer is joined to the domain and if it is running on a domain controller of the domain.<7>

2. Locators running on a domain controller MUST initialize as a master locator.

3. Locators running on a domain-joined computer, but not running as a domain controller, MUST initialize as a non-master locator. A non-master locator MUST initialize the list of potential master locators by enumerating the computers running as domain controllers in the domain.

4. Locators running on a non-domain-joined computer MUST initialize as a non-master locator. An implementation SHOULD change its role from non-master locator to master locator if no master locator is discovered by the master locator discovery process or if the master locators discovered are not reachable. As part of this process, the locator MUST perform master locator-specific initialization, as specified in section 3.4.1.3.

5. Any locator that becomes a master locator and responds to a master locator discovery query or sends a broadcast lookup request as specified in section 3.4 SHOULD continue to remain as a master locator.

6. Master locator reachability MAY be determined by making a call on the **I_nsi_ping_locator** method on the LocToLoc interface.<8>

3.1.4 Message Processing Events and Sequencing Rules

The ILocToLoc interface is used by client locators to forward lookup requests to master locators.

Methods in RPC Opnum Order

Method	Description
I_nsi_lookup_begin	Invoked by a client locator to enumerate the binding information for a set of RPC servers that satisfy a given set of criteria. Opnum: 0
I_nsi_lookup_done	Invoked to free any resources associated with the context handle returned by a preceding call to the **I_nsi_lookup_begin** method. Opnum: 1
I_nsi_lookup_next	Invoked to continue an enumeration of binding vectors that satisfy the criteria specified in a call to the **I_nsi_lookup_begin** method. The number of bindings in the binding_vector is limited by the parameter binding_max_count specified in the call to the **I_nsi_lookup_begin** method. Opnum: 2

Method	Description
I_nsi_entry_object_inq_next	Invoked to continue an enumeration initiated by a previous call to the **I_nsi_entry_object_inq_next** method. Opnum: 3
I_nsi_ping_locator	Invoked by the client to determine whether the target computer is available as a master locator. Opnum: 4
I_nsi_entry_object_inq_done	Invoked to free any resources associated with the context handle returned by a preceding call to the I_nsi_entry_object_inq_begin method. Opnum: 5
I_nsi_entry_object_inq_begin	Invoked to enumerate the object UUIDs on a name service entry. Opnum: 6

3.1.4.1 I_nsi_lookup_begin (Opnum 0)

The **I_nsi_lookup_begin** method is invoked by a client locator to enumerate the binding information for a set of RPC servers that satisfy a given set of criteria. The **Microsoft Interface Definition Language (MIDL)** syntax of the method is specified as follows:

```
void I_nsi_lookup_begin(
  [in] handle_t hrpcPrimaryLocatorHndl,
  [in] unsigned long entry_name_syntax,
  [in] STRING_T entry_name,
  [in, unique] RPC_SYNTAX_IDENTIFIER* interfaceid,
  [in, unique] RPC_SYNTAX_IDENTIFIER* xfersyntax,
  [in] NSI_UUID_P_T obj_uuid,
  [in] unsigned long binding_max_count,
  [in] unsigned long MaxCacheAge,
  [out] NSI_NS_HANDLE_T* import_context,
  [out] unsigned short* status
);
```

hrpcPrimaryLocatorHndl: MUST contain the RPC primitive binding handle, as specified in [C706] Part 3, "Interface Definition Language and Stubs."

entry_name_syntax: An identifier that represents the syntax used for entry_name. The value MUST be RPC_C_NS_SYNTAX_DCE.<9>

entry_name: A [UNICODE] string optionally specifying the entry name of the name service entry, using the syntax identified by the entry_name_syntax parameter, as specified in section 2.2.2. This parameter can optionally be null or an empty string.

interfaceid: An optional interface specification. Specified to request only bindings for server entries that have advertised interfaces compatible with this parameter. Interface compatibility is specified in section 3.4.1.5.1.

xfersyntax: An optional transfer syntax specification. Specified to request only bindings for server entries that have advertised interfaces compatible with this parameter. Interface compatibility is specified in section 3.4.1.5.1.

obj_uuid: An optional pointer to an object UUID specification. Specified to request only bindings for the server entries that export this object UUID. If the parameter is NULL or if it contains a null GUID, the parameter is ignored.

binding_max_count: The maximum number of elements allowed in the binding vector returned from the I_nsi_lookup_next method. If 0 is specified, then an appropriate implementation-specific default maximum MUST be used.<10>

MaxCacheAge: Specifies the maximum number of seconds that any results returned from a cache may have been present in the cache without being refreshed. This information is as specified in [C706], Part 2 "Name Service Caching."

import_context: On successful completion of this method, returns a context handle for enumerating binding vectors by using the **I_nsi_lookup_next** method. This context handle MUST be closed by using the I_nsi_lookup_done method.

status: A 16-bit value that indicates the results of the method call. In case of success, the value MUST be NSI_S_OK. The value MUST be a non-zero value on failure. All failures MUST be treated identically as failure of the whole enumeration process.

Return Values: This method does not return any values. RPC exceptions might be thrown from this method.

Exceptions Thrown

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

3.1.4.2 I_nsi_lookup_next (Opnum 2)

The **I_nsi_lookup_next** method is invoked to continue an enumeration of binding vectors that satisfy the criteria specified in a call to the I_nsi_lookup_begin method. The number of bindings in the binding_vector is limited by the parameter binding_max_count specified in the call to the I_nsi_lookup_begin method. The MIDL syntax of this method is specified as follows:

```
void I_nsi_lookup_next(
  [in] handle_t hrpcPrimaryLocatorHndl,
  [in] NSI_NS_HANDLE_T import_context,
  [out] NSI_BINDING_VECTOR_P_T* binding_vector,
  [out] unsigned short* status
);
```

hrpcPrimaryLocatorHndl: An RPC primitive binding handle, as specified in [C706] Part 3, "Interface Definition Language and Stubs."

import_context: A context handle returned by a preceding call to the I_nsi_lookup_begin method.

binding_vector: On successful completion, returns a vector containing bindings that satisfy the criteria defined in the preceding call to the I_nsi_lookup_begin method.

status: A 16-bit value that indicates the result of the method call. All other values besides those listed below MUST be treated as failures and MUST be treated identically.

Value	Meaning
NSI_S_OK 0x00000000	The call returned successfully. There may be additional bindings that satisfy the criteria.
NSI_S_NO_MORE_BINDINGS 0x00000001	There are no more bindings that satisfy the criteria.

Return Values: This method does not return any values.

Exceptions Thrown

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

3.1.4.3 I_nsi_lookup_done (Opnum 1)

The **I_nsi_lookup_done** method is invoked to free any resources associated with the context handle returned by a preceding call to the **I_nsi_lookup_begin** method. The MIDL syntax of this method is specified as follows:

```
void I_nsi_lookup_done(
  [in] handle_t hrpcPrimaryLocatorHndl,
  [in, out] NSI_NS_HANDLE_T* import_context,
  [out] unsigned short* status
);
```

hrpcPrimaryLocatorHndl: An RPC primitive binding handle, as specified in [C706] Part 3, "Interface Definition Language and Stubs."

import_context: A context handle returned by the server from a preceding call to the **I_nsi_lookup_begin** method. On successful completion, this parameter MUST be set to NULL by the server, and MUST NOT be modified on failure.

status: A 16-bit value that indicates the results of the method call. In case of success, the value will contain NSI_S_OK. The value will contain a non-zero value on failure. All failures MUST be treated identically as failure of the freeing process initiated by this method.

Return Values: This method does not return any values.

Exceptions Thrown

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

3.1.4.4 I_nsi_ping_locator (Opnum 4)

The **I_nsi_ping_locator** method is invoked by the client to determine if the target computer is available as a master locator. The Microsoft Interface Definition Language (MIDL) syntax of the method is specified as follows:

```
void I_nsi_ping_locator(
  [in] handle_t hLocatortoPing,
  [out] error_status_t* status
```

```
);
```

hLocatortoPing: An RPC primitive binding handle, as specified in [C706] Part 3, "Interface Definition Language and Stubs."

status: A 16-bit value that indicates the results of the method call. In case of success, the value will contain NSI_S_OK, or a non-zero value on failure. All failures MUST be treated identically as failure of the pinging process initiated by this method, and the target computer SHOULD be treated as unavailable as a master locator.

Return Values: This method does not return any values.

Exceptions Thrown

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

3.1.4.5 I_nsi_entry_object_inq_begin (Opnum 6)

The **I_nsi_entry_object_inq_begin** method is invoked to enumerate the object UUIDs on a name service entry. The MIDL syntax of the method is specified as follows:

```
void I_nsi_entry_object_inq_begin(
  [in] handle_t hrpcPrimaryLocatorHndl,
  [in] unsigned long EntryNameSyntax,
  [in] STRING_T EntryName,
  [out] NSI_NS_HANDLE_T* InqContext,
  [out] unsigned short* status
);
```

hrpcPrimaryLocatorHndl: An RPC primitive binding handle, as specified in [C706] Part 3, "Interface Definition Language and Stubs."

EntryNameSyntax: An identifier that represents the syntax used for the entry_name parameter. The value MUST be RPC_C_NS_SYNTAX_DCE.

EntryName: A Unicode [UNICODE] string specifying the entry name of the name service entry, using the syntax identified by the entry_name_syntax parameter, as specified in section 2.2.2.

InqContext: On successful completion, returns a context handle for enumerating object UUID vectors by using the **I_nsi_entry_object_inq_next** method. This context handle MUST be closed by using the **I_nsi_entry_object_inq_done** method.

status: A 16-bit value that indicates the results of the method call. In case of success, the value will contain NSI_S_OK, or a non-zero value on failure. All failures MUST be treated identically as failure of the whole enumeration process.

Return Values: This method does not return any values. RPC exceptions can be thrown from this method.

Exceptions Thrown

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

3.1.4.6 I_nsi_entry_object_inq_next (Opnum 3)

The **I_nsi_entry_object_inq_next** method is invoked to continue an enumeration initiated by a previous call to the **I_nsi_entry_object_inq_begin** method. The MIDL syntax of the method is specified as follows:

```
void I_nsi_entry_object_inq_next(
  [in] handle_t hrpcPrimaryLocatorHndl,
  [in] NSI_NS_HANDLE_T InqContext,
  [out] NSI_UUID_VECTOR_P_T* uuid_vec,
  [out] unsigned short* status
);
```

hrpcPrimaryLocatorHndl: An RPC primitive binding handle, as specified in [C706] Part 3, " Interface Definition Language and Stubs."

InqContext: A context handle returned by the server from a preceding call to the I_nsi_entry_object_inq_begin method.

uuid_vec: On successful completion, returns a vector of object UUIDs for the name service entry.

status: A 16-bit value that indicates the results of the method call. In case of success the value will contain NSI_S_OK, or a non-zero value on failure. All failures MUST be treated identically as failure of the continuation of the enumeration process.

Return Values: This method does not return any values. RPC exceptions can be thrown from this method.

Exceptions Thrown

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

3.1.4.7 I_nsi_entry_object_inq_done (Opnum 5)

The **I_nsi_entry_object_inq_done** method is invoked to free any resources associated with the context handle returned by a preceding call to the **I_nsi_entry_object_inq_begin** method. The MIDL syntax of the method is specified as follows:

```
void I_nsi_entry_object_inq_done(
  [in, out] NSI_NS_HANDLE_T* InqContext,
  [out] unsigned short* status
);
```

InqContext: A context handle returned by the server from a preceding **I_nsi_entry_object_inq_begin** call. On successful completion, this parameter MUST be set to NULL by the server, and MUST NOT be modified on failure.

status: A 16-bit value that indicates the results of the method call. In case of success the value will contain NSI_S_OK, or a non-zero value on failure. All failures MUST be treated identically as failure of the freeing of resources initiated by this method.

Return Values: This method does not return any values. RPC exceptions can be thrown from this method.

Exceptions Thrown

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

3.1.5 Timer Events

No common timer events are applicable across all locator roles.

3.1.6 Other Local Events

No other local events are applicable across all locator roles.

3.2 LocToLoc Server Locator Details

A server locator receives calls to export entries and, based on its mode, optionally responds to broadcast lookup requests.

For more information on Windows APIs that implement RPC name service functionality, see section 6.

3.2.1 Non-Directory Mode

In this mode, a server locator stores server entry information in a local non-persistent cache and uses the cached information to respond to broadcast lookup requests. Only server entries are supported in this mode; profile and group entries are not supported.

3.2.1.1 Abstract Data Model

Entry Cache: Each server locator MUST maintain a cache of its server entries and its associated object UUIDs and interface information. This cache is used to respond to broadcast lookup requests. The server entries in the cache MUST be indexed as follows:

- By name of the server entry.
- By interface identifier of the interfaces exported in the server entry.

3.2.1.2 Timers

No timers are required in this mode.

3.2.1.3 Initialization

The entry cache MUST be initialized to an empty list that contains no server entries.

The server locator initializes the mailslot used to receive broadcast lookups (section 2.1) for name service entries, and then starts listening on the mailslot for queries.

3.2.1.4 Higher-Layer Triggered Events

A higher-level protocol or application may make an implementation-specific call to modify server entries. Group and profile entries MUST NOT be supported in this mode.

When a server entry has been cached in the entry cache, and the server entry is later modified by an implementation-specific call, the cached entry MUST be updated with the same modifications.

3.2.1.5 Message Processing Events and Sequencing Rules

When a broadcast lookup request for server entries is received into a QueryPacket structure, the following actions MUST be taken.

1. The server locator MUST read the computer name of the requester from the **WkstaName** field in the QueryPacket structure.
2. The server locator MUST extract the objectUUID, interfaceid, and entry_name information from the corresponding fields specified in the QueryPacket.
3. The server locator MUST locate interfaces exported to the server entries in the entry cache that match the request, as specified in the following items.
   - Entry Name Criterion: An entry name criterion MUST be treated as unspecified if the entry_name field is filled with zero bytes.
     1. If the entry name is specified with a domain name part that does not match the NetBIOS domain name of the computer in a case-insensitive comparison, all cached entries MUST be treated as not matching the request's entry name criterion.
     2. If the entry name is specified with no domain name part, any cached entry whose name matches the specified entry name in a case-insensitive comparison MUST be treated as matching the request's entry name criterion. All the Interfaces exported to this entry MUST be treated as matching the request's entry name criterion.
     3. If the entry name is unspecified, all cached entries MUST be treated as matching the request's entry name criterion. Any interface exported to any entry MUST be treated as matching the request's entry name criterion.
   - Interface Identifier Criterion: An interface identifier criterion MUST be treated as unspecified if the interface field in the QueryPacket is filled with zero bytes.
     1. If the interface identifier is specified, any interfaces with a UUID that matches the QueryPacket interface identifier UUID MUST be treated as matching the request's interface identifier criterion. The interface version MUST be ignored for evaluating this criterion.
     2. If the interface identifier is unspecified, all cached entries MUST be treated as matching the request's interface identifier criterion. Any interface exported to any entry MUST be treated as matching the request's interface identifier criterion.
   - The object UUID value MUST be ignored.
4. Interfaces that match both criteria MUST be treated as matching the request. If both criteria are unspecified, the request MUST be treated as matching no interfaces. Matching interfaces and the server entries to which it belongs MUST be used to form a QueryReply structure.

3.2.1.5.1 Broadcast Lookup Response

The server locator MUST compose one or more QueryReply messages in response to the QueryPacket as follows:

1. Initialize the **Domain** field in QueryReply structure with the NetBIOS domain name of the computer. If the computer is a non-domain-joined computer, the **Domain** field MUST be initialized to an empty string.

2. Form a sequence of Reply Buffers. For each binding in a matching interface, the server locator initializes a Reply Buffer structure.

    1. **entryName** field of the Reply Buffer is initialized with the name of the server entry.

    2. objUUID MUST be initialized with the list of object UUIDs exported to the server entry.

    3. fpr.Interface and fpr.XferSyntax MUST be initialized, respectively, by the interface identifier and transfer syntax of the matching interface.

3. The **Buffer** field in QueryPacket MUST be filled with the sequence of the Reply Buffer structures.

4. If the QueryReply structure reached its maximum length, the server locator MUST queue the QueryReply structure for transmission, and then initialize an additional QueryReply to hold any remaining reply buffers.

5. All the QueryReply structures composed MUST be sent individually to the requester specified in the QueryPacket.WkstaName over Remote Mailslot Protocol to the address: \Mailslot\RpcLoc_c.

3.2.1.6 Timer Events

No timer events are applicable in this mode.

3.2.1.7 Other Local Events

No other local events are applicable in this mode.

3.2.2 Directory-Only Mode

In this mode, when a name service entry is exported, a persistent entry is created in AD. Server, profile, and group entries are supported and corresponding objects are created in AD by using the schema classes specified in section 1.3.3. In this mode, the server locator does not maintain any local cache of server entries and does not listen or respond to broadcast lookup requests.

3.2.2.1 Abstract Data Model

There is no specific abstract data model in this mode. Active Directory MUST be used as the store for a persistent representation of the name service entries, as specified in section 3.1.1.1.

3.2.2.2 Timers

No timers are required in this mode.

3.2.2.3 Initialization

An implementation MAY cache a connection to AD for optimization.<11>

3.2.2.4 Higher-Layer Triggered Events

A higher-level protocol or application can make an implementation-specific call to modify server, group, or profile entries.

When a name service entry is modified, the server locator MUST update the AD of the computer's domain with the modification by using LDAP, as specified in [MS-ADTS]. This LDAP request SHOULD be made in the context of the security principal who originated the call into the RPC name service. AD schema classes are specified in section 2.2.5.

3.2.2.4.1 Update of a Server Entry

Server entries MUST be represented by using the rpcServer class in AD. When a server entry is modified by adding or removing an interface, interface binding, or an object UUID, the following actions MUST be taken.

1. The server locator MUST form the entry FQDN, as specified in section 3.1.1.1, and issues an LDAP query to retrieve the corresponding AD object.
2. The server locator MUST create or modify the AD object as follows:
    1. If an AD object exists with the entry FQDN, the server locator MUST verify that the object represents a server entry by verifying that the AD object is of class rpcServer.
    2. If the AD object represents a server entry and if the description attribute on the AD object is **Created Entry**, the AD object MUST be treated as an empty name service entry, as specified in section 3.2.2.4.4. The server locator MUST modify the description to an implementation-specific value other than **Created Entry**.
    3. If no AD object exists for the entry FQDN, the server locator MUST create a new AD object of class rpcServer to represent the server entry.
3. The server locator MUST compare information in this RPC name service modification with the data already in AD. If there are any differences, the server locator MUST modify the new or pre-existing server entry as follows:
    1. The server locator MUST update the object UUIDs of the server entry to match the data in the export.
    2. The server locator MUST create or modify the corresponding child AD object of type rpcServerElement (section 2.2.5.6) to update its interface information.
        1. The server locator MUST generate the RDN attribute of the rpcServerElement as specified in section 2.2.5.6.
        2. The server locator MUST modify the **rpcNsBindings** attribute with bindings in this export.
        3. The server locator MUST modify the **rpcNsInterfaceID** attribute with the interface identifier and version.
        4. The server locator MUST modify the **rpcNsTransferSyntax** attribute with the transfer syntax of the interface.

3.2.2.4.2 Update of a Group Entry

Group entries MUST be represented by using the rpcGroup class in AD. When a group entry is modified by adding a member or removing a member, the following actions MUST be taken.

1. The server locator forms the entry FQDN, as specified in section 3.1.1.1, and issues an LDAP query to retrieve the corresponding AD object.
2. The server locator MUST create or modify the AD object as follows:

1. If an AD object exists with the entry FQDN, the server locator MUST verify that the AD object represents a group entry by verifying that the AD object is of class rpcGroup.

2. If the AD object is of class rpcServer, and if its description matches the string Created Entry, the AD object MUST be treated as an empty name service entry, as specified in section 3.2.2.4.4. The server locator MUST delete the AD object and recreate an AD object of class rpcGroup in its place.

3. If no AD object exists for the entry FQDN, the server locator MUST create a new object of class rpcGroup to represent the group entry.

3. The server locator MUST compare information in this RPC name service modification with the data already in AD. If there are any differences, the server locator MUST modify the new or pre-existing group entry as follows:

   - The server locator MUST update group members represented in the **rpcNsGroup** attribute.

3.2.2.4.3 Update of a Profile Entry

Profile entries MUST be represented by using the rpcProfile class in AD. When a profile entry is modified by adding or removing a profile element, the following actions MUST be taken.

1. The server locator MUST form the entry FQDN, as specified in section 3.1.1.1, and issues an LDAP query to retrieve the corresponding AD object.

2. The server locator MUST create or modify the AD object as follows:

   1. If an AD object exists with the entry FQDN, the server locator MUST verify that the AD object represents a profile entry by verifying that the AD object is of class rpcProfile.

   2. If the AD object is of class rpcServer, and if its description matches the string Created Entry, the object MUST be treated as an empty name service entry, as specified in section 3.2.2.4.4. The server locator MUST delete the AD object and recreate an AD object of class rpcProfile in its place.

   3. If no AD object exists for the entry FQDN, the server locator MUST create a new object of class rpcProfile to represent the Profile Entry.

3. The server locator MUST compare information in this RPC name service modification with the data already in AD. If there are any differences, the server locator MUST modify the new or pre-existing profile entry as follows:

4. The server locator MUST create or modify the corresponding child AD object of type rpcProfileElement 2.2.5.7 to represent a profile element.

   1. The server locator MUST generate the RDN attribute of the rpcProfileElement, as specified in section 2.2.5.7.

   2. The **rpcNsProfileEntry** attribute MUST be initialized as a Reference attribute referring to the name service entry referred to by this profile element, as modified LDAP URL string referring to the name service entry's actual location in AD.

   3. The **rpcNsInterfaceId**, **rpcNsPriority**, and **rpcNsAnnotation** attributes MUST be updated with the interface identifier, and Priority and Annotation properties of the profile entry, respectively.

3.2.2.4.4 Creation of a New Entry

The server locator MUST take the following actions to create an AD object representing a name service entry.

1. The server locator forms the entry FQDN, as specified in section 3.1.1.1, and issues an LDAP query.
2. If an AD object exists with the entry FQDN, the server locator MUST make no further modifications.
3. If no AD object exists for the entry FQDN, the server locator MUST create a new object of class rpcServer to represent an empty name service entry. The server locator MUST update the description of the entry to be "Created Entry".

3.2.2.5 Message Processing Events and Sequencing Rules

All message processing events and sequencing rules are as specified in section 3.2.2.4 in the context of processing higher-layer events.

3.2.2.6 Timer Events

No timer events are applicable in this mode.

3.2.2.7 Other Local Events

No other local events are applicable in this mode.

3.2.3 Directory Mode

In directory mode, when a name service entry MUST be exported, a persistent entry is created in AD.

Server, group, and profile entries are supported and corresponding objects are created in AD, as specified in section 1.3.3. In addition, in directory mode, the server locator stores server entry information in a non-persistent cache and uses the cached information to respond to broadcast lookup requests.

3.2.3.1 Abstract Data Model

The abstract data model is as specified in sections 3.2.1.1 and 3.2.2.1.

3.2.3.2 Timers

No timers are required in this mode.

3.2.3.3 Initialization

The server locator initializes as specified in sections 3.2.1.3 and 3.2.2.3.

3.2.3.4 Higher-Layer Triggered Events

A higher-level protocol or application can make an implementation-specific call to modify server, group, or profile entries.

When an entry is modified, the server locator MUST update the AD store of the computer's domain with information, as specified in section 3.2.2.4.

3.2.3.4.1 Updating a Server Entry

The entry MUST be exported to AD, as specified in section 3.2.2.4.1.

In addition, the server locator MUST update its cache with information as specified in section 3.2.1.4.

3.2.3.4.2 Updating a Group Entry

See section 3.2.2.4.2.

3.2.3.4.3 Updating a Profile Entry

See section 3.2.2.4.3.

3.2.3.5 Message Processing Events and Sequencing Rules

When a broadcast lookup request for server entries is received, the server locator MUST respond, as specified in section 3.2.1.5.

Additional message processing events and sequencing rules are specified in section 3.2.2.5, in the context of processing higher-layer events.

3.2.3.6 Timer Events

No timer events are applicable in this mode.

3.2.3.7 Other Local Events

No other local events are applicable in this mode.

3.3 LocToLoc Client Locator Details

The client locator receives lookup requests from applications and higher-layer protocols and returns results from RPC name service to the caller.

For details on Windows APIs that implement RPC name service functionality, see section 6.

3.3.1 Non-Directory Mode

In non-directory mode, the client locator only supports the look up of server entries; profile and group entries are not supported. When a request for lookup is received, the client locator MUST forward the request to the master locator and collect the results, if the results cannot be found in the cache.

3.3.1.1 Abstract Data Model

Discovered Entries Cache: The client locator MAY maintain a local cache of recently discovered server entries with its associated object UUIDs and interface information. Each server entry in the cache MUST also have the time stamp when it was added to the cache so it can be removed if necessary. This value MUST also be used to calculate whether the name service entry has expired.<12>

Master Locator Cache: The client locator MUST maintain a list of master locators that can be used to forward the request.

3.3.1.2 Timers

Master Locator Response Timer: The client locator MUST use this timer to wait for responses to a master locator discovery request. This timer is started when the client locator sends a master locator discovery request.<13>

3.3.1.3 Initialization

The Discovered Entries cache MUST be initialized as empty.

Master Locator Cache: On a domain-joined computer, the client locator MUST initialize with the list of locators running on the domain controllers for the computer's domain. Domain controllers can be discovered as specified in [MS-ADTS].

On a non-domain-joined computer, the client locator MUST initialize this list to an empty list.

3.3.1.4 Higher-Layer Triggered Events

A higher-level protocol or application can make an implementation-specific call to look up information from a server entry. These can be to:

- Enumerate properties of a given server entry.
- Look up bindings with optional criteria specifying interface identifier and its transfer syntax, object UUID, or entry_name, as supported by the implementation.

When a request is received, the client locator MUST take the following actions:

1. The client locator MAY look up in the Discovered Entries cache for entries that have not yet expired.
2. The client locator MUST forward the request to the master locator by using the LocToLoc RPC interface 3.1.4, if there was no matching entry cached, or if matching entries in the cache have expired. The client locator MAY optimize by first returning results from the discovered entries from the cache before forwarding the request to the master locator.
3. A non-domain-joined client locator MAY initiate a master locator Discovery request if the master locator cache is empty 3.3.1.4.3. It MAY use any of the locators that responded to the request as the master locator to forward the lookup request.
4. The client locator MUST take the following actions to forward the request to master locator 3.3.1.4.1 and 3.3.1.4.2.<14>

3.3.1.4.1 Binding Lookup

1. The client locator MUST invoke the I_nsi_lookup_begin method with the following parameters.
    1. If an entry name is specified in the request, the client locator MUST initialize the entry_name parameter with the entry name in the syntax specified by RPC_C_NS_SYNTAX_DCE and the entry_name_syntax MUST be initialized to RPC_C_NS_SYNTAX_DCE. If entry name is not specified in the request, this parameter MUST be set to NULL.

2. If an interface is specified in the request, the client locator MUST initialize the *interfaceid* parameter with the interface identifier and version information. If interface is not specified in the request, this parameter MUST be set to NULL.

3. If a transfer syntax is specified for the interface specified in the request, the client locator MUST initialize the *xfersyntax* parameter with the transfer syntax identifier and version information. If a transfer syntax is not specified in the request, this parameter MUST be set to NULL.

4. If object UUIDs are specified in the request, the client locator MUST initialize the *obj_uuid* parameter with the object UUIDs specified in the request. If the object UUIDs are not specified in the request, this parameter MUST be set to NULL.

5. The client locator MUST initialize a value for the Binding_max_count as appropriate for the implementation.<15>

6. The client locator MUST initialize a value for the MaxCacheAge based on the request.<16>

2. The client locator MUST use the context handle received to enumerate results matching the criteria by invoking the **I_nsi_lookup_next** method with the context handle.

3. Client locator MUST invoke the **I_nsi_lookup_done** method with the context handle to free resources associated with the context handle.

3.3.1.4.2 Object UUID Lookup

1. The client locator MUST call the **I_nsi_entry_object_inq_begin** method with the following parameters.

   - The client locator MUST initialize the *entry_name* parameter with the entry name in the syntax specified by RPC_C_NS_SYNTAX_DCE and entry_name_syntax MUST be initialized to RPC_C_NS_SYNTAX_DCE.

2. The client locator MUST use the context handle received to enumerate object UUIDs by invoking the **I_nsi_entry_object_inq_next** method with the context handle.<17>

3. Client locator MUST invoke the **I_nsi_entry_object_inq_done** method with the context handle to free resources associated with the context handle.

3.3.1.4.3 Master Locator Discovery

A non-domain-joined client locator MAY initiate a locator discovery process. To initiate this, the client locator MUST take the following actions.

1. The client locator MUST wait for any ongoing broadcast request to complete.

2. The client locator MUST form a QUERYLOCATOR structure and initialize in the **RequesterName** field with the NetBIOS name of the computer on which it is running.

3. The client locator MUST broadcast the resulting message request to all reachable computers as specified for master locator discovery in section 2.1

4. The client locator MUST start the master locator response timer and initiate a wait for the response on the mailslot, as specified for master locator discovery in section 2.1

   - While the master locator response timer has not yet expired, the client locator MUST receive each valid response into a QUERYLOCATORREPLY structure.

5. On expiration of the master locator response timer, the client locator MUST stop processing responses.

6. The client locator MUST update the master locator cache with the **SenderName** field in all valid received QUERYLOCATORREPLY structures.

3.3.1.5 Message Processing Events and Sequencing Rules

All message processing events and sequencing rules are specified in section 3.3.1.4 in the context of processing higher-layer events.

3.3.1.6 Timer Events

On expiration of the master locator response timer, the client locator MUST stop processing responses to the master locator discovery request.

3.3.1.7 Other Local Events

No other local events are applicable in this mode.

3.3.2 Directory-Only Mode

In directory-only mode, the client locator supports the lookup of server, group, and profile entries. When a request for lookup is received, a lookup for the corresponding object is made in AD.

3.3.2.1 Abstract Data Model

There is no specific abstract data model in directory-only mode. Active Directory is used as the store for a persistent representation of the name service entries.

3.3.2.2 Timers

No timers are required in this mode.

3.3.2.3 Initialization

An implementation MAY cache a connection to AD for optimization.<18>

3.3.2.4 Higher-Layer Triggered Events

A higher-level protocol or application can make an implementation-specific call to look up information from a name service entry. These can be to:

- Look up server, group, or profile entries and enumerate their properties.
- Look up bindings with optional criteria specifying interface identifier and its transfer syntax, object UUID, or entry_name, as supported by the implementation.

The client locator looks up AD for the name service entries specified in the lookup request in the context of the security principal who originated the call into the RPC name service. This is specified in the following sections.

3.3.2.4.1 Query with Entry Name

If the entry name is specified in the lookup, the following actions MUST be taken.

1. The client locator MUST form the entry FQDN, as specified in section 3.1.1.1 and issue an LDAP query.

2. The client locator MUST check whether the AD object is of class rpcServer, rpcGroup, or rpcProfile to determine whether it represents a name service entry of type server, group, or profile entry, respectively.

3. If the object is of class rpcServer or rpcProfile, the client locator MUST issue an LDAP query to look up child AD objects representing the interfaces or profile elements associated with the name service entry.

4. If the lookup request is to enumerate bindings under the name service entry, the client locator MUST issue LDAP queries as specified below.

5. If the name service entry is a group entry, the client locator MUST issue LDAP queries to enumerate the name service entries that are members of the group. The entry FQDN is formed by concatenating the scheme(ldap:) to the LDAP URL strings in the rpcNsGroup attribute, as specified in section 2.2.5.1.2.

6. If the name service entry is a profile entry, the client locator MUST issue LDAP queries to enumerate the name service entries that are referred to by the profile elements. The entry FQDN is formed by concatenating the scheme (ldap:) to the LDAP URL string in the rpcNsProfileEntry attribute, as specified in section 2.2.5.1.2.<19>

3.3.2.4.2 Query Without Entry Name

If an entry name is not specified in a lookup operation, the client locator MUST treat it as a look up of a server entry, and the following actions MUST be taken.

1. The client locator MUST issue a one-level LDAP query under the RPC services container with one of the following queries.

   - (& (objectClass = rpcServer) (rpcNsObjectID=<string object UUID>)) if an object UUID is specified in the request. The object UUID is encoded as the string representation of the object UUID, as specified in "Universal Unique Identifier" in [C706] Part 4.

   - (& (objectClass = rpcServer)) if no object UUID is specified in the request.

2. For each matching object returned from AD, the client locator MUST enumerate all children to assemble a list of interfaces and their bindings exported in the Server Entry.

3. If the lookup request is to enumerate bindings, the request may specify an interface criteria by specifying the interface identifier and version information. The client locator MUST perform further refinement of the search results returned from AD to only return binding information from interfaces that match the interface specification.

3.3.2.5 Message Processing Events and Sequencing Rules

All message processing events and sequencing rules are specified in section 3.3.2.4 in the context of processing higher-layer events.

3.3.2.6 Timer Events

No timer events are applicable in this mode.

3.3.2.7 Other Local Events

No other local events are applicable in this mode.

3.3.3 Directory Mode

In this mode, the client locator supports the lookup of server, group, or profile entries. When a request for lookup is received, the client locator does a lookup in AD. If the name service entry is not found in AD, the client locator forwards the request to a master locator. Note that since this mode is only valid on a domain-joined computer, none of the non-domain-joined behavior (including master locator discovery) is applicable in this mode.

3.3.3.1 Abstract Data Model

The abstract data model is as specified in sections 3.3.1.1 and 3.3.2.1.

3.3.3.2 Timers

No timers are required in this mode.

3.3.3.3 Initialization

The client locator initializes as specified in section 3.3.1.3 for domain-joined computers.

3.3.3.4 Higher-Layer Triggered Events

A higher-level protocol or an application can make an implementation-specific call to look up information from a name service entry. These can be to:

Look up server, group, or profile entries, and enumerate their properties.

Look up bindings with some optional criteria like object UUID, or entry_name, as supported by the implementation.

3.3.3.4.1 Query with Entry Name

If the Entry Name is specified, the following actions MUST be taken.

1. The client locator MUST query AD for the entry as specified in section 3.3.2.4.1.
2. If the entry is found in AD, the client locator MUST return this information to the caller. The client locator MUST NOT forward the request to the master locator.
3. If the entry was not found in AD, the client locator MUST forward the request to the master locator, as specified in section 3.3.1.4, as applicable to a domain-joined computer.

3.3.3.4.2 Query Without Entry Name

If the Entry Name is not specified, the client locator MUST forward the request to the master locator 3.3.1.4 as applicable to a domain-joined computer.

3.3.3.5 Message Processing Events and Sequencing Rules

All message processing events and sequencing rules are specified in sections 3.3.1.4 and 3.3.2.4 in the context of processing higher-layer events.

3.3.3.6 Timer Events

No timer events are applicable in this mode.

3.3.3.7 Other Local Events

No other local events are applicable in this mode.

3.4 LocToLoc Master Locator Details

A master locator facilitates communication between client locators and server locators. A master locator MUST listen for forwarded requests from client locators on the LocToLoc RPC interface 3.1.4 and MUST broadcast the requests to reach any potential server locators. There MAY be multiple master locators, and different client locators may forward requests to different master locators. An implementation of a master locator MAY choose to cache the responses it receives from server locators by implementing the discovered entries cache as specified in section 3.4.1.1.<20>

3.4.1 Non-Directory Mode

The master locator facilitates lookup of server entries from computers on which the server entry is not directly exported. Profile and group entries are not supported in this mode.

3.4.1.1 Abstract Data Model

Discovered Entries Cache: Each master locator MAY maintain a cache of Server Entries and their associated object UUIDs and interface information that have been received as a response to a Broadcast Lookup request. Each Server Entry MUST also have the time stamp when it was added to the cache so it can be removed if necessary. This value MUST also be used to calculate whether the name service entry has expired. Master locator MAY use the cache entries that have not expired for a lookup request instead of broadcasting a Broadcast Lookup request.<21>

Client Response Cache: Master locator MUST maintain a cache of Server Entries that have been received as part of the broadcast but have not been enumerated by the client locator that invoked the method on the LocToLoc interface. This cache MAY be combined with Discovered Entries cache.<22>

3.4.1.2 Timers

Broadcast Response Timer: Master locator MUST use this timer to wait for messages in response to a Broadcast Lookup request. This timer is started when a Broadcast Lookup request is sent.<23>

3.4.1.3 Initialization

The master locator MUST initialize the Discovered Entries cache to an empty list.

The master locator MUST initialize the LocToLoc interface and begin listening for requests.

The master locator MUST initialize the Mailslot addresses 2.1 as follows:

- To receive responses to Broadcast Lookup request.
- To respond to master locator Discovery requests.

3.4.1.4 Higher-Layer Triggered Events

No higher-Layer triggered events are applicable in this mode.

3.4.1.5 Message Processing Events and Sequencing Rules

A master locator responds to the following:

- Lookup requests received on the LocToLoc RPC interface.
- Master locator discovery requests received.

3.4.1.5.1 Lookup Request

When a master locator receives a lookup request on the LocToLoc interface 3.1.4 the following actions MUST be taken.

1. The master locator MUST validate the parameters as follows:
   - entry_name_syntax:
     - This parameter MUST be RPC_C_NS_SYNTAX_DCE.<24>
   - entry_name:
     - The length of the parameter MUST not exceed the maximum length as specified in section 2.2.2.
     - The name MUST match the syntax specified by the entry_name_syntax parameter 2.2.2. The master locator MUST NOT validate the name field of the Entry Name to check for characters that are not allowed in an RDN.
     - This parameter MUST NOT be NULL for an **I_nsi_entry_object_inq_begin** call on the LocToLoc interface.
     - If parameter validation fails, processing MUST terminate and the master locator SHOULD either return an error in response or raise an RPC exception.
2. The master locator MUST locate any unexpired Server Entries in the Discovered Entries cache that match the request. If a match is found, the master locator MUST ignore the match if it has been in the cache for longer than the time specified in the *MaxCacheAge* parameter in **I_nsi_lookup_begin** call. If *MaxCacheAge* is not specified in the parameter, an appropriate default value MAY be used.<25>
3. If no such entries are found, the master locator MUST initiate Broadcast Lookup and collect the responses as specified in section 3.4.1.5.1.1.
4. If the Broadcast Lookup is initiated to handle a lookup request made by invoking the **I_nsi_lookup_begin** method on LocToLoc Interface, the master locator MUST compute compatible Interfaces in the responses in the following manner.
   1. If an interfaceid is specified, interfaces with the following properties MUST be considered compatible.
      - Interface Identifier of the interface equals the value in the SyntaxGUID field in *interfaceid* parameter.

- Major version of the interface equals the SyntaxVersion.MajorVersion field in the *interfaceid* parameter.
- Minor version of the interface is greater than or equal to the SyntaxVersion.MinorVersion field in *interfaceid* parameter.

2. If a TransferSyntax is specified by the parameter xfersyntax, interfaces with the following properties MUST be considered compatible.
   - TransferSyntax identifier of the interface equals the value in the SyntaxGUID field in *xfersyntax* parameter.
   - Major version of the TransferSyntax of the interface equals the SyntaxVersion.MajorVersion field in the *xfersyntax* parameter.
   - Minor version of the TransferSyntax of the interface is greater than or equal to the SyntaxVersion.MinorVersion field in *xfersyntax* parameter.

5. The master locator MUST return binding information from the compatible interfaces to the callers of **I_nsi_lookup_next**.
6. The master locator MUST return all object UUID information from the matching entry to the callers of **I_nsi_entry_object_inq_next** in a single UUID vector.
7. The master locator MUST maintain entries in the Client Response cache until the client locator has finished enumerating through the results. Master locator MUST consider a client locator has finished the enumeration if:
   1. Client locator invokes the **I_nsi_lookup_done** or **I_nsi_entry_object_inq_done** method with the corresponding context handle.
   2. A disconnect is detected by RPC as specified in section "Context Handle Rundown" in [C706] Part 3.

3.4.1.5.1.1 Broadcast Lookup

The master locator MUST initiate a Broadcast Lookup to look for entries exported to server locators on other computers. To initiate this, the master locator MUST take the following actions.

1. The master locator MUST wait for any ongoing Broadcast Lookup request to complete.
2. The master locator MUST initialize a QueryPacket structure as follows:
   1. Initialize the **WkstaName** field in the structure with the NetBIOS name of the computer.
   2. If the method called was **I_nsi_lookup_begin**, initialize the QueryPacket structure's Interface, Object, and EntryName fields with the interfaceid, obj_uuid and entry_name parameters specified in the request, respectively. For any parameter not specified in the request, the corresponding QueryPacket field MUST be initialized to all zeros.
   3. If the method called was **I_nsi_entry_object_inq_begin**, the master locator MUST initialize the QueryPacket structure's EntryName field with the entry_name parameter. The QueryPacket structure's interfaceid and obj_uuid field MUST be initialized to all zeros.
3. The QueryPacket structure MUST be broadcast to a destination chosen based on the entry name, as follows:

1. If the domainname component of entry name is absent in the entry_name parameter, the master locator MUST broadcast the request to all reachable computers on the network, as specified for broadcast lookup in section 2.1.

2. If the domainname component of entry name is present in the entry_name parameter, the master locator MUST broadcast the request to all computers in the domain represented by the domainname as specified for broadcast lookup in section 2.1.

4. The master locator MUST start the broadcast response timer and wait for responses, as specified for broadcast lookup in section 2.1.

5. While the broadcast response timer has not yet expired, the master locator MUST receive each valid response into a QueryReply structure.

6. The master locator MUST ignore any responses in which the domain field in the QueryReply structure does not match the NetBIOS name of the domain of the computer in a case-insensitive comparison.

7. The master locator MUST ignore the remaining reply buffers after an invalid reply buffer (section 2.2.4.2.2.2). A reply buffer MUST be considered invalid under the following conditions.

   1. If the type field in the fixed_part_of_reply structure does not match MailslotServerEntryType(1).

   2. If the length of the Unicode [UNICODE] string in **entryName** field is not equal to the field EntryNameLength in the fixed_part_of_reply structure.

   3. If the **entryName** field does not match the syntax specified in section 2.2.2. The master locator MUST NOT do any validations on the name field of the entry name to check for characters that are not allowed in an RDN 2.2.2.

   4. If the **objListSize** field exceeds the maximum length that can fit in the reply buffer.

   5. If the length of Binding field is not equal to the field **BindingLength** in the fixed_part_of_reply.

8. The master locator MUST update the client response cache and discovered entries cache with valid responses.

9. On expiration of the broadcast response timer, the master locator MUST stop processing responses.

3.4.1.5.2 Master Locator Response

When a master locator discovery request is received into a QUERYLOCATOR structure, the following actions MUST be taken.

1. The master locator MUST read the computer name of the requester from **RequesterName** field in the QueryPacket structure.

2. The master locator MUST initialize a QUERYLOCATORREPLY structure with the following values.

   1. The Hint field in QUERYLOCATORREPLY MUST be initialized with REPLY_MASTER_LOCATOR.

   2. The **Uptime** field in QUERYLOCATOR MUST be initialized with the amount of time the master locator has been running since startup.<26>

3. The **SenderName** field in QUERYLOCATOR MUST be initialized with the name of the computer.

3. The master locator MUST send the composed QUERYLOCATORREPLY structure to the requester over the Remote Mailslot Protocol at the destination address "\Mailslot\Resp_c"".

3.4.1.6 Timer Events

On expiration of the broadcast response timer, the master locator MUST stop accepting responses for the broadcast lookup request, as specified in section 3.4.1.5.1.1.

3.4.1.7 Other Local Events

No other local events are applicable in this mode.

3.4.2 Directory Mode

In this mode, the master locator behaves in a manner identical to the specification in section 3.4.1.

3.4.3 Directory-Only Mode

In this mode, a locator MUST NOT act as a master locator.

4 Protocol Examples

The following sections describe several operations as used in common scenarios to illustrate the function of the Remote Procedure Call Location Services Extensions.

4.1 Non-Directory Mode Operation

The following diagram shows an example of the protocol wherein all client, server, and master locators are in non-directory mode.

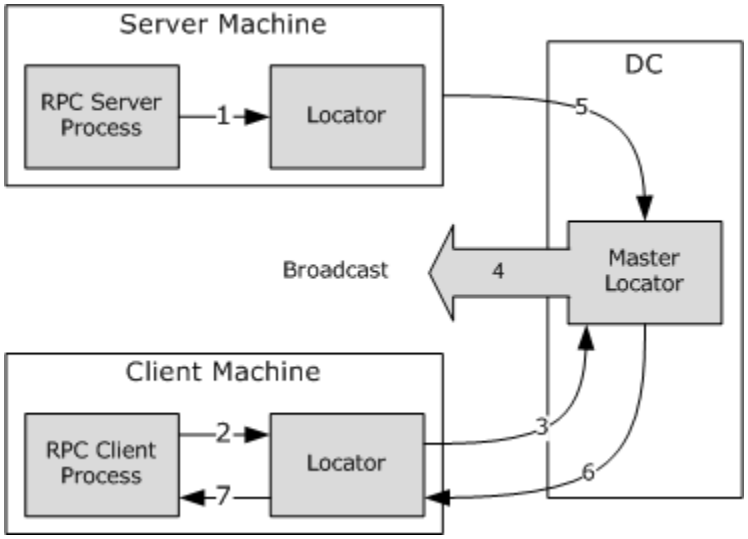


Figure 2: Non-directory mode

The sequence is described in the following steps.

1. The server exports its interface and the server locator updates its cache. For more information, see section 3.2.1.4.
2. The client initiates a lookup by name.
3. The client locator forwards the request to the master locator. For more information, see section 3.3.1.4.
4. The master locator sends out a broadcast. For more information, see section 3.4.1.5.
5. The server locator responds to the broadcast. For more information, see section 3.2.1.5.
6. The master locator gets the information and returns it to the client locator. For more information, see section 3.4.1.5.
7. The client locator returns the lookup handle to the client process.

4.2 Directory-Only Mode Operation

The following diagram shows an example of the protocol wherein all client, server, and master locators are in directory-only mode.

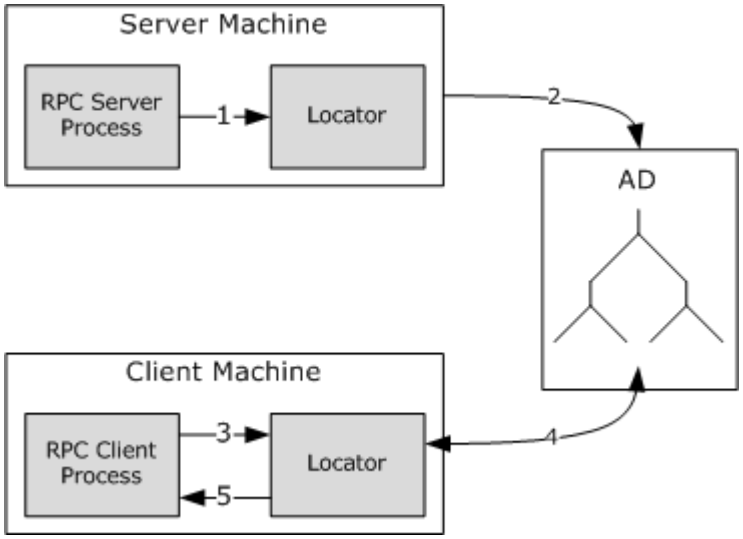


Figure 3: Directory-only mode

The sequence is described in the following steps.

1. Server Exports Interface.
2. The server locator exports the name service entry to AD. For more information, see section 3.2.2.4.1.
3. The client initiates a lookup by name.
4. The client locator initiates a DS lookup and finds the name service entry. For more information, see section 3.3.2.4.1.
5. The client locator returns the lookup handle to the client process.

4.3 Server in Non-Directory Mode and Client in Directory Mode

The following diagram shows an example of the protocol wherein the client and master locators are running in directory mode, and the server locator is in non-directory mode.

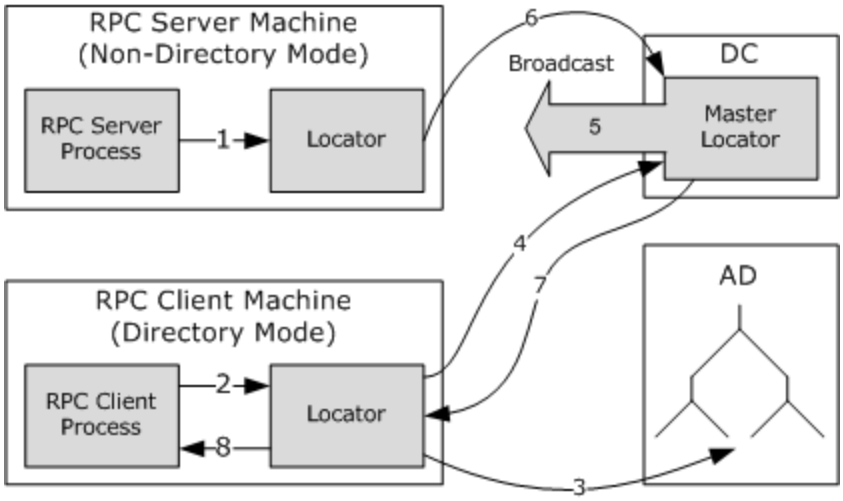


Figure 4: Directory mode

The sequence is described in the following steps.

1. The server exports interface and server locator updates to its local cache. For more information, see section 3.2.1.4.
2. The client initiates a lookup by name.
3. The client locator initiates a DS lookup and does not find the name service entry. For more information, see section 3.3.3.4.1.
4. The client locator forwards the request to the master locator. For more information, see section 3.3.3.4.1.
5. The master locator sends out a Broadcast request. For more information, see section 3.4.1.5.
6. The server locator responds to the request. For more information, see section 3.2.1.5.
7. The master locator gets the information and returns it to the client locator. For more information, see section 3.4.1.5.
8. The client locator returns the lookup handle to the client process.

5 Security

The following sections specify security considerations for implementers of the Remote Procedure Call Location Services Extensions.

5.1 Security Considerations for Implementers

Mailslots have no security on them. This MAY be disabled in directory-only mode, as specified in section 1.3.2.

ACLs on the default RPC services container are specified in [MS-ADTS].

The LocToLoc interface uses the default security settings and does not register any security providers, as specified in [MS-RPCE] section 3.3.3.3.

5.2 Index of Security Parameters

Security parameter	Section
Discussion of security on mailslots, default RPC containers, and the RPC interface.	Security Considerations for Implementers (section 5.1)

6 Appendix A: Full IDL

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

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

typedef struct _RPC_VERSION {
    unsigned short MajorVersion;
    unsigned short MinorVersion;
} RPC_VERSION;

typedef struct _RPC_SYNTAX_IDENTIFIER {
    GUID        SyntaxGUID;
    RPC_VERSION SyntaxVersion;
} RPC_SYNTAX_IDENTIFIER;

typedef [string, unique] wchar_t *STRING_T;

typedef [string] wchar_t *NSI_STRING_BINDING_T;

typedef [context_handle] void *NSI_NS_HANDLE_T;

typedef [unique] GUID *NSI_UUID_P_T;

typedef struct _NSI_BINDING_T {
    NSI_STRING_BINDING_T string;
    unsigned long   entry_name_syntax;
    STRING_T        entry_name;
} NSI_BINDING_T;

typedef struct _NSI_BINDING_VECTOR_T {
    unsigned long count;
    [size_is(count)] NSI_BINDING_T binding[*];
} NSI_BINDING_VECTOR_T;

typedef [unique] NSI_BINDING_VECTOR_T *NSI_BINDING_VECTOR_P_T;

typedef struct _NSI_UUID_VECTOR_T {
    unsigned long count;
    [size_is(count)] NSI_UUID_P_T uuid[*];
} NSI_UUID_VECTOR_T;

typedef [unique] NSI_UUID_VECTOR_T *NSI_UUID_VECTOR_P_T;


[
    uuid (e33c0cc4-0482-101a-bc0c-02608c6ba218),
    version (1.0),
    pointer_default (unique)
]
interface LocToLoc
{
    void I_nsi_lookup_begin(
        [in]        handle_t              hrpcPrimaryLocatorHndl,
```

```
        [in]        unsigned long         entry_name_syntax,
        [in]        STRING_T               entry_name,
        [in,unique] RPC_SYNTAX_IDENTIFIER *interfaceid,
        [in,unique] RPC_SYNTAX_IDENTIFIER *xfersyntax,
        [in]        NSI_UUID_P_T          obj_uuid,
        [in]        unsigned long         binding_max_count,
        [in]        unsigned long         MaxCacheAge,
        [out]       NSI_NS_HANDLE_T       *import_context,
        [out]       unsigned short        *status
    );

    void I_nsi_lookup_done(
        [in]     handle_t        hrpcPrimaryLocatorHndl,
        [in,out] NSI_NS_HANDLE_T *import_context,
        [out]    unsigned short  *status
    );

    void I_nsi_lookup_next(
        [in]  handle_t               hrpcPrimaryLocatorHndl,
        [in]  NSI_NS_HANDLE_T        import_context,
        [out] NSI_BINDING_VECTOR_P_T *binding_vector,
        [out] unsigned short         *status
    );

    void I_nsi_entry_object_inq_next(
        [in]  handle_t            hrpcPrimaryLocatorHndl,
        [in]  NSI_NS_HANDLE_T     InqContext,
        [out] NSI_UUID_VECTOR_P_T *uuid_vec,
        [out] unsigned short      *status
    );

    void I_nsi_ping_locator(
        [in]  handle_t      hLocatortoPing,
        [out] error_status_t *status
    );

    void I_nsi_entry_object_inq_done(
        [in,out] NSI_NS_HANDLE_T *InqContext,
        [out]    unsigned short  *status
    );

    void I_nsi_entry_object_inq_begin(
        [in]  handle_t        hrpcPrimaryLocatorHndl,
        [in]  unsigned long   EntryNameSyntax,
        [in]  STRING_T         EntryName,
        [out] NSI_NS_HANDLE_T *InqContext,
        [out] unsigned short  *status
    );
}
```

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 Vista
- 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 1.7: The Windows NT 4.0 version of this protocol always runs in non-directory mode.

<2> Section 2.2.3.6: The Windows implementation of locator does not store endpoint information.

<3> Section 2.2.4.3.2: This value is unused by the receiver of this message.

<4> Section 2.2.5.1.1: Locator ignores the domain name if any is specified in the entry name.

<5> Section 3.1.3.1: The Windows NT 4.0 locator always runs in this mode.

<6> Section 3.1.3.1: By default, nameServiceFlags is not set in AD.

<7> Section 3.1.3.2: Windows implementation of this specification invokes the DsRoleGetPrimaryDomainInformation method as specified in [MS-DSSP] tp retrieve this information.

<8> Section 3.1.3.2: On a non-domain-joined computer, the non-master locator discovers the master locator by issuing a master locator discovery query. In addition, locators also cache the sender of a broadcast lookup request as a master locator (see section 3.4.1.5.1). On a non-domain-joined computer, if none of the locators in its cache is accessible as determined by calling the **I_nsi_ping_locator** method on the target locator once, a non-master locator changes its role to be a master locator.

<9> Section 3.1.4.1: The master locator ignores this value if the entry_name is null or an empty string.

<10> Section 3.1.4.1: The locator assumes the default value of 100 if 0 is specified.

<11> Section 3.2.2.3: The server locator establishes and caches a connection to AD at startup.

<12> Section 3.3.1.1: The client locator maintains a local cache of recently discovered name service entries.

<13> Section 3.3.1.2: This timer is initialized to 3 seconds and has a granularity of 1 msec. Each time a response is received, the current time-out period is halved and used as the new time-out period.

<14> Section 3.3.1.4: The client locator only forwards the request to the master locator if the request cannot be satisfied locally first by looking at the unexpired name service entries in the

cache. The client locator only forwards the request to the master locator if the request cannot be satisfied locally first by looking at the unexpired name service entries in the cache.

<15> Section 3.3.1.4.1: Windows initializes this value to be the same as the value specified by the caller. If the caller has not specified a value, a default value of 100 is used.

<16> Section 3.3.1.4.1: Windows initializes this value to be the same as the value specified by the caller. If the caller has not specified a value, a default value of 7,200 is used.

<17> Section 3.3.1.4.2: Client locator invokes this method to retrieve all the object UUIDs associated with the name service entry.

<18> Section 3.3.2.3: The client locator initializes a connection to AD at startup.

<19> Section 3.3.2.4.1: The client locator eliminates cycles in this lookup by keeping track of the nodes visited by the search algorithm. The client locator performs the lookup of the referred entries only as needed.

<20> Section 3.4: Windows implements discovered entries cache.

<21> Section 3.4.1.1: Master locator only broadcasts a lookup request once it has returned the results from the Discovered Entries cache (taking into account the cache expiration age specified by the caller) and if the last Broadcast Lookup request for the entry was made before the cache expiration time.

<22> Section 3.4.1.1: Master locator updates the Discovered Entries cache with the information from Broadcast Lookup responses and uses the Discovered Entries cache to return results for subsequent enumerations by the Client Locators.

<23> Section 3.4.1.2: This timer is initialized to 3 seconds and has a granularity of 1 msec. Each time a response is received, the current time-out period is halved and used as the new time-out period.

<24> Section 3.4.1.5.1: The master locator ignores this value if the entry_name is null or an empty string.

<25> Section 3.4.1.5.1: The master locator uses a default value of 7,200 seconds if a value of 0 is specified and no calls have been made to update the cache expiration value on the master locator.

<26> Section 3.4.1.5.2: This parameter wraps around in 136 years.

8 Appendix C: API Mappings

The following table specifies mapping between APIs [C706] for RPC name service and corresponding APIs in Windows.

All APIs are as specified in [C706] Part 2 RPC API Manual.

RPC Name Service APIs in [C706]	Windows APIs
rpc_ns_binding_export	RpcNsBindingExportA/ RpcNsBindingExportW
rpc_ns_binding_import_begin	RpcNsBindingImportBeginA/ RpcNsBindingImportBeginW
rpc_ns_binding_import_done	RpcNsBindingImportDone
rpc_ns_binding_import_next	RpcNsBindingImportNext
rpc_ns_binding_inq_entry_name	No equivalent API
rpc_ns_binding_lookup_begin	RpcNsBindingLookupBeginA/ RpcNsBindingLookupBeginW
rpc_ns_binding_lookup_done	RpcNsBindingLookupDone
rpc_ns_binding_lookup_next	RpcNsBindingLookupNext
rpc_ns_binding_select	RpcNsBindingSelect
rpc_ns_binding_unexport	RpcNsBindingUnexportA/ RpcNsBindingUnexportW
rpc_ns_entry_expand_name	RpcNsEntryExpandNameA/ RpcNsEntryExpandNameW
rpc_ns_entry_inq_resolution	No equivalent API
rpc_ns_entry_object_inq_begin	RpcNsEntryObjectInqBeginA/ RpcNsEntryObjectInqBeginW
rpc_ns_entry_object_inq_done	RpcNsEntryObjectInqDone
rpc_ns_entry_object_inq_next	RpcNsEntryObjectInqNext
rpc_ns_group_delete	RpcNsGroupDeleteA/ RpcNsGroupDeleteW
rpc_ns_group_mbr_add	RpcNsGroupMbrAddA/ RpcNsGroupMbrAddW
rpc_ns_group_mbr_inq_begin	RpcNsGroupMbrInqBeginA/ RpcNsGroupMbrInqBeginW
rpc_ns_group_mbr_inq_done	RpcNsGroupMbrInqDone

RPC Name Service APIs in [C706]	Windows APIs
rpc_ns_group_mbr_inq_next	RpcNsGroupMbrInqNextA/ RpcNsGroupMbrInqNextW
rpc_ns_group_mbr_remove	RpcNsGroupMbrRemoveA/ RpcNsGroupMbrRemoveW
rpc_ns_import_ctx_add_eval	No equivalent API
rpc_ns_mgmt_binding_unexport	RpcNsMgmtBindingUnexportA/ RpcNsMgmtBindingUnexportW
rpc_ns_mgmt_entry_create	RpcNsMgmtEntryCreateA/ RpcNsMgmtEntryCreateW
rpc_ns_mgmt_entry_delete	RpcNsMgmtEntryDeleteA/ RpcNsMgmtEntryDeleteW
rpc_ns_mgmt_entry_inq_if_ids	RpcNsMgmtEntryInqIfIdsA/ RpcNsMgmtEntryInqIfIdsW
rpc_ns_mgmt_free_codesets	No equivalent API
rpc_ns_mgmt_handle_set_exp_age	RpcNsMgmtHandleSetExpAge
rpc_ns_mgmt_inq_exp_age	RpcNsMgmtInqExpAge
rpc_ns_mgmt_read_codesets	No equivalent API
rpc_ns_mgmt_remove_attribute	No equivalent API
rpc_ns_mgmt_set_attribute	No equivalent API
rpc_ns_mgmt_set_exp_age	RpcNsMgmtSetExpAge
rpc_ns_profile_delete	RpcNsProfileDeleteA/ RpcNsProfileDeleteW
rpc_ns_profile_elt_add	RpcNsProfileEltAddA/ RpcNsProfileEltAddW
rpc_ns_profile_elt_inq_begin	RpcNsProfileEltInqBeginA/ RpcNsProfileEltInqBeginW
rpc_ns_profile_elt_inq_done	RpcNsProfileEltInqDone
rpc_ns_profile_elt_inq_next	RpcNsProfileEltInqNextA/ RpcNsProfileEltInqNextW
rpc_ns_profile_elt_remove	RpcNsProfileEltRemoveA/ RpcNsProfileEltRemoveW

9 Index

L

M

N

O

P

Q

R

S

T

V

W