Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation ("this documentation") for protocols, file formats, data portability, computer languages, and standards support. Additionally, overview documents cover inter-protocol relationships and interactions.

- **Copyrights.** This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you can make copies of it in order to develop implementations of the technologies that are described in this documentation and can distribute portions of it in your implementations that use these technologies or in your documentation as necessary to properly document the implementation. You can also distribute in your implementation, with or without modification, any schemas, IDLs, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications documentation.

- **No Trade Secrets.** Microsoft does not claim any trade secret rights in this documentation.

- **Patents.** Microsoft has patents that might cover your implementations of the technologies described in the Open Specifications documentation. Neither this notice nor Microsoft's delivery of this documentation grants any licenses under those patents or any other Microsoft patents. However, a given Open Specifications document might be covered by the Microsoft Open Specifications Promise or the Microsoft Community Promise. If you would prefer a written license, or if the technologies described in this documentation are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.

- **License Programs.** To see all of the protocols in scope under a specific license program and the associated patents, visit the Patent Map.

- **Trademarks.** The names of companies and products contained in this documentation might be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights. For a list of Microsoft trademarks, visit www.microsoft.com/trademarks.

- **Fictitious Names.** The example companies, organizations, products, domain names, email addresses, logos, people, places, and events that are depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

**Reservation of Rights.** All other rights are reserved, and this notice does not grant any rights other than as specifically described above, whether by implication, estoppel, or otherwise.

**Tools.** The Open Specifications documentation does not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments, you are free to take advantage of them. Certain Open Specifications documents are intended for use in conjunction with publicly available standards specifications and network programming art and, as such, assume that the reader either is familiar with the aforementioned material or has immediate access to it.

**Support.** For questions and support, please contact dochelp@microsoft.com.
## Revision Summary

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision History</th>
<th>Revision Class</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>6/10/2011</td>
<td>0.1</td>
<td>New</td>
<td>Released new document.</td>
</tr>
<tr>
<td>8/5/2011</td>
<td>1.0</td>
<td>Major</td>
<td>Significantly changed the technical content.</td>
</tr>
<tr>
<td>10/7/2011</td>
<td>1.0</td>
<td>None</td>
<td>No changes to the meaning, language, or formatting of the technical content.</td>
</tr>
<tr>
<td>1/20/2012</td>
<td>2.0</td>
<td>Major</td>
<td>Significantly changed the technical content.</td>
</tr>
<tr>
<td>4/27/2012</td>
<td>3.0</td>
<td>Major</td>
<td>Significantly changed the technical content.</td>
</tr>
<tr>
<td>7/16/2012</td>
<td>3.0</td>
<td>None</td>
<td>No changes to the meaning, language, or formatting of the technical content.</td>
</tr>
<tr>
<td>10/8/2012</td>
<td>3.1</td>
<td>Minor</td>
<td>Clarified the meaning of the technical content.</td>
</tr>
<tr>
<td>2/11/2013</td>
<td>4.0</td>
<td>Major</td>
<td>Significantly changed the technical content.</td>
</tr>
<tr>
<td>7/26/2013</td>
<td>4.0</td>
<td>None</td>
<td>No changes to the meaning, language, or formatting of the technical content.</td>
</tr>
<tr>
<td>11/18/2013</td>
<td>4.1</td>
<td>Minor</td>
<td>Clarified the meaning of the technical content.</td>
</tr>
<tr>
<td>2/10/2014</td>
<td>4.1</td>
<td>None</td>
<td>No changes to the meaning, language, or formatting of the technical content.</td>
</tr>
<tr>
<td>4/30/2014</td>
<td>4.1</td>
<td>None</td>
<td>No changes to the meaning, language, or formatting of the technical content.</td>
</tr>
<tr>
<td>7/31/2014</td>
<td>5.0</td>
<td>Major</td>
<td>Significantly changed the technical content.</td>
</tr>
<tr>
<td>10/30/2014</td>
<td>6.0</td>
<td>Major</td>
<td>Significantly changed the technical content.</td>
</tr>
<tr>
<td>3/16/2015</td>
<td>7.0</td>
<td>Major</td>
<td>Significantly changed the technical content.</td>
</tr>
<tr>
<td>5/26/2015</td>
<td>8.0</td>
<td>Major</td>
<td>Significantly changed the technical content.</td>
</tr>
<tr>
<td>9/14/2015</td>
<td>8.0</td>
<td>None</td>
<td>No changes to the meaning, language, or formatting of the technical content.</td>
</tr>
<tr>
<td>6/13/2016</td>
<td>8.0</td>
<td>None</td>
<td>No changes to the meaning, language, or formatting of the technical content.</td>
</tr>
<tr>
<td>9/14/2016</td>
<td>8.0</td>
<td>None</td>
<td>No changes to the meaning, language, or formatting of the technical content.</td>
</tr>
<tr>
<td>10/17/2016</td>
<td>9.0</td>
<td>Major</td>
<td>Significantly changed the technical content.</td>
</tr>
<tr>
<td>12/15/2016</td>
<td>9.1</td>
<td>Minor</td>
<td>Clarified the meaning of the technical content.</td>
</tr>
<tr>
<td>7/24/2018</td>
<td>10.0</td>
<td>Major</td>
<td>Significantly changed the technical content.</td>
</tr>
<tr>
<td>10/1/2018</td>
<td>11.0</td>
<td>Major</td>
<td>Significantly changed the technical content.</td>
</tr>
<tr>
<td>12/11/2018</td>
<td>11.1</td>
<td>Minor</td>
<td>Clarified the meaning of the technical content.</td>
</tr>
<tr>
<td>4/22/2021</td>
<td>12.0</td>
<td>Major</td>
<td>Significantly changed the technical content.</td>
</tr>
<tr>
<td>8/17/2021</td>
<td>13.0</td>
<td>Major</td>
<td>Significantly changed the technical content.</td>
</tr>
<tr>
<td>Date</td>
<td>Revision History</td>
<td>Revision Class</td>
<td>Comments</td>
</tr>
<tr>
<td>-----------</td>
<td>------------------</td>
<td>----------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>2/15/2022</td>
<td>13.1</td>
<td>Minor</td>
<td>Clarified the meaning of the technical content.</td>
</tr>
</tbody>
</table>
# Table of Contents

1 Introduction .......................................................................................................................... 7
  1.1 Glossary ............................................................................................................................ 7
  1.2 References .......................................................................................................................... 10
      1.2.1 Normative References .............................................................................................. 10
      1.2.2 Informative References ............................................................................................ 10
  1.3 Overview .............................................................................................................................. 11
  1.4 Relationship to Other Protocols .......................................................................................... 11
  1.5 Prerequisites/Preconditions .................................................................................................. 12
  1.6 Applicability Statement ....................................................................................................... 12
  1.7 Versioning and Capability Negotiation ............................................................................... 12
  1.8 Vendor-Extensible Fields .................................................................................................... 13
  1.9 Standards Assignments ....................................................................................................... 13

2 Messages................................................................................................................................ 14
  2.1 Transport ............................................................................................................................. 14
  2.2 Common Data Types .............................................................................................................. 14
      2.2.1 Constant Value Definitions ........................................................................................ 15
          2.2.1.1 Permitted Property Type Values ............................................................................. 16
          2.2.1.2 Permitted Error Code Values ............................................................................... 17
          2.2.1.3 Display Type Values ........................................................................................... 18
          2.2.1.4 Default Language Code Identifier ....................................................................... 18
          2.2.1.5 Required Code Pages ......................................................................................... 19
          2.2.1.6 Unicode Comparison Flags ................................................................................. 19
          2.2.1.6.1 Comparison Flags .......................................................................................... 19
          2.2.1.7 Permanent Entry ID GUID ................................................................................... 21
          2.2.1.8 Positioning Minimal Entry IDs ............................................................................. 21
          2.2.1.9 Ambiguous Name Resolution Minimal Entry IDs ............................................... 21
          2.2.1.10 Table Sort Orders ............................................................................................... 22
          2.2.1.11 Retrieve Property Flags ..................................................................................... 22
          2.2.1.12 NspiGetSpecialTable Flags ............................................................................. 23
          2.2.1.13 NspiQueryColumns Flag .................................................................................. 23
          2.2.1.14 NspiGetTemplateInfo Flags ............................................................................. 23
          2.2.1.15 NspiModLinkAtt Flags ...................................................................................... 24
      2.2.2 Property Values ............................................................................................................. 24
          2.2.2.1 FlatUID_r Structure ............................................................................................ 24
          2.2.2.2 PropertyTagArray_r Structure ............................................................................. 24
          2.2.2.3 Binary_r Structure ............................................................................................... 24
          2.2.2.4 ShortArray_r Structure ....................................................................................... 25
          2.2.2.5 LongArray_r Structure ....................................................................................... 25
          2.2.2.6 StringArray_r Structure ....................................................................................... 25
          2.2.2.7 BinaryArray_r Structure ...................................................................................... 26
          2.2.2.8 FlatUIDArray_r Structure ................................................................................... 26
          2.2.2.9 WStringArray_r Structure ................................................................................... 26
          2.2.2.10 DateTimeArray_r Structure ............................................................................... 26
          2.2.2.11 PROP_VAL_ARRAY Structure .......................................................................... 27
          2.2.2.12 PropertyValue_r Structure ................................................................................. 28
      2.2.3 PropertyRow_r Structure .............................................................................................. 29
      2.2.4 PropertyRowSet_r Structure ........................................................................................ 29
      2.2.5 Restrictions ................................................................................................................ 29
          2.2.5.1 AndRestriction_r Restriction, OrRestriction_r Restriction ..................................... 30
          2.2.5.2 NotRestriction_r Restriction .............................................................................. 30
          2.2.5.3 ContentRestriction_r Restriction ....................................................................... 30
          2.2.5.4 PropertyRestriction_r Restriction ....................................................................... 31
          2.2.5.5 ExistRestriction_r Restriction ............................................................................ 31
          2.2.5.6 RestrictionUnion_r Restriction .......................................................................... 32
3 Protocol Details ........................................................................................................39
  3.1 Server Details ........................................................................................................39
    3.1.1 Abstract Data Model .........................................................................................39
    3.1.2 Timers ...............................................................................................................39
    3.1.3 Initialization .......................................................................................................39
    3.1.4 Message Processing Events and Sequencing Rules ........................................39
      3.1.4.1 NSPI Methods ..........................................................................................41
        3.1.4.1.1 NspiBind (Opnum 0) .............................................................................41
        3.1.4.1.2 NspiUnbind (Opnum 1) ........................................................................42
        3.1.4.1.3 NspiGetSpecialTable (Opnum 12) ......................................................43
        3.1.4.1.4 NspiUpdateStat (Opnum 2) .................................................................45
        3.1.4.1.5 NspiQueryColumns (Opnum 16) .........................................................46
        3.1.4.1.6 NspiGetPropList (Opnum 8) .................................................................47
        3.1.4.1.7 NspiGetProps (Opnum 9) .......................................................................48
        3.1.4.1.8 NspiQueryRows (Opnum 3) .................................................................50
        3.1.4.1.9 NspiSeekEntries (Opnum 4) .................................................................52
        3.1.4.1.10 NspiGetMatches (Opnum 5) .................................................................55
        3.1.4.1.11 NspiResortRestriction (Opnum 6) ......................................................58
        3.1.4.1.12 NspiCompareMIds (Opnum 10) .........................................................59
        3.1.4.1.13 NspiDNTOMId (Opnum 7) ...................................................................61
        3.1.4.1.14 NspiModProps (Opnum 11) .................................................................61
        3.1.4.1.15 NspiModLinkAtt (Opnum 14) ..............................................................62
        3.1.4.1.16 NspiResolveNames (Opnum 19) .........................................................64
        3.1.4.1.17 NspiResolveNamesW (Opnum 20) .....................................................65
        3.1.4.1.18 NspiGetTemplateInfo (Opnum 13) .....................................................66
      3.1.4.2 Required Properties .....................................................................................68
    3.1.4.3 String Handling .............................................................................................68
      3.1.4.3.1 Required Native Categorizations ..........................................................69
      3.1.4.3.2 Required Code Page Support ...............................................................69
      3.1.4.3.3 Conversion Rules for String Values Specified by the Server to the Client 69
      3.1.4.3.4 Conversion Rules for String Values Specified by the Client to the Server 70
      3.1.4.3.5 String Comparison ................................................................................71
        3.1.4.3.5.1 Unicode String Comparison ..........................................................71
        3.1.4.3.5.2 8-Bit String Comparison ..................................................................71
      3.1.4.3.6 String Sorting .........................................................................................71
    3.1.4.4 Tables ............................................................................................................72
      3.1.4.4.1 Status-Based Tables ..............................................................................72
      3.1.4.4.2 Explicit Tables .......................................................................................72
        3.1.4.4.2.1 Restriction-Based Explicit Tables .....................................................72
        3.1.4.4.2.2 Property Value-Based Explicit Tables .........................................72
      3.1.4.4.3 Specific Instantiations of Special Tables ..............................................72
        3.1.4.4.3.1 Address Book Hierarchy Table .......................................................72
        3.1.4.4.3.2 Address Creation Table ..................................................................73
    3.1.4.5 Positioning in a Table ....................................................................................73
      3.1.4.5.1 Absolute Positioning ..............................................................................73
1 Introduction

The Exchange Server Name Service Provider Interface (NSPI) Protocol provides a way for messaging clients to access and manipulate address data that is stored by a server. This protocol enables the client to use a single remote procedure call (RPC) interface and several interface methods to manipulate Address Book object data stored on the server.

Sections 1.5, 1.8, 1.9, 2, and 3 of this specification are normative. All other sections and examples in this specification are informative.

1.1 Glossary

This document uses the following terms:

- **address book**: A collection of Address Book objects, each of which are contained in any number of address lists.

- **address book container**: An Address Book object that describes an address list.

- **address book hierarchy table**: A collection of address book containers arranged in a hierarchy.

- **Address Book object**: An entity in an address book that contains a set of attributes, each attribute with a set of associated values.

- **address creation table**: A table containing information about the templates that an address book server supports for creating new email addresses.

- **address creation template**: A template that describes how to present a dialog to a messaging user along with a script describing how to construct a new email address from the user's response.

- **address list**: A collection of distinct Address Book objects.

- **ambiguous name resolution (ANR)**: A search algorithm that permits a client to search multiple naming-related attributes on objects by way of a single clause of the form "(anr=value)" in a Lightweight Directory Access Protocol (LDAP) search filter. This permits a client to query for an object when the client possesses some identifying material related to the object but does not know which attribute of the object contains that identifying material.

- **Augmented Backus-Naur Form (ABNF)**: A modified version of Backus-Naur Form (BNF), commonly used by Internet specifications. ABNF notation balances compactness and simplicity with reasonable representational power. ABNF differs from standard BNF in its definitions and uses of naming rules, repetition, alternatives, order-independence, and value ranges. For more information, see [RFC5234].

- **code page**: An ordered set of characters of a specific script in which a numerical index (code-point value) is associated with each character. Code pages are a means of providing support for character sets and keyboard layouts used in different countries. Devices such as the display and keyboard can be configured to use a specific code page and to switch from one code page (such as the United States) to another (such as Portugal) at the user's request.

- **display template**: A template that describes how to display or allow a user to modify information about an Address Book object.

- **distinguished name (DN)**: A name that uniquely identifies an object by using the relative distinguished name (RDN) for the object, and the names of container objects and domains that contain the object. The distinguished name (DN) identifies the object and its location in a tree.
**distribution list**: A collection of users, computers, contacts, or other groups that is used only for email distribution, and addressed as a single recipient.

**endpoint**: (1) A client that is on a network and is requesting access to a network access server (NAS).

(2) A network-specific address of a remote procedure call (RPC) server process for remote procedure calls. The actual name and type of the endpoint depends on the RPC protocol sequence that is being used. For example, for RPC over TCP (RPC Protocol Sequence ncacn_ip_tcp), an endpoint might be TCP port 1025. For RPC over Server Message Block (RPC Protocol Sequence ncacn_np), an endpoint might be the name of a named pipe. For more information, see [C706].

**entry ID**: See **EntryID**.

**EntryID**: A sequence of bytes that is used to identify and access an object.

**Global Address List (GAL)**: An address list that conceptually represents the default address list for an address book.

**globally unique identifier (GUID)**: A term used interchangeably with universally unique identifier (UUID) in Microsoft protocol technical documents (TDs). Interchanging the usage of these terms does not imply or require a specific algorithm or mechanism to generate the value. Specifically, the use of this term does not imply or require that the algorithms described in [RFC4122] or [C706] must be used for generating the GUID. See also universally unique identifier (UUID).

**Hypertext Transfer Protocol Secure (HTTPS)**: An extension of HTTP that securely encrypts and decrypts web page requests. In some older protocols, "Hypertext Transfer Protocol over Secure Sockets Layer" is still used (Secure Sockets Layer has been deprecated). For more information, see [SSL3] and [RFC5246].

**Interface Definition Language (IDL)**: The International Standards Organization (ISO) standard language for specifying the interface for remote procedure calls. For more information, see [C706] section 4.

**Kerberos**: An authentication system that enables two parties to exchange private information across an otherwise open network by assigning a unique key (called a ticket) to each user that logs on to the network and then embedding these tickets into messages sent by the users. For more information, see [MS-KILE].

**language code identifier (LCID)**: A 32-bit number that identifies the user interface human language dialect or variation that is supported by an application or a client computer.

**Lightweight Directory Access Protocol (LDAP)**: The primary access protocol for Active Directory. Lightweight Directory Access Protocol (LDAP) is an industry-standard protocol, established by the Internet Engineering Task Force (IETF), which allows users to query and update information in a directory service (DS), as described in [MS-ADTS]. The Lightweight Directory Access Protocol can be either version 2 [RFC1777] or version 3 [RFC3377].

**little-endian**: Multiple-byte values that are byte-ordered with the least significant byte stored in the memory location with the lowest address.

**Minimal Entry ID**: A property of an Address Book object that can be used to uniquely identify the object.

**name service provider interface (NSPI)**: A method of performing address-book-related operations on Active Directory.
Network Data Representation (NDR): A specification that defines a mapping from Interface Definition Language (IDL) data types onto octet streams. NDR also refers to the runtime environment that implements the mapping facilities (for example, data provided to NDR). For more information, see [MS-RPCE] and [C706] section 14.

NT LAN Manager (NTLM) Authentication Protocol: A protocol using a challenge-response mechanism for authentication in which clients are able to verify their identities without sending a password to the server. It consists of three messages, commonly referred to as Type 1 (negotiation), Type 2 (challenge) and Type 3 (authentication).

opnum: An operation number or numeric identifier that is used to identify a specific remote procedure call (RPC) method or a method in an interface. For more information, see [C706] section 12.5.2.12 or [MS-RPCE].

Permanent Entry ID: A property of an Address Book object that can be used to uniquely identify the object.

property type: A 16-bit quantity that specifies the data type of a property value.

remote procedure call (RPC): A communication protocol used primarily between client and server. The term has three definitions that are often used interchangeably: a runtime environment providing for communication facilities between computers (the RPC runtime); a set of request-and-response message exchanges between computers (the RPC exchange); and the single message from an RPC exchange (the RPC message). For more information, see [C706].

RPC protocol sequence: A character string that represents a valid combination of a remote procedure call (RPC) protocol, a network layer protocol, and a transport layer protocol, as described in [C706] and [MS-RPCE].

RPC transport: The underlying network services used by the remote procedure call (RPC) runtime for communications between network nodes. For more information, see [C706] section 2.

security provider: A pluggable security module that is specified by the protocol layer above the remote procedure call (RPC) layer, and will cause the RPC layer to use this module to secure messages in a communication session with the server. The security provider is sometimes referred to as an authentication service. For more information, see [C706] and [MS-RPCE].

shared folder: A folder for which a sharing relationship has been created to share items in the folder between two servers.

Transmission Control Protocol (TCP): A protocol used with the Internet Protocol (IP) to send data in the form of message units between computers over the Internet. TCP handles keeping track of the individual units of data (called packets) that a message is divided into for efficient routing through the Internet.

Unicode: A character encoding standard developed by the Unicode Consortium that represents almost all of the written languages of the world. The Unicode standard [UNICODE5.0.0/2007] provides three forms (UTF-8, UTF-16, and UTF-32) and seven schemes (UTF-8, UTF-16, UTF-16 BE, UTF-16 LE, UTF-32, UTF-32 LE, and UTF-32 BE).

universally unique identifier (UUID): A 128-bit value. UUIDs can be used for multiple purposes, from tagging objects with an extremely short lifetime, to reliably identifying very persistent objects in cross-process communication such as client and server interfaces, manager entry-point vectors, and RPC objects. UUIDs are highly likely to be unique. UUIDs are also known as globally unique identifiers (GUIDs) and these terms are used interchangeably in the Microsoft protocol technical documents (TDs). Interchanging the usage of these terms does not imply or require a specific algorithm or mechanism to generate the UUID. Specifically, the use of this term does not imply or require that the algorithms described in [RFC4122] or [C706] must be used for generating the UUID.
UTF-16LE: The Unicode Transformation Format - 16-bit, Little Endian encoding scheme. It is used to encode Unicode characters as a sequence of 16-bit codes, each encoded as two 8-bit bytes with the least-significant byte first.

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as defined in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

Links to a document in the Microsoft Open Specifications library point to the correct section in the most recently published version of the referenced document. However, because individual documents in the library are not updated at the same time, the section numbers in the documents may not match. You can confirm the correct section numbering by checking the Errata.

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.


Note Registration is required to download the document.

[MS-DTYP] Microsoft Corporation, "Windows Data Types".
[MS-LCID] Microsoft Corporation, "Windows Language Code Identifier (LCID) Reference".
[MS-OXCDATA] Microsoft Corporation, "Data Structures".
[MS-OXCFOLD] Microsoft Corporation, "Folder Object Protocol".
[MS-OXOCNCTC] Microsoft Corporation, "Contact Object Protocol".
[MS-RPCE] Microsoft Corporation, "Remote Procedure Call Protocol Extensions".

1.2.2 Informative References

[MS-NSPI] Microsoft Corporation, "Name Service Provider Interface (NSPI) Protocol".
1.3 Overview

Messaging clients that implement a browsable address book need a way to communicate with an address data store in order to access and manipulate that data. This protocol enables communication between a messaging client and a data store.

This protocol is a protocol layer that uses the remote procedure call (RPC) protocol as a transport, with a series of interface methods as described in this document, that clients can use to communicate with a server. The server will use Lightweight Directory Access Protocol (LDAP) and NSPI to retrieve data that is returned to the client.

The following figure shows a graphical representation of a typical communication sequence between a messaging client and a server.

![Graphical representation of a typical communication sequence between a messaging client and a server.]

Figure 1: Exchange Server NSPI Protocol message sequence

1.4 Relationship to Other Protocols

The Exchange Server NSPI Protocol depends on the following protocols:

- The Remote Procedure Call (RPC) Protocol, as described in [C706] and [MS-RPCE], as a transport.
- The Kerberos Network Authentication Service (V5), as described in [MS-KILE], [RFC1510], and [RFC4120] for client authentication.
- The NT LAN Manager (NTLM) Authentication Protocol, as described in [MS-NLMP], for client authentication.
- The Address Book Object Protocol, as described in [MS-OXOABK], for property definitions.
- The Address Book User Interface Templates Protocol, as described in [MS-OXOABKT], for the definition of address book templates.
- The Lightweight Directory Access Protocol (LDAP), as described in [RFC4511].
- The Name Service Provider Interface (NSPI) Protocol, as described in [MS-NSPI].
For conceptual background information and overviews of the relationships and interactions between this and other protocols, see [MS-OXPROTO].

1.5 Prerequisites/Preconditions

The client implementation has to have the network address of the server. This network address satisfies the requirements of a network address for the underlying transport of remote procedure call (RPC). This allows the client to initiate communication with the server by using the RPC Protocol.

This protocol uses security information as specified in [MS-RPCE]. The client and Exchange NSPI server are required to share one or both of the NT LAN Manager (NTLM) Authentication Protocol or Kerberos security providers in common for the RPC transport. Additionally, the server is required to register the negotiation security provider.

The protocol does not require mutual authentication. The client and Exchange NSPI server use an authentication mechanism that is capable of authenticating the client to the server. The protocol does not require that the client be capable of authenticating the server.

The credentials of the client have to be recognized by the server. These credentials are obtained from the shared security provider. The mechanism for obtaining these credentials is specific to the protocol of the security provider that is used.

The server has to have determined any local policies as described in sections 2, 3, and 5. This allows the server to provide consistent behavior for all communications in the protocol.

The server has to be configured to support the required code pages and language code identifiers (LCIDs), as described in sections 2.2.1.4 and 2.2.1.5. This allows the server to provide the minimal required string conversions and sort orders.

The server has to be started and fully initialized before the protocol can start.

1.6 Applicability Statement

The Exchange Server NSPI Protocol is appropriate for messaging clients that implement online access to address books for browsing and viewing of Address Book objects that are stored in a data store.

1.7 Versioning and Capability Negotiation

This document covers versioning issues in the following areas:

- **Supported Transports:** This protocol uses multiple RPC protocol sequences, as specified in section 2.1.
- **Protocol Versions:** This protocol has a single interface version. This version is defined in section 2.1.
- **Security and Authentication Methods:** This protocol supports the NTLM and Kerberos authentication methods.
- **Localization:** This protocol passes text strings in various methods. Localization considerations for such strings are specified in section 3.1.4.3.
- **Capability Negotiation:** The Exchange Server NSPI Protocol does not support negotiation. There is only one interface version.
1.8 Vendor-Extensible Fields

None.

1.9 Standards Assignments

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Reference</th>
</tr>
</thead>
</table>
2 Messages

The following sections specify transport methods of Exchange Server NSPI Protocol messages and common Exchange Server NSPI Protocol data types.

Unless otherwise specified, all numeric values in this specification are in little-endian format.

Unless otherwise specified, all Unicode string representations are in UTF-16LE format.

2.1 Transport

All remote procedure call (RPC) protocols use RPC dynamic endpoints (2) and well-known endpoints (2), as specified in [C706].

The Exchange Server NSPI Protocol uses the following RPC protocol sequences:

- RPC over HTTPS
- RPC over TCP<1>

The protocol allows a server to be configured to use a specific port for RPC over TCP. The mechanism for configuring a server to use a specific port is not constrained by the Exchange Server NSPI Protocol. The mechanism for a client to discover this configured TCP port is not constrained by the Exchange Server NSPI Protocol.

For the network protocol sequence RPC over HTTPS, this protocol MUST use the well-known endpoint 6004. For RPC over TCP, this protocol can use RPC dynamic endpoints, as defined in Part 4 of [C706].

This protocol MUST use the UUID F5CC5A18-4264-101A-8C59-08002B2F8426. The protocol MUST use the RPC version number 56.0.

The protocol uses the underlying RPC protocol to retrieve the identity of the client that made the method call, as specified in [MS-RPCE]. The server MAY use this identity to perform access checks, as described in section 5 of this document.

The server MAY enforce limits on the maximum RPC packet size that it will accept.

2.2 Common Data Types

This protocol enables the ms_union extension, as specified in [MS-RPCE].

This protocol requests that the RPC runtime, via the strict_context_handle attribute, rejects the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE].

In addition to the RPC base types and definitions specified in [C706] and [MS-RPCE], the Exchange Server NSPI Protocol uses additional data types.

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

<table>
<thead>
<tr>
<th>Data type name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FlatUID_r</td>
<td>Byte order specified GUIDs</td>
</tr>
<tr>
<td>PropertyTagArray_r</td>
<td>Property value structure</td>
</tr>
<tr>
<td>Binary_r</td>
<td>Property value structure</td>
</tr>
<tr>
<td>ShortArray_r</td>
<td>Property value structure</td>
</tr>
<tr>
<td>Data type name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>LongArray_r</td>
<td>Property value structure</td>
</tr>
<tr>
<td>StringArray_r</td>
<td>Property value structure</td>
</tr>
<tr>
<td>BinaryArray_r</td>
<td>Property value structure</td>
</tr>
<tr>
<td>FlatUIDArray_r</td>
<td>Property value structure</td>
</tr>
<tr>
<td>WStringArray_r</td>
<td>Property value structure</td>
</tr>
<tr>
<td>DateTimeArray_r</td>
<td>Property value structure</td>
</tr>
<tr>
<td>PROP_VAL_UNION</td>
<td>Property value structure</td>
</tr>
<tr>
<td>PropertyValue_r</td>
<td>Property value structure</td>
</tr>
<tr>
<td>PropertyRow_r</td>
<td>Table row structure</td>
</tr>
<tr>
<td>PropertyRowSet_r</td>
<td>Table rows structure</td>
</tr>
<tr>
<td>AndRestriction_r</td>
<td>Table restriction structure</td>
</tr>
<tr>
<td>OrRestriction_r</td>
<td>Table restriction structure</td>
</tr>
<tr>
<td>NotRestriction_r</td>
<td>Table restriction structure</td>
</tr>
<tr>
<td>ContentRestriction_r</td>
<td>Table restriction structure</td>
</tr>
<tr>
<td>PropertyRestriction_r</td>
<td>Table restriction structure</td>
</tr>
<tr>
<td>ExistRestriction_r</td>
<td>Table restriction structure</td>
</tr>
<tr>
<td>RestrictionUnion_r</td>
<td>Table restriction structure</td>
</tr>
<tr>
<td>Restriction_r</td>
<td>Table restriction structure</td>
</tr>
<tr>
<td>PropertyName_r</td>
<td>Address book property specifier</td>
</tr>
<tr>
<td>StringsArray_r</td>
<td>Collection of 8-bit character strings</td>
</tr>
<tr>
<td>WStringsArray_r</td>
<td>Collection of Unicode strings</td>
</tr>
<tr>
<td>STAT</td>
<td>Table status structure</td>
</tr>
<tr>
<td>MinimalEntryID</td>
<td>Address Book object identification</td>
</tr>
<tr>
<td>EphemeralEntryID</td>
<td>Address Book object identification</td>
</tr>
<tr>
<td>PermanentEntryID</td>
<td>Address Book object identification</td>
</tr>
<tr>
<td>NSPI_HANDLE</td>
<td>RPC context handle</td>
</tr>
</tbody>
</table>

### 2.2.1 Constant Value Definitions

This section defines common values that are used in multiple messages.
2.2.1.1 Permitted Property Type Values

The property type values specified in this section are used to specify property types. They appear in various places in the Exchange Server NSPI Protocol. All Exchange NSPI servers and clients MUST recognize and be capable of accepting and returning these property types. Values that represent property types are defined in [MS-OXCDATA].

The values specified in [MS-OXCDATA] are 16-bit integers. The Exchange Server NSPI Protocol uses the same numeric values but expressed as 32-bit integers. The high-order 16 bits of the 32-bit representation that is used by the Exchange Server NSPI Protocol are always 0x0000. The following table lists the permitted values for the Exchange Server NSPI Protocol.

<table>
<thead>
<tr>
<th>Value name</th>
<th>Value as defined in [MS-OXCDATA]</th>
<th>Value as used in the Exchange Server NSPI Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>PtypInteger32</td>
<td>0x0003</td>
<td>0x00000003</td>
</tr>
<tr>
<td>PtypBoolean</td>
<td>0x000B</td>
<td>0x0000000B</td>
</tr>
<tr>
<td>PtypString8</td>
<td>0x001E</td>
<td>0x0000001E</td>
</tr>
<tr>
<td>PtypBinary</td>
<td>0x0102</td>
<td>0x00000102</td>
</tr>
<tr>
<td>PtypString</td>
<td>0x001F</td>
<td>0x0000010F</td>
</tr>
<tr>
<td>PtypTime</td>
<td>0x0040</td>
<td>0x00000040</td>
</tr>
<tr>
<td>PtypErrorCode</td>
<td>0x000A</td>
<td>0x0000000A</td>
</tr>
<tr>
<td>PtypMultipleString8</td>
<td>0x101E</td>
<td>0x0000101E</td>
</tr>
<tr>
<td>PtypMultipleBinary</td>
<td>0x1102</td>
<td>0x00001102</td>
</tr>
<tr>
<td>PtypMultipleString</td>
<td>0x101F</td>
<td>0x0000101F</td>
</tr>
</tbody>
</table>

In addition, all Exchange NSPI servers and clients MUST recognize and be capable of accepting and returning the property types that are listed in the following table.

<table>
<thead>
<tr>
<th>Property type name and value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PtypEmbeddedTable</td>
<td>Single 32-bit value, referencing an address list.</td>
</tr>
<tr>
<td>PtypNull</td>
<td>Clients MUST NOT specify this property type in any method's input parameters. The server MUST specify this property type in any method's output parameters to indicate that a property has a value that cannot be expressed in the Exchange Server NSPI Protocol.</td>
</tr>
<tr>
<td>PtypUnspecified</td>
<td>Clients specify this property type in a method's input parameter to indicate that the client will accept any property type the server chooses when</td>
</tr>
<tr>
<td>Property type name and value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>0x00000000</td>
<td>returning propvalues. Servers MUST NOT specify this property type in any method's output parameters.</td>
</tr>
</tbody>
</table>

All clients and servers MUST NOT use any other property types.

### 2.2.1.2 Permitted Error Code Values

The error code values listed in this section are used to specify status from an NSPI method. They appear as return codes from NSPI methods and as values of properties with property type PtypErrorCode ([MS-OXCDATA] section 2.11.1). All Exchange NSPI servers MUST recognize and be capable of accepting and returning these error codes. The values that represent the error codes are defined in [MS-OXCDATA] section 2.4. The following are the permitted error code values for the Exchange Server NSPI Protocol:

- Success
- UnbindSuccess
- UnbindFailure
- ErrorsReturned
- GeneralFailure
- NotSupported
- InvalidObject
- OutOfResources
- NotFound
- LogonFailed
- TooComplex
- InvalidCodepage
- InvalidLocale
- TableTooBig
- InvalidBookmark
- AccessDenied
- NotEnoughMemory
- InvalidParameter

All clients and servers MUST NOT use any other error codes.

### 2.2.1.3 Display Type Values

The values listed in this section are used to specify display types. They appear in various places in the Exchange Server NSPI Protocol as object properties and as part of EntryIDs. Except where otherwise specified in the following table, all Exchange NSPI servers MUST recognize and be capable of accepting...
and returning these display types. The following table lists the permitted display type values for the Exchange Server NSPI Protocol.

<table>
<thead>
<tr>
<th>Display type name and value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DT_MAILUSER 0x00000000</td>
<td>A typical messaging user.</td>
</tr>
<tr>
<td>DT_DISTLIST 0x00000001</td>
<td>A distribution list.</td>
</tr>
<tr>
<td>DT_FORUM 0x00000002</td>
<td>A forum, such as a bulletin board service or a public or shared folder.</td>
</tr>
<tr>
<td>DT_AGENT 0x00000003</td>
<td>An automated agent, such as Quote-Of-The-Day or a weather chart display.</td>
</tr>
<tr>
<td>DT_ORGANIZATION 0x00000004</td>
<td>An Address Book object defined for a large group, such as helpdesk, accounting, coordinator, or department. Department objects usually have this display type. An Exchange NSPI server MUST NOT return display type.</td>
</tr>
<tr>
<td>DT_PRIVATE_DISTLIST 0x00000005</td>
<td>A private, personally administered distribution list.</td>
</tr>
<tr>
<td>DT_REMOTE_MAILUSER 0x00000006</td>
<td>An Address Book object known to be from a foreign or remote messaging system.</td>
</tr>
<tr>
<td>DT_CONTAINER 0x00000100</td>
<td>An address book hierarchy table container. An Exchange NSPI server MUST NOT return this display type except as part of an EntryID of an object in the address book hierarchy table.</td>
</tr>
<tr>
<td>DT_TEMPLATE 0x00000101</td>
<td>A display template object. An Exchange NSPI server MUST NOT return this display type.</td>
</tr>
<tr>
<td>DT_ADDRESS_TEMPLATE 0x00000102</td>
<td>An address creation template. An Exchange NSPI server MUST NOT return this display type except as part of an EntryID of an object in the Address Creation Table.</td>
</tr>
<tr>
<td>DT_SEARCH 0x00000200</td>
<td>A search template. An Exchange NSPI server MUST NOT return this display type.</td>
</tr>
</tbody>
</table>

All clients and servers MUST NOT use any other display types.

2.2.1.4 Default Language Code Identifier

The language code identifier (LCID) specified in this section is associated with the minimal required sort order for Unicode strings. It appears in input parameters to Exchange Server NSPI Protocol methods. It affects Exchange NSPI server string handling, as specified in section 3.1.4.3. The following table lists and describes the default LCID for this protocol.

<table>
<thead>
<tr>
<th>Default LCID name and value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSPI_DEFAULT_LOCALE 0x00000409</td>
<td>Represents the default LCID that is used for comparison of Unicode string representations.</td>
</tr>
</tbody>
</table>

[MS-OXNSPI] - v20220215
Exchange Server Name Service Provider Interface (NSPI) Protocol
Copyright © 2022 Microsoft Corporation
Release: February 15, 2022
2.2.1.5 Required Code Pages

The required code pages listed in this section are associated with the string handling in the Exchange Server NSPI Protocol, and they appear in input parameters to methods in the Exchange Server NSPI Protocol. They affect Exchange NSPI server string handling, as specified in section 3.1.4.3. The following table lists the required code pages.

<table>
<thead>
<tr>
<th>Code page name and value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CP_TELETEX 0x00004F25</td>
<td>Represents the Teletex code page.</td>
</tr>
<tr>
<td>CP_WINUNICODE 0x000004B0</td>
<td>Represents the Unicode code page.</td>
</tr>
</tbody>
</table>

2.2.1.6 Unicode Comparison Flags

These values are associated with string handling in the Exchange Server NSPI Protocol. These values are defined in terms of definitions, as specified in section 2.2.1.6.1. The server uses these flags to modify the behavior of comparisons of Unicode string representations, as detailed in section 3.1.4.3.

<table>
<thead>
<tr>
<th>Name and value</th>
<th>Description</th>
</tr>
</thead>
</table>
| NSPI_DEFAULT_LOCALE_COMPARE_FLAGS (NORM_IGNORECASE | \ 
| NORM_IGNOREKANATYPE | \ 
| NORM_IGNORENONSPACE | \ 
| NORM_IGNOREWIDTH | \ 
| SORT_STRINGSORT) | Flags used when comparing Unicode strings in the language code identifier (LCID) represented by NSPI_DEFAULT_LOCALE. The comparison flag values are defined in section 2.2.1.6.1. |
| NSPI_NON_DEFAULT_LOCALE_COMPARE_FLAGS (NORM_IGNORECASE | \ 
| NORM_IGNOREKANATYPE | \ 
| NORM_IGNORENONSPACE | \ 
| NORM_IGNOREWIDTH | \ 
| NORM_IGNORESYMBOLS | \ 
| SORT_STRINGSORT) | Flags used when comparing Unicode strings in any LCID except the LCID represented by NSPI_DEFAULT_LOCALE. The comparison flag values are defined in section 2.2.1.6.1. |

2.2.1.6.1 Comparison Flags

The following defines the bit settings and meaning of the bits used by the Unicode comparison flags. The flags are presented in big-endian byte order.

```
0 1 2 3 4 5 6 7 8 9 1 0 1 2 3 4 5 6 7 8 9 2 0 1 2 3 4 5 6 7 8 9 3 0 1
V U T S R Q P O N M L K J I H G F E D C B A 9 8 7 6 5 4 3 2 1 0

V (1 bit): NORM_IGNORECASE: Ignore Case.
U (1 bit): NORM_IGNORENONSPACE: Ignore non-spacing characters.
```
T (1 bit): NORM IGNORESYMBOLS: Ignore symbols.
S (1 bit): Unused.
R (1 bit): Unused.
Q (1 bit): Unused.
P (1 bit): Unused.
O (1 bit): Unused.
N (1 bit): Unused.
M (1 bit): Unused.
L (1 bit): Unused.
K (1 bit): Unused.
J (1 bit): SORT STRINGSORT: Treat punctuation the same as symbols.
I (1 bit): Unused.
H (1 bit): Unused.
G (1 bit): Unused.
F (1 bit): NORM IGNOREKANJIATYPE: Do not differentiate between hiragana and katakana characters. Corresponding hiragana and katakana characters compare as equal.
E (1 bit): NORM IGNOREWIDTH: Ignore the difference between half-width and full-width characters.
D (1 bit): Unused.
C (1 bit): Unused.
B (1 bit): Unused.
A (1 bit): Unused.
9 (1 bit): Unused.
8 (1 bit): Unused.
7 (1 bit): Unused.
6 (1 bit): Unused.
5 (1 bit): Unused.
4 (1 bit): Unused.
3 (1 bit): Unused.
2 (1 bit): Unused.
1 (1 bit): Unused.
0 (1 bit): Unused.
2.2.1.7 Permanent Entry ID GUID

The following table lists the **Permanent Entry ID** that is associated with the Exchange Server NSPI Protocol.

<table>
<thead>
<tr>
<th>Permanent Entry ID name and value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GUID_NSPI {0xDC, 0xA7, 0x40, 0xC8, 0xC0, 0x42, 0x10, 0x1A, 0xB4, 0xB9, 0x08, 0x00, 0x2B, 0x2F, 0xE1, 0x82}</td>
<td>Represents the Exchange Server NSPI Protocol in Permanent Entry IDs.</td>
</tr>
</tbody>
</table>

2.2.1.8 Positioning Minimal Entry IDs

The positioning **Minimal Entry IDs** are used to specify objects in the **address book** as a function of their positions in tables. They appear as Minimal Entry IDs in the **CurrentRec** field of the **STAT** structure, as specified in section 2.2.8. The following table lists the possible values.

<table>
<thead>
<tr>
<th>Minimal Entry ID name and value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MID_BEGINNING_OF_TABLE 0x00000000</td>
<td>Specifies the position before the first row in the current address book container.</td>
</tr>
<tr>
<td>MID_END_OF_TABLE 0x00000002</td>
<td>Specifies the position after the last row in the current address book container.</td>
</tr>
<tr>
<td>MID_CURRENT 0x00000001</td>
<td>Specifies the current position in a table. This Minimal Entry ID is only valid in the <strong>NspiUpdateStat</strong> method, as specified in section 3.1.4.1.4. In all other cases, it is an invalid Minimal Entry ID, guaranteed to not specify any object in the address book.</td>
</tr>
</tbody>
</table>

2.2.1.9 Ambiguous Name Resolution Minimal Entry IDs

**Ambiguous name resolution (ANR) Minimal Entry IDs** are used to specify the outcome of the ANR process. They appear in return data from the **NspiResolveNames** method, as specified in section 3.1.4.1.16, and the **NspiResolveNamesw** method, as specified in section 3.1.4.1.17. The following table lists the possible values.

<table>
<thead>
<tr>
<th>Minimal Entry ID name and value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MID_UNRESOLVED 0x00000000</td>
<td>The ANR process was unable to map a string to any objects in the address book.</td>
</tr>
<tr>
<td>MID_AMBIGUOUS 0x00000001</td>
<td>The ANR process mapped a string to multiple objects in the address book.</td>
</tr>
<tr>
<td>MID_RESOLVED 0x00000002</td>
<td>The ANR process mapped a string to a single object in the address book.</td>
</tr>
</tbody>
</table>
2.2.1.10  Table Sort Orders

The following table lists the values that are used to specify specific sort orders for tables. These values appear in the **SortType** field of the **STAT** data structure, as specified in section 2.2.8.

<table>
<thead>
<tr>
<th>Sort type name and value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SortTypeDisplayName</td>
<td>The table is sorted ascending on the <strong>PidTagDisplayName</strong> property, as specified in [MS-OXCFOLD] section 2.2.2.2.5. All Exchange NSPI servers MUST support this sort order for at least one <strong>LCID</strong>.</td>
</tr>
<tr>
<td>SortTypePhoneticDisplayName</td>
<td>The table is sorted ascending on the <strong>PidTagAddressBookPhoneticDisplayName</strong> property, as specified in [MS-OXOABK] section 2.2.3.9. Exchange NSPI servers SHOULD support this sort order.</td>
</tr>
<tr>
<td>SortTypeDisplayName_RO</td>
<td>The table is sorted ascending on the <strong>PidTagDisplayName</strong> property. The client MUST set this value only when using the <strong>NspiGetMatches</strong> method, as specified in section 3.1.4.1.10, to open a non-writable table on an object-valued property.</td>
</tr>
<tr>
<td>SortTypeDisplayName_W</td>
<td>The table is sorted ascending on the <strong>PidTagDisplayName</strong> property. The client MUST set this value only when using the <strong>NspiGetMatches</strong> method to open a writable table on an object-valued property.</td>
</tr>
</tbody>
</table>

2.2.1.11  Retrieve Property Flags

The following table lists the property flag values that are used to specify optional behavior to a server. They appear as bit flags in methods that return property values to the client (**NspiGetPropList**, **NspiGetProps**, and **NspiQueryRows**).

<table>
<thead>
<tr>
<th>Property flag name and value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fSkipObjects</td>
<td>Client requires that the server MUST NOT include proptags with the <strong>PtypEmbeddedTable</strong> property type in any lists of proptags that the server creates on behalf of the client.</td>
</tr>
<tr>
<td>fEphID</td>
<td>Client requires that the server MUST return <strong>Entry ID</strong> values in Ephemeral Entry ID form.</td>
</tr>
</tbody>
</table>

2.2.1.12  NspiGetSpecialTable Flags

**NspiGetSpecialTable** flag values are used to specify optional behavior to a server. They appear as bit flags in the **NspiGetSpecialTable** method, as specified in section 3.1.4.1.3. The following table lists the possible values.

<table>
<thead>
<tr>
<th>Flag name and value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NspiAddressCreationTemplates</td>
<td>Specifies that the server MUST return the table of the available address creation templates. Specifying this flag causes the server to ignore the <strong>NspiUnicodeStrings</strong> flag.</td>
</tr>
<tr>
<td>Flag name and value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>NspiUnicodeStrings</strong></td>
<td>Specifies that the server MUST return all strings as <strong>Unicode</strong> representations rather than as multibyte strings in the client’s <strong>code page</strong>.</td>
</tr>
</tbody>
</table>

### 2.2.1.13 NspiQueryColumns Flag

The **NspiQueryColumns** flag value is used to specify optional behavior to a server. It appears as a bit flag in the **NspiQueryColumns** method. The following table lists the value for this flag.

<table>
<thead>
<tr>
<th>Flag name and value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NspiUnicodeProptypes</strong></td>
<td>Specifies that the server MUST return all proptags that specify values with string representations as having the <strong>PtypString property type</strong>. If the <strong>NspiUnicodeProptypes</strong> flag is not set, the server MUST return all proptags specifying values with string representations as having the <strong>PtypString8 property type</strong>.</td>
</tr>
</tbody>
</table>

### 2.2.1.14 NspiGetTemplateInfo Flags

The **NspiGetTemplateInfo** flag values are used to specify optional behavior to a server. They appear as bit flags in the **NspiGetTemplateInfo** method. The following table lists the possible values.

<table>
<thead>
<tr>
<th>Flag name and value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TI_TEMPLATE</strong></td>
<td>Specifies that the server is to return the value that represents a template.</td>
</tr>
<tr>
<td>0x00000001</td>
<td></td>
</tr>
<tr>
<td><strong>TI_SCRIPT</strong></td>
<td>Specifies that the server is to return the value of the script that is associated with a template.</td>
</tr>
<tr>
<td>0x00000004</td>
<td></td>
</tr>
<tr>
<td><strong>TI_EMT</strong></td>
<td>Specifies that the server is to return the e-mail type that is associated with a template.</td>
</tr>
<tr>
<td>0x00000010</td>
<td></td>
</tr>
<tr>
<td><strong>TI_HELPFILE_NAME</strong></td>
<td>Specifies that the server is to return the name of the help file that is associated with a template.</td>
</tr>
<tr>
<td>0x00000020</td>
<td></td>
</tr>
<tr>
<td><strong>TI_HELPFILE_CONTENTS</strong></td>
<td>Specifies that the server is to return the contents of the help file that is associated with a template.</td>
</tr>
<tr>
<td>0x00000040</td>
<td></td>
</tr>
</tbody>
</table>

### 2.2.1.15 NspiModLinkAtt Flags

The **NspiModLinkAtt** flag value is used to specify optional behavior to a server. It appears as a bit flag in the **NspiModLinkAtt** method. The following table lists the value of the flag.

<table>
<thead>
<tr>
<th>Flag name and Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>fDelete</strong></td>
<td>Specifies that the server is to remove values when modifying. If the <strong>fDelete</strong> flag is not set, the server adds values when modifying.</td>
</tr>
<tr>
<td>0x00000001</td>
<td></td>
</tr>
</tbody>
</table>
2.2.2 Property Values

The following structures are used to represent specific property values.

2.2.2.1 FlatUID_r Structure

The FlatUID_r structure is an encoding of the FlatUID data structure defined in [MS-OXCDATA] section 2.5.2. The semantic meaning is unchanged from the FlatUID data structure.

```c
typedef struct {
    BYTE ab[16];
} FlatUID_r;
```

ab: Encodes the ordered bytes of the FlatUID data structure.

2.2.2.2 PropertyTagArray_r Structure

The PropertyTagArray_r structure is an encoding of the PropertyTagArray_r data structure defined in [MS-OXCDATA] section 2.10.2. The permissible number of proptag values in the PropertyTagArray_r structure exceeds that of the PropertyTagArray_r data structure. The semantic meaning is otherwise unchanged from the PropertyTagArray_r data structure.

```c
typedef struct PropertyTagArray_r {
    DWORD cValues;
    [size_is(cValues+1), length_is(cValues)]
        DWORD aulPropTag[];
} PropertyTagArray_r;
```

cValues: Encodes the Count field of the PropertyTagArray_r data structure.

aulPropTag: Encodes the PropertyTags field of the PropertyTagArray_r data structure.

2.2.2.3 Binary_r Structure

The Binary_r structure encodes an array of uninterpreted bytes.

```c
typedef struct Binary_r {
    [range(0,2097152)] DWORD cb;
    [size_is(cb)] BYTE* lpb;
} Binary_r;
```

cb: The number of uninterpreted bytes represented in this structure. This value MUST NOT exceed 2,097,152.

lpb: The uninterpreted bytes.

2.2.2.4 ShortArray_r Structure

The ShortArray_r structure encodes an array of 16-bit integers.

---

[MS-OXNSPI] - v20220215
Exchange Server Name Service Provider Interface (NSPI) Protocol
Copyright © 2022 Microsoft Corporation
Release: February 15, 2022
typedef struct ShortArray_r {
    [range(0,100000)] DWORD cValues;
    [size_is(cValues)] short int* lpi;
} ShortArray_r;

cValues: The number of 16-bit integer values represented in the ShortArray_r structure. This value MUST NOT exceed 100,000.
lpi: The 16-bit integer values.

2.2.2.5 LongArray_r Structure
The LongArray_r structure encodes an array of 32-bit integers.

typedef struct _LongArray_r {
    [range(0,100000)] DWORD cValues;
    [size_is(cValues)] long* lpl;
} LongArray_r;

cValues: The number of 32-bit integers represented in this structure. This value MUST NOT exceed 100,000.
lpl: The 32-bit integer values.

2.2.2.6 StringArray_r Structure
The StringArray_r structure encodes an array of references to 8-bit character strings.

typedef struct _StringArray_r {
    [range(0,100000)] DWORD cValues;
    [string, size_is(cValues)] char** lppszA;
} StringArray_r;

cValues: The number of 8-bit character string references represented in the StringArray_r structure. This value MUST NOT exceed 100,000.
lppszA: The 8-bit character string references. The strings referred to are NULL-terminated.

2.2.2.7 BinaryArray_r Structure
The BinaryArray_r structure is an array of Binary_r data structures.

typedef struct _BinaryArray_r {
    [range(0,100000)] DWORD cValues;
    [size_is(cValues)] Binary_r* lpbin;
} BinaryArray_r;

cValues: The number of Binary_r data structures represented in the BinaryArray_r structure. This value MUST NOT exceed 100,000.
lpbin: The Binary_r data structures.
2.2.2.8 FlatUIDArray_r Structure

The FlatUIDArray_r structure encodes an array of FlatUID_r data structures.

typedef struct _FlatUIDArray_r {
    [range(0,100000)] DWORD cValues;
    [size_is(cValues)] FlatUID_r** lpguid;
} FlatUIDArray_r;

cValues: The number of FlatUID_r structures represented in the FlatUIDArray_r structure. This value MUST NOT exceed 100,000.
lpguid: The FlatUID_r data structures.

2.2.2.9 WStringArray_r Structure

The WStringArray_r structure encodes an array of references to Unicode strings.

typedef struct _WStringArray_r {
    [range(0,100000)] DWORD cValues;
    [string, size_is(cValues)] wchar_t** lppszW;
} WStringArray_r;

cValues: The number of Unicode character string references represented in the WStringArray_r structure. This value MUST NOT exceed 100,000.
lppszW: The Unicode character string references. The strings referred to are NULL-terminated.

2.2.2.10 DateTimeArray_r Structure

The DateTimeArray_r structure encodes an array of FILETIME structures.

typedef struct _DateTimeArray_r {
    [range(0,100000)] DWORD cValues;
    [size_is(cValues)] FILETIME* lpft;
} DateTimeArray_r;

cValues: The number of FILETIME data structures represented in the DateTimeArray_r structure. This value MUST NOT exceed 100,000.
lpft: The FILETIME data structures.

2.2.2.11 PROP_VAL_UNION Structure

The PROP_VAL_UNION structure encodes a single instance of any type of property value. It is an aggregation data structure, allowing a single parameter to an NSPI method to contain any type of property value.

typedef
t[switch_type(long)]
union _PV_r {
    [case(0x00000002)] short int i;
i: PROP_VAL_UNION contains an encoding of the value of a property that can contain a single 16-bit integer value.

l: PROP_VAL_UNION contains an encoding of the value of a property that can contain a single 32-bit integer value.

b: PROP_VAL_UNION contains an encoding of the value of a property that can contain a single Boolean value. The client and server MUST NOT set this to values other than 1 or 0.

lpszA: PROP_VAL_UNION contains an encoding of the value of a property that can contain a single 8-bit character string value. This value is NULL-terminated.

bin: PROP_VAL_UNION contains an encoding of the value of a property that can contain a single binary data value. The number of bytes that can be encoded in this structure is 2,097,152.

lpszW: PROP_VAL_UNION contains an encoding of the value of a property that can contain a single Unicode string value. This value is NULL-terminated.

lpguid: PROP_VAL_UNION contains an encoding of the value of a property that can contain a single GUID value. The value is encoded as a FlatUID_r data structure.

ft: PROP_VAL_UNION contains an encoding of the value of a property that can contain a single 64-bit integer value. The value is encoded as a FILETIME structure.

er: PROP_VAL_UNION contains an encoding of the value of a property that can contain a single PtypErrorCode value.

MVi: PROP_VAL_UNION contains an encoding of the values of a property that can contain multiple 16-bit integer values. The number of values that can be encoded in this structure is 100,000.
**MVl: PROP_VAL_UNION** contains an encoding of the values of a property that can contain multiple 32-bit integer values. The number of values that can be encoded in this structure is 100,000.

**MVszA: PROP_VAL_UNION** contains an encoding of the values of a property that can contain multiple 8-bit character string values. These string values are NULL-terminated. The number of values that can be encoded in this structure is 100,000.

**MVbin: PROP_VAL_UNION** contains an encoding of the values of a property that can contain multiple binary data values. The number of bytes that can be encoded in each value of this structure is 2,097,152. The number of values that can be encoded in this structure is 100,000.

**MVguid: PROP_VAL_UNION** contains an encoding of the values of a property that can contain multiple GUID values. The values are encoded as FlatUID_r data structures. The number of values that can be encoded in this structure is 100,000.

**MVszW: PROP_VAL_UNION** contains an encoding of the values of a property that can contain multiple Unicode string values. These string values are NULL-terminated. The number of values that can be encoded in this structure is 100,000.

**MVft: PROP_VAL_UNION** contains an encoding of the value of a property that can contain multiple 64-bit integer values. The values are encoded as FILETIME structures. The number of values that can be encoded in this structure is 100,000.

**lReserved:** Reserved. All clients and servers MUST set this value to the constant 0x00000000.

### 2.2.2.12 PropertyValue_r Structure

The PropertyValue_r structure is an encoding of the PropertyValue_r data structure, as specified in [MS-OXCDATA] section 2.11.2.2.

For property values with uninterpreted byte values, the permissible number of bytes in the PropertyValue_r structure exceeds that of the PropertyValue data structure. For property values with multiple values, the permissible number of values in the PropertyValue_r structure exceeds that of the PropertyValue data structure. The semantic meaning is otherwise unchanged from the PropertyValue data structure.

```c
typedef struct _PropertyValue_r {
    DWORD ulPropTag;
    DWORD ulReserved;
    [switch_is((long)(ulPropTag & 0x0000FFFF))]
        PROP_VAL_UNION Value;
} PropertyValue_r;
```

**ulPropTag:** Encodes the proptag of the property whose value is represented by the PropertyValue_r data structure.

**ulReserved:** Reserved. All clients and servers MUST set this value to the constant 0x00000000.

**Value:** Encodes the actual value of the property represented by the PropertyValue_r data structure. The type value held is specified by the property type of the proptag in the ulPropTag field.

### 2.2.3 PropertyRow_r Structure

The PropertyRow_r structure is an encoding of the StandardPropertyRow data structure, as specified in [MS-OXCDATA] section 2.8.1.1. The semantic meaning is unchanged from the StandardPropertyRow data structure.
typedef struct _PropertyRow_r {
    DWORD Reserved;
    [range(0,100000)] DWORD cValues;
    [size_is(cValues)] PropertyValue_r* lpProps;
} PropertyRow_r;

Reserved: Reserved. All clients and servers MUST set this value to the constant 0x00000000.

cValues: The number of PropertyValue_r structures represented in the PropertyRow_r structure. This value MUST NOT exceed 100,000.

lpProps: Encodes the ValueArray field of the StandardPropertyRow data structure.

2.2.4 PropertyRowSet_r Structure

The PropertyRowSet_r structure is an encoding of the PropertyRowSet_r data structure, as specified in [MS-OXCDATA] section 2.8.2.2.

The permissible number of PropertyRows in the PropertyRowSet_r data structure exceeds that of the PropertyRowSet data structure. The semantic meaning is otherwise unchanged from the PropertyRowSet data structure.

typedef struct _PropertyRowSet_r {
    [range(0,100000)] DWORD cRows;
    [size_is(cRows)] PropertyRow_r aRow[];
} PropertyRowSet_r;

cRows: Encodes the RowCount field of the PropertyRowSet data structures. This value MUST NOT exceed 100,000.

aRow: Encodes the Rows field of the PropertyRowSet data structure.

2.2.5 Restrictions

The following structures are used to represent restrictions of a table, as defined in [MS-OXCDATA] section 2.12.

2.2.5.1 AndRestriction_r Restriction, OrRestriction_r Restriction

The AndRestriction_r, OrRestriction_r restriction types share a single RPC encoding. The AndOrRestriction_r structure is an encoding of both the AndRestriction_r data structure and the OrRestriction_r data structure, as specified in [MS-OXCDATA] sections 2.12.1.2 and 2.12.2.2. These two data structures share the same data layout, so a single encoding is included in the Exchange Server NSPI Protocol. The sense of the data structure's use is derived from the context of its inclusion in the RestrictionUnion_r data structure, as specified in section 2.2.5.6.

The permissible number of restriction structures in the AndRestriction_r and OrRestriction_r data structures exceeds that of the AndRestriction and OrRestriction structures. The semantic meaning is otherwise unchanged from the AndRestriction and OrRestriction data structures, as context dictates.

typedef struct _AndOrRestriction_r {
    [range(0,100000)] DWORD cRes;
    [size_is(cRes)] Restriction_r* lpRes;
} AndOrRestriction_r,
cRes: Encodes the RestrictCount field of the AndRestriction and OrRestriction data structures. This value MUST NOT exceed 100,000.

lpRes: Encodes the Restricts field of the AndRestriction and OrRestriction data structures.

### 2.2.5.2 NotRestriction_r Restriction

The NotRestriction_r structure is an encoding of the NotRestriction_r data structure, as specified in [MS-OXNSPI] section 2.12.3.2. The semantic meaning is unchanged from the NotRestriction data structure.

```c
typedef struct _NotRestriction_r {
    Restriction_r* lpRes;
} NotRestriction_r;
```

lpRes: Encodes the Restriction field of the NotRestriction data structure.

### 2.2.5.3 ContentRestriction_r Restriction

The ContentRestriction_r structure is an encoding of the ContentRestriction_r data structure, as specified in [MS-OXNSPI] section 2.12.4.2. The semantic meaning is unchanged from the ContentRestriction data structure.

```c
typedef struct _ContentRestriction_r {
    DWORD ulFuzzyLevel;
    DWORD ulPropTag;
    PropertyValue_r * lpProp;
} ContentRestriction_r;
```

ulFuzzyLevel: Encodes the FuzzyLevelLow and FuzzyLevelHigh fields of the ContentRestriction data structure.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 |
|   |   |   |   |   |   |   | FuzzyLevelLow |   |   |   |   |   |   |   | FuzzyLevelHigh |   |   |   |   |   |   |   |

R1: Reserved. All clients MUST set this value to the constant 0x00.

ulPropTag: Encodes the PropertyTag field of the ContentRestriction data structure.

lpProp: Encodes the TaggedValue field of the ContentRestriction data structure.
2.2.5.4 PropertyRestriction_r Restriction

The PropertyRestriction_r structure is an encoding of the PropertyRestriction_r data structure, as specified in [MS-OXCDATA] section 2.12.5.2. The semantic meaning is unchanged from the PropertyRestriction data structure.

```c
typedef struct _PropertyRestriction_r {
    DWORD relop;
    DWORD ulPropTag;
    PropertyValue_r* lpProp;
} PropertyRestriction_r;
```

**relop**: Encodes the RelOp field of the PropertyRestriction data structure.

**ulPropTag**: Encodes the PropTag field of the PropertyRestriction data structure.

**lpProp**: Encodes the TaggedValue field of the PropertyRestriction data structure.

2.2.5.5 ExistRestriction_r Restriction

The ExistRestriction_r structure is an encoding of the ExistRestriction_r data structure, as specified in [MS-OXCDATA] section 2.12.9.2. The semantic meaning is unchanged from the ExistRestriction data structure.

```c
typedef struct _ExistRestriction_r {
    DWORD ulReserved1;
    DWORD ulPropTag;
    DWORD ulReserved2;
} ExistRestriction_r;
```

**ulReserved1**: Reserved. All clients MUST set this value to the constant 0x00000000.

**ulPropTag**: Encodes the PropTag field of the ExistRestriction data structure.

**ulReserved2**: Reserved. All clients MUST set this value to the constant 0x00000000.

2.2.5.6 RestrictionUnion_r Restriction

The RestrictionUnion_r structure encodes a single instance of any type of restriction. It is an aggregation data structure, allowing a single parameter to an NSPI method to contain any type of restriction data structure.

```c
typedef
[switch_type(long)]
union _RestrictionUnion_r {
    [case(0x00000000)]
    AndRestriction_r resAnd;
    [case(0x00000001)]
    OrRestriction_r resOr;
    [case(0x00000002)]
    NotRestriction_r resNot;
    [case(0x00000003)]
    ContentRestriction_r resContent;
    [case(0x00000004)]
    PropertyRestriction_r resProperty;
    [case(0x00000008)]
    ExistRestriction_r resExist;
```

resAnd:  RestrictionUnion_r contains an encoding of an AndRestriction.
resOr:  RestrictionUnion_r contains an encoding of an OrRestriction.
resNot:  RestrictionUnion_r contains an encoding of a NotRestriction.
resContent:  RestrictionUnion_r contains an encoding of a ContentRestriction.
resProperty:  RestrictionUnion_r contains an encoding of a PropertyRestriction.
resExist:  RestrictionUnion_r contains an encoding of an ExistRestriction.

2.2.5.7 Restriction_r Restriction

The Restriction_r structure is an encoding of the Restriction filters, as specified in [MS-OXCDATA] section 2.12.

The permissible number of Restriction structures encoded in AndRestriction_r and OrRestriction_r data structures recursively included in the Restriction_r data type exceeds that of the AndRestriction_r and OrRestriction_r data structures recursively included in the Restriction filters. The semantic meaning is otherwise unchanged from the Restriction filters.

typedef struct _Restriction_r {
    DWORD rt;
    [switch_is((long)rt)] RestrictionUnion_r res;
} Restriction_r;

rt:  Encodes the RestrictType field common to all restriction structures.
res:  Encodes the actual restriction specified by the type in the rt field.

2.2.6 Property Name/Property ID Structures

The following structures are used to represent named properties, as specified in [MS-OXCDATA] section 2.6.

2.2.6.1 PropertyName_r Structure

The PropertyName_r structure is an encoding of the PropertyName_r data structure, as specified in [MS-OXCDATA] section 2.6.2. The semantic meaning is unchanged from the PropertyName data structure.

typedef struct PropertyName_r {
    FlatUID_r* lpguid;
    DWORD ulReserved;
    long lID;
} PropertyName_r;

lpguid:  Encodes the GUID field of the PropertyName data structure. This field is encoded as a FlatUID_r data structure.
ulReserved:  Reserved. All clients MUST set this value to the constant 0x00000000.
IID: Encodes the IID field of the PropertyName data structure. In addition to the definition of the LID field, this field is always present in the PropertyName_r data structure; it is not optional.

2.2.7 String Arrays

The following structures are used to aggregate a number of strings into a single data structure.

2.2.7.1 StringsArray_r

The StringsArray_r structure is used to aggregate a number of character type strings into a single data structure.

typedef struct _StringsArray {
    [range(0,100000)] DWORD Count;
    [string, size_is(Count)] char* Strings[];
} StringsArray_r;

Count: The number of character string structures in this aggregation. The value MUST NOT exceed 100,000.

Strings: The list of character type strings in this aggregation. The strings in this list are NULL-terminated.

2.2.7.2 WStringsArray_r

The WStringsArray_r structure is used to aggregate a number of wchar_t type strings into a single data structure.

typedef struct _WStringsArray {
    [range(0,100000)] DWORD Count;
    [string, size_is(Count)] wchar_t* Strings[];
} WStringsArray_r;

Count: The number of character strings structures in this aggregation. The value MUST NOT exceed 100,000.

Strings: The list of wchar_t type strings in this aggregation. The strings in this list are NULL-terminated.

2.2.8 STAT

The STAT structure is used to specify the state of a table and location information that applies to that table. It appears as both an input parameter and an output parameter in many NSPI methods.

typedef struct {
    DWORD SortType;
    DWORD ContainerID;
    DWORD CurrentRec;
    long Delta;
    DWORD NumPos;
    DWORD TotalRecs;
    DWORD CodePage;
    DWORD TemplateLocale;
    DWORD SortLocale;
} STAT;
SortType: This field contains a DWORD [MS-DTYP] value that represents a sort order. The client sets this field to specify the sort type of this table. Possible values are specified in section 2.2.10. The server MUST NOT modify this field.

ContainerID: This field contains a Minimal Entry ID. The client sets this field to specify the Minimal Entry ID of the address book container that this STAT structure represents. The client obtains these Minimal Entry IDs from the server's address book hierarchy table. The server MUST NOT modify this field in any NSPI method except the NspiGetMatches method.

CurrentRec: This field contains a Minimal Entry ID. The client sets this field to specify a beginning position in the table for the start of an NSPI method. The server sets this field to report the end position in the table after processing an NSPI method.

Delta: This field contains a long value. The client sets this field to specify an offset from the beginning position in the table for the start of an NSPI method. If the NSPI method returns a success value, the server MUST set this field to 0.

NumPos: This field contains a DWORD value that specifies a position in the table. The client sets this field to specify a fractional position for the beginning position in the table for the start of an NSPI method, as specified in section 3.1.4.5.2. If absolute positioning, as specified in section 3.1.4.5.1, is used, the value of this field specified by the client will be ignored by the server. The server sets this field to specify the approximate fractional position at the end of an NSPI method. This value is a zero index; the first element in a table has the numeric position 0. Although the protocol places no boundary or requirements on the accuracy of the approximation the server reports, it is recommended that implementations maximize the accuracy of the approximation to improve usability of the server for clients.

TotalRecs: This field contains a DWORD value that specifies the number of rows in the table. The client sets this field to specify a fractional position for the beginning position in the table for the start of an NSPI method, as specified in section 3.1.4.5.2. If absolute positioning, as specified in section 3.1.4.5.1, is used, the value of this field specified by the client will be ignored by the server. The server sets this field to specify the total number of rows in the table. Unlike the NumPos field, the server MUST report this number accurately; an approximation is insufficient.

CodePage: This field contains a DWORD value that represents a code page. The client sets this field to specify the code page the client uses for non-Unicode strings. The server MUST use this value during string handling, as specified in section 3.1.4.3. The server MUST NOT modify this field.

TemplateLocale: This field contains a DWORD value that represents a language code identifier (LCID). The client sets this field to specify the LCID associated with the template the client wants the server to return. The server MUST NOT modify this field.

SortLocale: This field contains a DWORD value that represents an LCID. The client sets this field to specify the LCID that it wants the server to use when sorting any strings. The server MUST use this value during sorting, as specified in section 3.1.4.3. The server MUST NOT modify this field.

2.2.9 EntryIDs

Each object in the address book is identified by one or more EntryIDs, as specified in section 3.1.4.6. The following table lists the three types of EntryIDs.

<table>
<thead>
<tr>
<th>EntryID name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MinimalEntryID</td>
<td>A minimal identifier</td>
</tr>
</tbody>
</table>

[MS-OXNSPI] - v20220215
Exchange Server Name Service Provider Interface (NSPI) Protocol
Copyright © 2022 Microsoft Corporation
Release: February 15, 2022
### 2.2.9.1 MinimalEntryID

A **Minimal Entry ID** is a single DWORD value that identifies a specific object in the *address book*. Minimal Entry IDs with values less than 0x00000010 are used by clients as signals to trigger specific behaviors in specific NSPI methods. Except in those places where the protocol defines a specific behavior for these Minimal Entry IDs, the server MUST treat these Minimal Entry IDs as Minimal Entry IDs that do not specify an object in the address book. Specific values used in this way are defined in sections 2.2.1.8 and 2.2.1.9.

Minimal Entry IDs are created and assigned by Exchange NSPI server. The algorithm used by a server to create a Minimal Entry ID is not restricted by this protocol. A Minimal Entry ID is valid only to servers that respond to an NspiBind method, as specified in section 3.1.4.1.1, with the same server GUID as that used by the server that created the Minimal Entry ID. It is not possible for a client to predict a Minimal Entry ID.

This type is declared as follows:

```c
typedef DWORD MinEntryID;
```

### 2.2.9.2 EphemeralEntryID

The **EphemeralEntryID** structure identifies a specific object in the *address book*. Additionally, it encodes the server that issued the Ephemeral Entry ID and enough information for a client to make a decision as to how to display the object to an end user.

A server MUST NOT change an object's Ephemeral Entry ID during the lifetime of an NSPI session.

<table>
<thead>
<tr>
<th>ID Type</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProviderUID</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Display Type</td>
<td>R4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**ID Type (1 byte):** The type of this ID. The value is the constant 0x87. The server uses the presence of this value to identify this EntryID as an Ephemeral Entry ID rather than a Permanent Entry ID.

**R1 (1 byte):** Reserved. All clients and servers MUST set this value to the constant 0x00.

**R2 (1 byte):** Reserved. All clients and servers MUST set this value to the constant 0x00.

**R3 (1 byte):** Reserved. All clients and servers MUST set this value to the constant 0x00.

**ProviderUID (16 bytes):** A FlatUID_r value, as specified in section 2.2.2.1, that contains the GUID of the server that issued this Ephemeral Entry ID.

**R4 (4 bytes):** Reserved. All clients and servers MUST set this value to the constant 0x00000001.

**Display Type (4 bytes):** The display type of the object specified by this Ephemeral Entry ID. This value is expressed in little-endian format. Valid values for this field are specified in section 2.2.1.3. The display type is not considered part of the object's identity; it is set in the EphemeralEntryID structure by the server as a convenience to clients. The server MUST set this field when this data structure is returned in an output parameter. A server MUST ignore values of this field on input parameters.

**MId (4 bytes):** The Minimal Entry ID of this object, as specified in section 2.2.9.1. This value is expressed in little-endian format.

### 2.2.9.3 PermanentEntryID

The PermanentEntryID structure identifies a specific object in the address book. Additionally, it encodes the constant Exchange Server NSPI Protocol interface (via the ProviderUID field) and enough information for a client to make a decision as to how to display the object to an end user.

**Permanent Entry IDs** are transmitted in the protocol as values with the PtypBinary property type.

A server MAY allow an object's distinguished name (DN) to change. If this happens, the server is expected to map a Permanent Entry ID that contains the original DN to the object with the new DN. When returning a PermanentEntryID structure to satisfy a query from a client, a server MUST use the most current version of an object's DN.
<table>
<thead>
<tr>
<th>Display Type String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distinguished Name (variable)</td>
</tr>
</tbody>
</table>

**ID Type (1 byte):** The type of this ID. The value is the constant 0x00. The server uses the presence of this value to identify this **EntryID** as a Permanent Entry ID rather than an Ephemeral Entry ID.

**R1 (1 byte):** Reserved. All clients and servers MUST set this value to the constant 0x00.

**R2 (1 byte):** Reserved. All clients and servers MUST set this value to the constant 0x00.

**R3 (1 byte):** Reserved. All clients and servers MUST set this value to the constant 0x00.

**ProviderUID (16 bytes):** A **FlatUID_r** value that contains the constant **GUID** specified in Permanent Entry ID **GUID**, as specified in section 2.2.1.7.

**R4 (4 bytes):** Reserved. All clients and servers MUST set this value to the constant 0x00000001.

**Display Type String (4 bytes):** The display type of the object specified by this Permanent Entry ID. This value is expressed in little-endian format. Valid values for this field are specified in section 2.2.1.3. The display type is not considered part of the object’s identity; it is set in the **PermanentEntryID** field by the server as a convenience to clients. A server MUST set this field when this data structure is returned in an output parameter. A server MUST ignore values of this field on input parameters.

**Distinguished Name (variable):** The DN of the object specified by this Permanent Entry ID. The value is expressed as a DN, as specified in [MS-OXOABK] section 2.2.1.1.<2>

### 2.2.10 NSPI_HANDLE

The **NSPI_HANDLE** handle is an **RPC** context handle that is used to share a session between method calls.

The RPC context handle is specified in [C706] section 2.3.1.

This type is declared as follows:

```
typedef [context_handle] void* NSPI_HANDLE;
```
3 Protocol Details

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

The client MUST call the NSPI NspiBind method, as specified in section 3.1.4.1.1, in order to obtain an RPC context handle to be used in all other NSPI methods. The NSPI NspiUnbind method, as specified in section 3.1.4.1.2, destroys this context handle. Therefore, it is not possible to call any methods other than NspiBind immediately after a call to NspiUnbind. The final method a client MUST call is NspiUnbind.

3.1 Server Details

This protocol enables address book access to a directory data store. This includes address book hierarchy table discovery, address creation template table discovery, address book container access and browsing, and read and modification of individual address book entries. In addition to the abstract data model specified here, this specification uses the address book data model, as specified in [MS-OXOABK], for the server of this protocol. This specification uses the definitions of object properties specified in [MS-OXPROPS].

3.1.1 Abstract Data Model

None.

3.1.2 Timers

This protocol does not introduce any timers. For details about any transport-level timers, see [MS-RPCE].

3.1.3 Initialization

Each server MUST have at least one unique GUID, used to identify an NSPI session, as specified in section 3.1.4.1.1. The server MUST acquire this GUID before it is prepared to respond to Exchange Server NSPI Protocol methods. The protocol does not constrain how a server acquires this GUID. The server MUST maintain this GUID for the duration of an NSPI session. Although the protocol places no further boundary or requirements on the time period for which the server maintains this GUID, it is recommended that implementations maximize this time period to improve the usability of the server for clients.

Each server maintains a set of Address Book objects and containers, as specified in [MS-OXOABK]. The Exchange Server NSPI Protocol does not constrain how a server obtains its initial data set, nor does it constrain the contents of this initial data set. How a server obtains this data is an implementation-specific detail.

When a server is prepared to respond to Exchange Server NSPI Protocol methods, it creates an RPC listening endpoint, as specified in section 2.1.

3.1.4 Message Processing Events and Sequencing Rules

This protocol MUST indicate to the RPC runtime that it is to perform a strict Network Data Representation (NDR) data consistency check at target level 6.0, as specified in [MS-RPCE].
This protocol MUST indicate to the RPC runtime via the `strict_context_handle` property that it is to reject use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE].

This protocol MUST indicate to the RPC runtime via the `type_strict_context_handle` property that it is to reject use of context handles created by a method that creates a different type of context handle, as specified in [MS-RPCE].

This interface includes the methods listed in the following table.

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NspiBind</td>
<td>Initiates a session with the server.</td>
</tr>
<tr>
<td></td>
<td><strong>Opnum</strong>: 0</td>
</tr>
<tr>
<td>NspiUnbind</td>
<td>Concludes a session with the server.</td>
</tr>
<tr>
<td></td>
<td><strong>Opnum</strong>: 1</td>
</tr>
<tr>
<td>NspiUpdateStat</td>
<td>Updates the logical position in a specified table.</td>
</tr>
<tr>
<td></td>
<td><strong>Opnum</strong>: 2</td>
</tr>
<tr>
<td>NspiQueryRows</td>
<td>Returns information about a set of rows in a table.</td>
</tr>
<tr>
<td></td>
<td><strong>Opnum</strong>: 3</td>
</tr>
<tr>
<td>NspiSeekEntries</td>
<td>Seeks forward in a specified table and update the logical position in that table.</td>
</tr>
<tr>
<td></td>
<td><strong>Opnum</strong>: 4</td>
</tr>
<tr>
<td>NspiGetMatches</td>
<td>Restricts a specific table based on input parameters and return the resultant Explicit Table.</td>
</tr>
<tr>
<td></td>
<td><strong>Opnum</strong>: 5</td>
</tr>
<tr>
<td>NspiResortRestriction</td>
<td>Changes the sort order of an Explicit Table.</td>
</tr>
<tr>
<td></td>
<td><strong>Opnum</strong>: 6</td>
</tr>
<tr>
<td>NspiDNToMId</td>
<td>Translates a DN to a Minimal Entry ID.</td>
</tr>
<tr>
<td></td>
<td><strong>Opnum</strong>: 7</td>
</tr>
<tr>
<td>NspiGetPropList</td>
<td>Returns a list of all the properties which exist on a specific object in the address book.</td>
</tr>
<tr>
<td></td>
<td><strong>Opnum</strong>: 8</td>
</tr>
<tr>
<td>NspiGetProps</td>
<td>Returns a list of properties and their values for a specific object in the address book.</td>
</tr>
<tr>
<td></td>
<td><strong>Opnum</strong>: 9</td>
</tr>
<tr>
<td>NspiCompareMIds</td>
<td>Compares the position of two rows in a table.</td>
</tr>
<tr>
<td></td>
<td><strong>Opnum</strong>: 10</td>
</tr>
<tr>
<td>NspiModProps</td>
<td>Modifies a property of a row in the address book.</td>
</tr>
<tr>
<td></td>
<td><strong>Opnum</strong>: 11</td>
</tr>
<tr>
<td>NspiGetSpecialTable</td>
<td>Retrieves the address book hierarchy table of the server, or retrieves the address creation table from the server.</td>
</tr>
<tr>
<td></td>
<td><strong>Opnum</strong>: 12</td>
</tr>
<tr>
<td>NspiGetTemplateInfo</td>
<td>Retrieves addressing or display templates from the server.</td>
</tr>
<tr>
<td></td>
<td><strong>Opnum</strong>: 13</td>
</tr>
<tr>
<td>Method name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NspiModLinkAtt</td>
<td>Modifies a property of a row in the address book. Applies only to rows that support the PtypEmbeddedTable property type. Opnum: 14</td>
</tr>
<tr>
<td>NspiQueryColumns</td>
<td>Returns the information about a list of all the properties that the server is aware of. Opnum: 16</td>
</tr>
<tr>
<td>NspiResolveNames</td>
<td>Performs ANR on a set of provided names. The names are specified in the code page of the client. Opnum: 19</td>
</tr>
<tr>
<td>NspiResolveNamesW</td>
<td>Performs ANR on a set of provided names. The names are specified in the Unicode character set. Opnum: 20</td>
</tr>
</tbody>
</table>

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

The server MUST return the value **NotEnoughMemory** if unable to complete processing a method due to errors allocating memory.

The server MUST return the value **OutOfResources** if unable to complete processing a method due to lack of some non-memory resource.

The server MUST return the value **GeneralFailure** if unable to complete processing a method for reasons other than those specified here or in the methods details.

The server MUST return the value **Success** if it completes without some other return value being specified in the method details.

**Note** Gaps in the opnum numbering sequence represent opnums that are reserved for local use. The server behavior is undefined, because it does not affect interoperability.

### 3.1.4.1 NSPI Methods

#### 3.1.4.1.1 NspiBind (Opnum 0)

The **NspiBind** method initiates a session between a client and the server.

```c
long NspiBind(
    [in] handle_t hRpc,
    [in] DWORD dwFlags,
    [in] STAT* pStat,
    [in, out, unique] FlatUID_r* pServerGuid,
    [out, ref] NSPI_HANDLE* contextHandle
);
```

**hRpc:** An **RPC** binding handle parameter, as specified in [C706] section 2.

**dwFlags:** A **DWORD** [MS-DTYP] value that contains a set of bit flags. The server MUST ignore values other than the bit flag **fAnonymousLogin** (0x00000020).

**pStat:** A pointer to a **STAT** block that describes a logical position in a specific **address book container**. This parameter is used to specify input parameters from the client.

**pServerGuid:** The value NULL or a pointer to a **GUID** value that is associated with the specific server.
**contextHandle:** An RPC context handle, as specified in section 2.2.10.

**Return Values:** The server returns a **LONG** [MS-DTYP] value that specifies the return status of the method.

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

**Server Processing Rules:** Upon receiving this message, the server MUST process the data from the message subject to the following constraints:

1. If the **CodePage** field of the input parameter **pStat** contains the value **CP_WINUNICODE**, the server MUST return one of the return values specified in section 2.2.1.2. No further constraints are applied to server processing of this method; in this case server behavior is undefined. Note especially that there is no constraint on the data the server returns in any output parameter other than the return value, nor is there any constraint on how or if the server changes its state.

2. The server MAY make additional validations including but not limited to limiting the number of concurrent connections to any specific client or checking the data access rights of the client. If these checks fail, the server MUST return "LogonFailed".

3. A value of "fAnonymousLogin" in the input parameter **dwFlags** indicates that the server did not validate that the client is an authenticated user. The server MAY ignore this request.

4. Subject to constraint 3, the server MAY authenticate the client. How a server authenticates a client is an implementation-specific detail.

5. The **CodePage** field of the input parameter **pStat** specifies the code page to use in this session. If the server will not service connections using that code page, the server MUST return the error code "InvalidCodepage".

6. Subject to the prior constraints, if the input parameter **pServerGuid** is not NULL, the server MUST set the output parameter **pServerGuid** to a GUID associated with the server. The server MAY use a different GUID for each RPC connection. Each server MUST use a different GUID.

7. If no other return code has been set, the server MUST return the value "Success".

### 3.1.4.1.2 NspiUnbind (Opnum 1)

The **NspiUnbind** method destroys the context handle. No other action is taken.

```c
DWORD NspiUnbind(
    [in, out] NSPI_HANDLE* contextHandle,
    [in] DWORD Reserved
);
```

**contextHandle:** An **RPC** context handle as specified in section 2.2.10.

**Reserved:** A **DWORD** [MS-DTYP] value reserved for future use. MUST be ignored by the server.

**Return Values:** The server returns a **DWORD** value that specifies the return status of the method.

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

**Server Processing Rules:** Upon receiving this message, the server MUST process the data from the message subject to the following constraints:

1. If the server successfully destroys the context handle, the server MUST return the value "UnbindSuccess", as specified in section 2.2.1.2.
2. If the server does not successfully destroy the context handle, the server MUST return the value "UnbindFailure", as specified in section 2.2.1.2.

3. The server MUST set the output parameter contextHandle to NULL.

3.1.4.1.3 NspiGetSpecialTable (Opnum 12)

The NspiGetSpecialTable method returns the rows of a special table to the client. The special table can be an address book hierarchy table or an address creation table.

```c
long NspiGetSpecialTable(
    [in] NSPI_HANDLE hRpc,
    [in] DWORD dwFlags,
    [in] STAT* pStat,
    [in, out] DWORD* lpVersion,
    [out] PropertyRowSet_r** ppRows
);
```

**hRpc**: An RPC context handle, as specified in section 2.2.10.

**dwFlags**: A DWORD [MS-DTYP] value that contains a set of bit flags. The server MUST ignore values other than the bit flags NspiAddressCreationTemplates and NspiUnicodeStrings.

**pStat**: A pointer to a STAT block that describes a logical position in a specific address book container. This parameter is used to specify input parameters from the client.

**lpVersion**: A reference to a DWORD. On input, holds the value of the version number of the address book hierarchy table that the client has.

**ppRows**: A PropertyRowSet_r structure. On return, holds the rows for the table that the client is requesting.

**Return Values**: The server returns a LONG [MS-DTYP] value that specifies the return status of the method.

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

**Server Processing Rules**: Upon receiving this message, the server MUST process the data from the message subject to the following constraints:

1. If the input parameter dwFlags does not contain the value "NspiUnicodeStrings", and the input parameter dwFlags does not contain the value "NspiAddressCreationTemplates", and the CodePage field of the input parameter pStat contains the value CP_WINUNICODE, the server MUST return one of the return values specified in section 2.2.1.2. No further constraints are applied to server processing of this method; in this case server behavior is undefined. Note especially that there is no constraint on the data the server returns in any output parameter other than the return value, nor is there any constraint on how or whether the server changes its state.

2. If the server returns any return value other than "Success", the server MUST return a NULL for the output parameter ppRows.

3. The server MAY make additional validations, as described in section 5. If the server chooses to limit the visibility of data based on these validations, the server MUST proceed as if that data did not exist in the address book.

4. If the input parameter dwFlags contains both the value "NspiAddressCreationTemplates" and the value "NspiUnicodeStrings", the server MUST ignore the value "NspiUnicodeStrings" and proceed as if the parameter dwFlags contained only the value "NspiAddressCreationTemplates".
5. If the input parameter \textit{dwFlags} does not contain the value "NspiAddressCreationTemplates", the client is requesting the rows of the server's address book hierarchy table, as specified in section 3.1.4.4.3.1.

6. If the client is requesting the rows of the server's address book hierarchy table and the server is not maintaining an address book hierarchy table, the server MUST return the error code "OutOfResources".

7. If the client is requesting the rows of the server's address book hierarchy table, the input parameter \textit{lpVersion} contains a version number. If the version number of the address book hierarchy table the server is holding matches this version number, the server MUST proceed as if the address book hierarchy table had no rows. <3>

8. If the client is requesting the rows of the server's address book hierarchy table and the server returns the value "Success", the server MUST set the output parameter \textit{lpVersion} to the version of the server's address book hierarchy table. <4>

9. If the input parameter \textit{dwFlags} contains the value "NspiAddressCreationTemplates", the client is requesting the rows of an address creation table, as specified in section 3.1.4.4.3.2.

10. There is no constraint on the parameter \textit{lpVersion} if the client is requesting the rows of an address creation table.

11. If the client is requesting the rows of an address creation table, the TemplateLocale field of the input parameter \textit{pStat} specifies the LCID for which the client requires an address creation table. If the server does not maintain an address creation table for that LCID, the server MUST proceed as if it maintained an address creation table with no rows for that LCID. That is, the server MUST NOT return an error code if it does not maintain an address creation table for that LCID.

12. If the input parameter \textit{dwFlags} contains the value "NspiUnicodeStrings" and the client is requesting the rows of the server's address book hierarchy table, the server MUST express string-valued properties in the table as \textit{Unicode} values, as specified in section 3.1.4.3.

13. If the input parameter \textit{dwFlags} does not contain the value "NspiUnicodeStrings" and the client is requesting the rows of the server's address book hierarchy table, and the CodePage field of the input parameter \textit{pStat} does not contain the value \textit{CP_WINUNICODE}, the server MUST express string-valued properties as 8-bit strings in the \textit{code page} specified by the field CodePage in the input parameter \textit{pStat}. For more details, see section 3.1.4.3.

14. The server MUST return the following properties for each container in the hierarchy, in the order listed:

   - \textit{PidTagEntryId} ([MS-OXPROPS] section 2.683)
   - \textit{PidTagContainerFlags} ([MS-OXPROPS] section 2.644)
   - \textit{PidTagDepth} ([MS-OXPROPS] section 2.673)
   - \textit{PidTagAddressBookContainerId} ([MS-OXPROPS] section 2.512)
   - \textit{PidTagDisplayName} ([MS-OXPROPS] section 2.676)
   - \textit{PidTagAddressBookIsMaster} ([MS-OXPROPS] section 2.545)
   - \textit{PidTagAddressBookParentEntryId} ([MS-OXPROPS] section 2.559) (optional, and MUST be the seventh column if it is included)

15. For every row returned, all of these properties except \textit{PidTagAddressBookParentEntryId} MUST be present, and each of them MUST have a value prescribed under its definition.
16. The PidTagEntryId property MUST be in the form of a PermanentEntryID structure, as section 2.2.9.3, with its PidTagDisplayType property having the value DT_CONTAINER, as specified in section 2.2.1.3, and its DN following the addresslist-dn format specification, as specified in [MS-OXOABK] section 2.2.1.1. When the object is the Global Address List (GAL) container, its DN MUST follow the gal-addresslist-dn format specification.

17. Subject to the prior constraints, the server returns the rows of the table requested by the client in the output parameter ppRows.

18. If no error condition has been specified by the previous constraints, the server MUST return the value "Success".

3.1.4.1.4 NspiUpdateStat (Opnum 2)

The NspiUpdateStat method updates the STAT block that represents position in a table to reflect positioning changes requested by the client.

```c
long NspiUpdateStat(
    [in] NSPI_HANDLE hRpc,
    [in] DWORD Reserved,
    [in, out] STAT* pStat,
    [in, out, unique] long* plDelta
);
```

hRpc: An RPC context handle, as specified in section 2.2.10.


pStat: A pointer to a STAT block describing a logical position in a specific address book container. This parameter is used to specify both input parameters from the client and return values from the server.

plDelta: The value NULL or a pointer to a LONG [MS-DTYP] value that, on return, indicates movement within the address book container specified by the input parameter pStat. The server MUST ignore the value specified by this parameter in the request if it is not NULL.

Return Values: The server returns a long value specifying the return status of the method.

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

Server Processing Rules: Upon receiving this message, the server MUST process the data from the message subject to the following constraints:

1. If the CodePage field of the input parameter pStat contains the value CP_WINUNICODE, the server MUST return one of the return values specified in section 2.2.1.2. No further constraints are applied to server processing of this method; in this case server behavior is undefined. Note especially that there is no constraint on the data the server returns in any output parameter other than the return value, nor is there any constraint on how or whether the server changes its state.

2. If the server returns any return value other than "Success", the server MUST NOT modify the output parameter pStat.

3. The server MAY make additional validations, as described in section 5. If the server chooses to limit the visibility of data based on these validations, the server MUST proceed as if that data did not exist in the address book.

4. If the server is unable to locate the address book container specified by the ContainerID field in the input parameter pStat, the server MUST return the return value InvalidBookmark.
5. The server locates the initial position row in the table specified by the **ContainerID** field of the input parameter **pStat** as follows:

   - If the row specified by the **CurrentRec** field of the input parameter **pStat** is not **MID_CURRENT**, the server locates that row as the initial position row using the absolute position, as specified in section 3.1.4.5.1. If the row cannot be found, the server MUST return the error "NotFound".
   
   - If the row specified by the **CurrentRec** field of the input parameter **pStat** is **MID_CURRENT**, the server locates the initial position row using the fractional position specified in the **NumPos** field of the input parameter **pStat** as specified in section 3.1.4.5.2.

6. After locating the initial position row in the current table, the server locates the final position row by moving forward or backward in the table from the current position row as specified in the **Delta** field of the input parameter **pStat**, with the constraints specified in section 3.1.4.5 with respect to movement beyond the beginning or end of a table.

7. If the input parameter **plDelta** is not null, the server MUST set it to the actual number of rows between the initial position row and the final position row. If the input parameter **plDelta** is null, the server MUST set the output parameter **plDelta** to null.

8. The server MUST set the **CurrentRec** field of the parameter **pStat** to the **Minimal Entry ID** of the current row in the current address book container.

9. The server MUST set the **NumPos** field of the parameter **pStat** to the approximate numeric position of the current row of the current address book container according to section 3.1.4.5.2.

10. The server MUST set the **TotalRecs** field of the parameter **pStat** to the number of rows in the current address book container according to section 3.1.4.5.2.

11. The server MUST leave all other fields of the parameter **pStat** unchanged.

12. If no error condition has been specified by the previous constraints, the server MUST return "Success".

### 3.1.4.1.5 NspiQueryColumns (Opnum 16)

The **NspiQueryColumns** method returns a list of all the properties that the server is aware of. It returns this list as an array of proptags.

```c
long NspiQueryColumns(
    [in]   NSPI_HANDLE hRpc,
    [in]   DWORD Reserved,
    [in]   DWORD dwFlags,
    [out]  PropertyTagArray_r** ppColumns);
```

- **hRpc**: An RPC context handle, as specified in section 2.2.10.
- **Reserved**: A DWORD [MS-DTYP] value reserved for future use. Ignored by the server.
- **dwFlags**: A DWORD value that contains a set of bit flags. The server MUST ignore values other than the bit flag **NspiUnicodeProptypes**.
- **ppColumns**: A reference to a **PropertyTagArray_r** structure. On return, contains a list of proptags.
- **Return Values**: The server returns a **LONG** [MS-DTYP] value that specifies the return status of the method.
Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Server Processing Rules: Upon receiving this message, the server MUST process the data from the message subject to the following constraints:

1. If the server returns any return value other than "Success", the server MUST return a NULL for the output parameter *ppColumns*.

2. The server MAY make additional validations, as described in section 5. If the server chooses to limit the visibility of data based on these validations, the server MUST proceed as if that data did not exist in the *address book*.

3. If the input parameter *dwFlags* contains the bit flag **NspiUnicodePropTypes**, then the server MUST report the *property type* of all string valued properties as **PtypString**.

4. If the input parameter *dwFlags* does not contain the bit flag **NspiUnicodePropTypes**, the server MUST report the property type of all string valued properties as **PtypString8**.

5. Subject to the prior constraints, the server MUST construct a list of all the properties it is aware of and return that list as a *PropertyTagArray_r* structure, as specified in section 2.2.2.2, in the output parameter *ppColumns*. The protocol does not constrain the order of this list.

6. If no error condition has been specified by the previous constraints, the server MUST return the value "Success".

3.1.4.1.6 NspiGetPropList (Opnum 8)

The **NspiGetPropList** method returns a list of all the properties that have values on a specified object.

```c
long NspiGetPropList(
    [in] NSPI_HANDLE hRpc,
    [in] DWORD dwFlags,
    [in] DWORD dwMId,
    [in] DWORD CodePage,
    [out] PropertyTagArray_r** ppPropTags
);
```

**hRpc:** An *RPC* context handle, as specified in section 2.2.10.

**dwFlags:** A DWORD [MS-DTYP] value that contains a set of bit flags. The server MUST ignore values other than the bit flag **fSkipObjects**.

**dwMId:** A DWORD value that contains a *Minimal Entry ID*.

**CodePage:** The *code page* in which the client wants the server to express string values properties.

**ppPropTags:** A *PropertyTagArray_r* value. On return, it holds a list of properties.

**Return Values:** The server returns a long value that specifies the return status of the method.

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

**Server Processing Rules:** Upon receiving this message, the server MUST process the data from the message subject to the following constraints:

1. If the server returns any return value other than "Success", the server MUST return a NULL for the output parameter *ppPropTags*. 

2. The server MAY make additional validations, as described in section 5. If the server chooses to limit the visibility of data based on these validations, the server MUST proceed as if that data did not exist in the address book.

3. If the input parameter dwFlags contains the bit flag fSkipObjects, the server MUST NOT return any proptags with the PtypEmbeddedTable property type in the output parameter ppPropTags.

4. The server MUST return all string valued properties as having the PtypString8 property type.

5. Subject to the previous constraints, the server constructs a list of all proptags that correspond to values on the object specified in the input parameter dwMId. The server MUST return this list in the output parameter ppPropTags. The protocol does not constrain the order of this list.

6. If no error condition has been specified by the previous constraints, the server MUST return the value "Success".

3.1.4.1.7 NspiGetProps (Opnum 9)
The NspiGetProps method returns an address book row that contains a set of the properties and values that exist on an object.

```c
long NspiGetProps(
    [in] NSPI_HANDLE hRpc,
    [in] DWORD dwFlags,
    [in] STAT* pStat,
    [in, unique] PropertyTagArray_r* pPropTags,
    [out] PropertyRow_r** ppRows
);
```

hRpc: An RPC context handle, as specified in section 2.2.10.

dwFlags: A DWORD [MS-DTYP] value that contains a set of bit flags. The server MUST ignore values other than the bit flags fEphID and fSkipObjects.

pStat: A pointer to a STAT block that describes a logical position in a specific address book container. This parameter is used to specify input parameters from the client.

pPropTags: The value NULL or a reference to a PropertyTagArray_r value. Contains list of the proptags of the properties that the client wants to be returned.

ppRows: A reference to a PropertyRow_r value. Contains the address book container row the server returns in response to the request.

Return Values: The server returns a long value specifying the return status of the method.

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

Server Processing Rules: Upon receiving this message, the server MUST process the data from the message subject to the following constraints:

1. If the CodePage field of the input parameter pStat is set to the value CP_WINUNICODE and the type of the proptags in the input parameter pPropTags is PtypString8, the server MUST return one of the return values specified in section 2.2.1.2. No further constraints are applied to server processing of this method; in this case server behavior is undefined. Note especially that there is no constraint on the data the server returns in any output parameter other than the return value, nor is there any constraint on how or if the server changes its state.

2. If the server returns any return values other than "ErrorsReturned" (0x00040380) or "Success" (0x00000000), the server MUST return a NULL for the output parameter ppRows.
3. The server MAY make additional validations, as described in section 5. If the server chooses to limit the visibility of data based on these validations, the server MUST proceed as if that data did not exist in the address book.

4. If the server is unable to locate the address book container specified by the ContainerID field in the input parameter pStat, the server MUST return the value "InvalidBookmark" (0x80040405).<5>

5. The server constructs a list of proptags for which it will return property values as follows:
   - If the input parameter pPropTags is not NULL, the client is requesting the server return only those properties and their values in the output parameter ppRows. The server MUST use this list.
   - If the input parameter pPropTags is NULL, the client is requesting that the server constructs a list of proptags on its behalf. The server MUST construct a proptag list that is exactly the same list that would be returned to the client in the pPropTags output parameter of the NspiGetPropList method, as specified in section 3.1.4.1.6, using the following parameters as inputs to the NspiGetPropList method:
     - The NspiGetProps parameter hRpc is used as the NspiGetPropList parameter hRpc.
     - The NspiGetProps parameter dwFlags is used as the NspiGetPropList parameter dwFlags.
     - The CurrentRec field of the NspiGetProps parameter pStat is used as the NspiGetPropList parameter dwMId.
     - The CodePage field of the NspiGetProps parameter pStat is used as the NspiGetPropList parameter CodePage.
   - If a call to the NspiGetPropList method with these parameters and relaxed constraints would return anything other than "Success", the server MUST return that error code as the return value for the NspiGetProps method.

6. If the length of the list of proptags for which the server will return property values is excessive, the server MUST return the return value "TableTooBig", as specified in [MS-OXCDATA] section 2.4. The Exchange Server NSPI Protocol does not prescribe what constitutes an excessive length.

7. If input parameter dwFlags contains the bit flag fEphID and the PidTagEntryId property is present in the list of proptags, the server MUST return the values of the PidTagEntryId property in the Ephemeral Entry ID format, as specified in section 2.2.9.2.

8. If input parameter dwFlags does not contain the bit flag fEphID and the PidTagEntryId property is present in the list of proptags, the server MUST return the values of the PidTagEntryId property in the Permanent Entry ID format, as specified in section 2.2.9.3.

9. The server MUST return string-valued properties in the code page specified in CodePage field of the input parameter pStat, as specified in section 3.1.4.3.

10. If the server can locate the object specified in the CurrentRec field of the input parameter pStat, the server MUST return values associated with this object.

11. If the server is unable to locate the object specified in the CurrentRec field of the input parameter pStat, the server MUST proceed as if the object was located but had no values for any properties.

12. If a property in the proptag list has no value on the object specified by the CurrentRec field, the server MUST return the error code ErrorsReturned. The server MUST set the aulPropTag member corresponding to the proptag with no value with the PtypErrorCode property type. Subject to the prior constraints, the server constructs a list of
properties and their values as a single PropertyRow_r structure with a one-to-one order preserving correspondence between the values in the proptag list specified by input parameters and the returned properties and values in the RowSet. If there are duplicate properties in the proptag list, the server MUST create duplicate values in the parameter RowSet. The server MUST return this RowSet in the output parameter ppRows.

13. If no other return values have been specified by these constraints, the server MUST return the return value "Success".

3.1.4.1.8 NspiQueryRows (Opnum 3)

The NspiQueryRows method returns to the client a number of rows from a specified table. Although the protocol places no boundary or requirements on the minimum number of rows the server returns, implementations SHOULD return as many rows as possible to improve usability of the server for clients.

```c
long NspiQueryRows(
    [in] NSPI_HANDLE hRpc,
    [in] DWORD dwFlags,
    [in, out] STAT* pStat,
    [in, range(0,100000)] DWORD dwETableCount,
    [in, unique, size_is(dwETableCount)]
        DWORD* lpETable,
    [in] DWORD Count,
    [in, unique] PropertyTagArray_r* pPropTags,
    [out] PropertyRowSet_r** ppRows
);
```

hRpc: An RPC context handle, as specified in section 2.2.10.

dwFlags: A DWORD [MS-DTYP] value that contains a set of bit flags. The server MUST ignore values other than the bit flags fEphID and fSkipObjects.

pStat: A pointer to a STAT block that describes a logical position in a specific address book container. This parameter is used to specify both input parameters from the client and return values from the server.

dwETableCount: A DWORD value that contains the number values in the input parameter lpETable. This value is limited to 100,000.

lpETable: An array of DWORD values, representing an Explicit Table, as specified in section 3.1.4.4.2.

Count: A DWORD value that contains the number of rows the client is requesting.

pPropTags: The value NULL or a reference to a PropertyTagArray_r value, containing a list of the proptags of the properties that client requires to be returned for each row returned.

ppRows: A reference to a PropertyRowSet_r value. Contains the address book container rows that the server returns in response to the request.

Return Values: The server returns a long value specifying the return status of the method.

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

Server Processing Rules: Upon receiving this message, the server MUST process the data from the message subject to the following constraints:
1. If the **CodePage** field of the input parameter `pStat` contains the value CP_WINUNICODE, the server MUST return one of the return values specified in section 2.2.1.2. No further constraints are applied to server processing of this method; in this case server behavior is undefined. Note especially that there is no constraint on the data the server returns in any output parameter other than the return value, nor is there any constraint on how or whether the server changes its state.

2. If the input parameter `lpETable` is NULL and the input parameter `Count` is 0, the server MUST return one of the return values specified in section 2.2.1.2. No further constraints are applied to server processing of this method; in this case server behavior is undefined. Note especially that there is no constraint on the data the server returns in any output parameter other than the return value, nor is there any constraint on how or if the server changes its state.

3. If the server returns any return values other than "Success", the server MUST return a NULL for the output parameter `ppRows` and MUST NOT modify the output parameter `pStat`.

4. The server MAY make additional validations, as described in section 5. If the server chooses to limit the visibility of data based on these validations, the server MUST proceed as if that data did not exist in the **address book**.

5. If the input parameter `lpETable` is NULL and the server is unable to locate the address book container specified by the **ContainerID** field in the input parameter `pStat`, the server MUST return the return value "InvalidBookmark".

6. The server constructs a list of proptags for which it will return property values as follows:
   - If the input parameter `pPropTags` is not NULL, the client is requesting the server return only those properties and their values in the output parameter `ppRows`. The server MUST use this list.
   - If the input parameter `pPropTags` is NULL, the client is requesting that the server construct a list of proptags on its behalf. This server MUST use the following proptag list (using proptags defined in [MS-OXPROPS]), in this order: `{**PidTagAddressBookContainerId** ([MS-OXOABK] section 2.2.2.3), **PidTagObjectType** ([MS-OXOABK] section 2.2.3.10), **PidTagDisplayType** ([MS-OXOABK] section 2.2.3.11), **PidTagDisplayName** ([MS-OXOABK] section 2.2.3.1) with the property type **PtypString8**, as specified in [MS-OXCDATA] section 2.11.1, **PidTagPrimaryTelephoneNumber** ([MS-OXOCNTC] section 2.2.1.4.5) with the property type **PtypString8**, **PidTagDepartmentName** ([MS-OXOABK] section 2.2.4.6) with the property type **PtypString8**, **PidTagOfficeLocation** ([MS-OXOABK] section 2.2.4.5) with the property type **PtypString8**}

7. If the input parameter `lpETable` is NULL, the server MUST use the table specified by the input parameter `pStat` when constructing the return parameter `ppRows`.

8. If the input parameter `lpETable` is not NULL, it contains an Explicit Table. The server MUST use that table when constructing the return parameter `ppRows`.

9. The client MUST NOT specify the value 0 for the input parameter `Count` if the input parameter `lpETable` is not NULL.

10. If there are any rows that satisfy the client's query, the server MUST return at least one row.

11. The server MUST return rows in the order they exist in the table being used.

12. If the server is using the table specified by the input parameter `pStat`, the server MUST process rows starting from the current position in the table specified in that parameter (including any values of the **Delta** field).

13. If the server is using the table specified by the input parameter `lpETable`, the server MUST process rows starting from the beginning of the table.
14. The server constructs a **RowSet**. Each row in the **RowSet** corresponds to a row in the table specified by input parameters. The rows in the **RowSet** are in a one-to-one order preserving correspondence with the rows in the table specified by input parameters. The Rows placed into the **RowSet** are exactly those rows that would be returned to the client in the **ppRows** output parameter of the **NsapiGetProps** method, as specified in section 3.1.4.1.7, using the following parameters:

- The **NsapiQueryRows** parameter **hRpc** is used as the **NsapiGetProps** parameter **hRpc**.
- The **NsapiQueryRows** parameter **dwFlags** is used as the **NsapiGetProps** parameter **dwFlags**.
- The **NsapiQueryRows** input parameter **pStat** is used as the **NsapiGetProps** parameter **pStat**. The **CurrentRec** field is set to the **Minimal Entry ID** of the row being returned.
- The list of proptags the server constructs as specified by constraint 6 is used as the **NsapiGetProps** parameter **pPropTags**.
- If a call to the **NsapiGetProps** method with these parameters would return any value other than "Success" or "ErrorsReturned", the server MUST return that error code as the return value for the **NsapiQueryRows** method. Otherwise, the server MUST return the **RowSet** constructed in the output parameter **ppRows**.

15. If the server has no rows that satisfy this query, the server MUST return the value "Success" and place a **PropertyRowSet_r** with rows according to the input parameter "Count" in the output parameter **ppRows**, in which the property type fields of the property are all set to 0x0000000A (PtypErrorCode).

16. If the server is using the table specified by the input parameter **pStat**, the server MUST update the status of the table. This update MUST be exactly the same update that would occur via the **NsapiUpdateStat** method with the following parameters:

- The **NsapiQueryRows** parameter **hRpc** is used as the **NsapiUpdateStat** parameter **hRpc**.
- The value 0 is used as **NsapiUpdateStat** parameter **Reserved**.
- The **NsapiQueryRows** output parameter **pStat** (as modified by the prior constraints) is used as the **NSPIUpdateStat** parameter **pStat**. The number of rows returned in the **NsapiQueryRows** output parameter **ppRows** is added to the **Delta** field.
- The value NULL is used as the **NsapiUpdateStat** parameter **plDelta**.

17. If no other return values have been specified by these constraints, the server MUST return the return value "Success".

### 3.1.4.1.9 NsapiSeekEntries (Opnum 4)

The **NsapiSeekEntries** method searches for and sets the logical position in a specific table to the first entry greater than or equal to a specified value. Optionally, it might also return information about rows in the table.

```c
long NsapiSeekEntries(
    [in] NSPI_HANDLE hRpc,
    [in] DWORD Reserved,
    [in, out] STAT* pStat,
    [in] PropertyValue_r* pTarget,
    [in, unique] PropertyTagArray_r* lpETable,
    [in, unique] PropertyTagArray_r* pPropTags,
    [out] PropertyRowSet_r** ppRows
);
```
**hRpc:** An **RPC** context handle, as specified in section 2.2.10.

**Reserved:** A DWORD [MS-DTYP] value that is reserved for future use. Ignored by the server.

**pStat:** A pointer to a **STAT** block that describes a logical position in a specific **address book container**. This parameter is used to both specify input parameters from the client and return values from the server.

**pTarget:** A **PropertyValue_r** value holding the value that is being sought.

**lpETable:** The value NULL or a **PropertyTagArray_r** value. It holds a list of **Minimal Entry IDs** that comprises a restricted address book container.

**pPropTags:** The value NULL or a reference to a **PropertyTagArray_r** value. Contains list of the proptags of the columns that client wants to be returned for each row returned.

**ppRows:** A reference to a **PropertyRowSet_r** value. Contains the address book container rows the server returns in response to the request.

**Return Values:** The server returns a long value specifying the return status of the method.

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

**Server Processing Rules:** Upon receiving this message, the server MUST process the data from the message subject to the following constraints:

1. If the **CodePage** field of the input parameter **pStat** contains the value CP_WINUNICODE, the server MUST return one of the return values specified in section 2.2.1.2. No further constraints are applied to server processing of this method; in this case server behavior is undefined. Note especially that there is no constraint on the data the server returns in any output parameter other than the return value, nor is there any constraint on how or whether the server changes its state.

2. If the input parameter **lpETable** is not NULL and does not contain an Explicit Table both containing a restriction of the table specified by the input parameter **pStat** and sorted as specified by the **SortType** field of the input parameter **pStat**, the server MUST return one of the return values specified in section 2.2.1.2. No further constraints are applied to server processing of this method; in this case server behavior is undefined. Note especially that there is no constraint on the data the server returns in any output parameter other than the return value, nor is there any constraint on how or whether the server changes its state.

3. If the input parameter **Reserved** contains any value other than 0, the server MUST return one of the return values specified in section 2.2.1.2. No further constraints are applied to server processing of this method; in this case server behavior is undefined. Note especially that there is no constraint on the data the server returns in any output parameter other than the return value, nor is there any constraint on how or if the server changes its state.

4. If the server returns any return values other than "Success", the server MUST return a NULL for the output parameter **ppRows** and MUST NOT modify the value of the parameter **pStat**.

5. The server MAY make additional validations as described in section 5. If the server chooses to limit the visibility of data based on these validations, the server MUST proceed as if that data did not exist in the **address book**.

6. If the server is unable to locate the address book container specified by the **ContainerID** field in the input parameter **pStat**, the server MUST return the return value "InvalidBookmark".

7. If the input parameter **lpETable** is NULL, the server MUST use the table specified by the input parameter **pStat** when constructing the return parameter **ppRows**.
8. If the input parameter `lpETable` contains an Explicit Table, the server MUST use that table when constructing the return parameter `ppRows`.

9. If the `SortType` field in the input parameter `pStat` has any value other than `SortTypeDisplayName`, the server MUST return the value `GeneralFailure`.

10. If the `SortType` field in the input parameter `pStat` is `SortTypeDisplayName` and the property specified in the input parameter `pTarget` is anything other than `PidTagDisplayName` (with either the Property Type `PtypString8` or `PtypString`), the server MUST return the value `GeneralFailure`.

11. The server MUST locate the first row in the specified table that has a value equal to or greater than the value specified in the input parameter `pTarget`. If no such row exists, the server MUST return the value `NotFound`.

12. If a qualifying row was found, the server MUST update the position information in the parameter `pStat`.
   - The server MUST set `CurrentRec` field of the parameter `pStat` to the Minimal Entry ID of the qualifying row.
   - If the server is using the table specified by the input parameter `lpETable`, the server MUST set the `NumPos` field of the parameter `pStat` to the accurate numeric position of the qualifying row in the table.
   - If the server is using the table specified by the input parameter `pStat`, the server MUST set the `NumPos` field of the parameter `pStat` to the approximate numeric position of the qualifying row in the table.
   - The `TotalRecs` field of the parameter `pStat` MUST be set to the accurate number of records in the table used.
   - The server MUST NOT modify any other fields of the parameter `pStat`.

13. If the input parameter `pPropTags` is not NULL, the client is requesting the server to return an `PropertyRowSet_r`. Subject to the prior constraints, the server MUST construct an `PropertyRowSet_r` to return to the client in the output parameter `ppRows`. The server MUST return the same `PropertyRowSet_r` for the `NsapiSeekEntries` method after the `NsapiQueryRows` method is called using the input parameters. This `PropertyRowSet_r` MUST be exactly the same `PropertyRowSet_r` that would be returned in the `ppRows` parameter of a call to the `NsapiQueryRows` method with the following parameters:
   - The `NsapiSeekEntries` parameter `hRpc` is used as the `NsapiQueryRows` parameter `hRpc`.
   - The value `fEphID` is used as the `NsapiQueryRows` parameter `dwFlags`.
   - The `NsapiSeekEntries` output parameter `pStat` (as modified by the prior constraints) is used as the `NsapiQueryRows` parameter `pStat`.
   - If the `NsapiSeekEntries` input parameter `lpETable` is NULL, the value 0 is used as the `NsapiQueryRows` parameter `dwETableCount`, and the value NULL is used as the `NsapiQueryRows` parameter `lpETable`.
   - If the `NsapiSeekEntries` input parameter `lpETable` is not NULL, the server constructs an explicit table from the table specified by `lpETable` by copying rows in order from `lpETable` to the new explicit table. The server begins copying from the row specified by the `NumPos` field of the `pStat` parameter (as modified by the prior constraints), and continues until all remaining rows are added to the new table. The number of rows in this new table is used as the `NsapiQueryRows` parameter `dwETableCount`, and the new table is used as the `NsapiQueryRows` `lpETable` parameter.
The list of Minimal Entry IDs in the input parameter \textit{lpETable} starting with the qualifying row is used as the \textbf{NsipiQueryRows} parameter \textit{lpETable}. These Minimal Entry IDs are expressed as a simple array of DWORD values rather than as a PropertyTagArray\_r value. Note that the qualifying row is included in this list, and that the order of the Minimal Entry IDs from the input parameter \textit{lpETable} is preserved in this list.

If the \textbf{NsipiSeekEntries} input parameter \textit{lpETable} is NULL, the server MUST choose a value for the \textbf{NsipiQueryRows} parameter \textit{Count}. The Exchange Server NSPI Protocol does not prescribe any particular algorithm. The server MUST use a value greater than 0.

If the \textbf{NsipiSeekEntries} input parameter \textit{lpETable} is not NULL, the value used for the \textbf{NsipiQueryRows} parameter \textit{dwETableCount} is used for the \textbf{NsipiQueryRows} parameter \textit{Count}.

The \textbf{NsipiSeekEntries} parameter \textit{pPropTags} is used as the \textbf{NsipiQueryRows} parameter \textit{pPropTags}.

Note that the server MUST NOT modify the return value of the \textbf{NsipiSeekEntries} output parameter \textit{pStat} in any way in the process of constructing the output PropertyRowSet\_r.

14. If no other return values have been specified by these constraints, the server MUST return the return value "Success".

\subsection{NsipiGetMatches (Opnum 5)}

The \textbf{NsipiGetMatches} method returns an Explicit Table. The rows in the table are chosen based on two possible criteria: a restriction applied to an address book container or the values of a property on a single object that hold references to other objects.

```c
long NsipiGetMatches(
    [in] NSPI\_HANDLE hRpc,
    [in] DWORD Reserved1,
    [in, out] STAT* pStat,
    [in, unique] PropertyTagArray\_r* pReserved,
    [in] DWORD Reserved2,
    [in, unique] Restriction\_r* Filter,
    [in, unique] PropertyName\_r* lpPropName,
    [in] DWORD ulRequested,
    [out] PropertyTagArray\_r** ppOutMIds,
    [in, unique] PropertyTagArray\_r* pPropTags,
    [out] PropertyRowSet\_r** ppRows
);
```

\textbf{hRpc}: An RPC context handle, as specified in section \ref{sec:rpc-context}.

\textbf{Reserved1}: A DWORD [MS-DTYP] value reserved for future use.

\textbf{pStat}: A reference to a STAT block describing a logical position in a specific address book container.

\textbf{pReserved}: A PropertyTagArray\_r reserved for future use.

\textbf{Reserved2}: A DWORD value reserved for future use. Ignored by the server.

\textbf{Filter}: The value NULL or an Restriction\_r value. Holds a logical restriction to apply to the rows in the address book container specified in the \textit{pStat} parameter.

\textbf{lpPropName}: The value NULL or a PropertyName\_r value. Holds the property to be opened as a restricted address book container.
ulRequested: A DWORD value. Contains the maximum number of rows to return in a restricted address book container.

ppOutMIds: A PropertyTagArray_r value. On return, it holds a list of Minimal Entry IDs that comprise a restricted address book container.

pPropTags: The value NULL or a reference to a PropertyTagArray_r value. Contains list of the proptags of the columns that client wants to be returned for each row returned.

ppRows: A reference to a PropertyRowSet_r value. Contains the address book container rows the server returns in response to the request.

Return Values: The server returns a long value specifying the return status of the method.

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

Server Processing Rules: Upon receiving this message, the server MUST process the data from the message subject to the following constraints:

1. If the CodePage field of the input parameter pStat contains the value CP_WINUNICODE, the server MUST return one of the return values specified in section 2.2.1.2. No further constraints are applied to server processing of this method; in this case server behavior is undefined. Note especially that there is no constraint on the data the server returns in any output parameter other than the return value, nor is there any constraint on how or if the server changes its state.

2. If the input parameter Filter contains any value other than NULL and the SortType field of the input parameter pStat contains any value other than SortTypeDisplayName or SortTypePhoneticDisplayName, the server MUST return one of the return values specified in section 2.2.1.2. No further constraints are applied to server processing of this method; in this case server behavior is undefined. Note especially that there is no constraint on the data the server returns in any output parameter other than the return value, nor is there any constraint on how or if the server changes its state.

3. If the input parameter Reserved1 contains any value other than 0, the server MUST return one of the return values specified in section 2.2.1.2. No further constraints are applied to server processing of this method; in this case server behavior is undefined. Note especially that there is no constraint on the data the server returns in any output parameter other than the return value, nor is there any constraint on how or if the server changes its state.

4. If the server returns any return values other than "Success", the server MUST return a NULL for the output parameters ppOutMIds and ppRows and MUST NOT modify the value of the parameter pStat.

5. The server MAY make additional validations as described in section 5. If the server chooses to limit the visibility of data based on these validations, the server MUST proceed as if that data did not exist in the address book.

6. If the input SortType field of the input parameter pStat is SortTypeDisplayName or SortTypePhoneticDisplayName and the server is unable to locate the address book container specified by the ContainerID field in the input parameter pStat, the server MUST return the return value "InvalidBookmark".

7. If the input parameter Filter is not NULL, the server constructs an Explicit Table as follows:

   - If the input parameter Filter is not NULL, it specifies a restriction, as specified in [MS-OXCDATA].
- If the server will not support the call because the search is too complex, the server MUST return the value "TooComplex". The Exchange Server NSPI Protocol does not prescribe what constitutes a search that is too complex.

- If the server will support the filter, it identifies the rows in the table specified in the input parameter pStat for which the filter is true. The Minimal IDs of these rows are inserted into the Explicit Table, maintaining their order from the originating table.

8. If the input parameter Filter is NULL, the server constructs an Explicit Table as follows:

- The Minimal Entry ID of the object the server is to read values from is specified in the CurrentRec field of the input parameter pStat. The server MUST ignore any values of the Delta and ContainerID fields while locating the object. That is, the server MUST NOT enforce any restrictions that the object specified by CurrentRec is actually in any particular address book container. Note that this is an exceptional use of the pStat parameter for position, not conforming to the semantic meaning of this field in all other NSPI methods.

- If the input parameter lpPropName is not NULL, it specifies the property the server is to read the values of. If the input parameter lpPropName is NULL, the server is to read the values of the property specified as a proptag value in the ContainerID field of the input parameter pStat. Note, this is an exceptional use of this field, not conforming to the semantic meaning of this field in all other NSPI methods.

11. The server locates the object specified by the client, subject to these restraints. If the server is unable to locate the object, the server MUST return the value "GeneralFailure".

12. If the SortType field of the input parameter pStat has the value SortTypeDisplayName_W and the server does not support modifying the value of the property specified by the client on the object specified by the client, the server MUST return the value "NotSupported".

13. If the server is unable to locate objects in the address book based on values of the property specified by the client on the object specified by the client, the server MUST return the value "NotSupported". Note that this constraint is intended to apply in the case where the server is categorically unable to locate specific objects based on the value of the property, not the case where the property has no values.

14. The server reads the values of the property specified by the client. For each value read, the server tries to locate a specific object in the address book corresponding to this value. If a specific object is located, the Minimal ID of the object is inserted into the Explicit Table.

15. The server MUST sort the rows in Explicit Table by the Unicode representation of the PidTagDisplayName property, as specified in section 3.1.4.3).

16. If the server returns "Success", the server MUST set the ContainerID field of the output parameter pStat to be equal to the CurrentRec field of the input parameter pStat. The server MUST NOT modify any other fields in this parameter.

17. If the number of rows in the constructed Explicit Table is greater than the input parameter ulRequested, the server MUST return the value "TableTooBig".

18. If the server will not support the call because the Explicit Table is larger than the server will allow, the server MUST return the value "TableTooBig". The Exchange Server NSPI Protocol does not prescribe what constitutes a table that is too large.

19. If the input parameter proptags is not NULL, the client is requesting the server to return a PropertyRowSet_r. Subject to the prior constraints, the server MUST construct a PropertyRowSet_r to return to the client in the output parameter ppRows. This PropertyRowSet_r MUST be exactly the same PropertyRowSet_r that would be returned in the ppRows parameter of a call to the NspiQueryRows method with the following parameters:
The `NsSpiGetMatches` parameter `hRpc` is used as the `NsSpiQueryRows` parameter `hRpc`.

The value "fEphID" is used as the `NsSpiQueryRows` parameter `dwFlags`.

The `NsSpiGetMatches` output parameter `pStat` (as modified by the prior constraints) is used as the `NsSpiQueryRows` parameter `pStat`.

The number of Minimal Entry IDs in the constructed Explicit Table is used as the `NsSpiQueryRows` parameter `dwETableCount`.

The constructed Explicit Table is used as the `NsSpiQueryRows` parameter `lpETable`. These Minimal Entry IDs are expressed as a simple array of DWORD values rather than as a `PropertyTagArray_r` value.

The number of Minimal Entry IDs in the constructed Explicit Table is used as the `NsSpiQueryRows` parameter `Count`.

The `NsSpiGetMatches` parameter `proptags` is used as the `NsSpiQueryRows` parameter `proptags`.

Note that the server MUST NOT modify the return value of the `NsSpiGetMatches` method output parameter `pStat` in any way in the process of constructing the output `PropertyRowSet_r`. The server MUST return the constructed `PropertyRowSet_r` in the output parameter `ppRows`.

20. If no other return values have been specified by these constraints, the server MUST return the return value "Success".

### 3.1.4.1.11 `NsSpiResortRestriction` (Opnum 6)

The `NsSpiResortRestriction` method applies a sort order to the objects in a restricted address book container.

```c
long NspiResortRestriction(  
    [in] NSPI_HANDLE hRpc,  
    [in] DWORD Reserved,  
    [in, out] STAT* pStat,  
    [in] PropertyTagArray_r* pInMIds,  
    [in, out] PropertyTagArray_r** ppOutMIds
);
```

**hRpc**: An RPC context handle, as specified in section 2.2.10.

**Reserved**: A DWORD [MS-DTYP] value reserved for future use. Ignored by the server.

**pStat**: A reference to a STAT block describing a logical position in a specific address book container.

**pInMIds**: A `PropertyTagArray_r` value. It holds a list of Minimal Entry IDs that comprise a restricted address book container.

**ppOutMIds**: A `PropertyTagArray_r` value. The server MUST ignore this parameter in the request. On return, it holds a list of Minimal Entry IDs that comprise a restricted address book container.

**Return Values**: The server returns a long value that specifies the return status of the method.

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

**Server Processing Rules**: Upon receiving this message, the server MUST process the data from the message subject to the following constraints:
1. If the **CodePage** field of the input parameter **pStat** contains the value CP_WINUNICODE, the server MUST return one of the return values specified in section 2.2.1.2. No further constraints are applied to server processing of this method; in this case server behavior is undefined. Note especially that there is no constraint on the data the server returns in any output parameter other than the return value, nor is there any constraint on how or if the server changes its state.

2. If the **SortType** field of the input parameter **pStat** contains any value other than "SortTypeDisplayName" or "SortTypePhoneticDisplayName", the server MUST return one of the return values specified in section 2.2.1.2. No further constraints are applied to server processing of this method; in this case server behavior is undefined. Note especially that there is no constraint on the data the server returns in any output parameter other than the return value, nor is there any constraint on how or if the server changes its state.

3. If the server returns any return values other than "Success", the server MUST return a NULL for the output parameter **ppOutMIds** and MUST NOT modify the value of the parameter **pStat**.

4. The server MAY make additional validations as described in section 5. If the server chooses to limit the visibility of data based on these validations, the server MUST proceed as if that data did not exist in the address book.

5. If the **SortType** field in the input parameter **pStat** has any value other than **SortTypeDisplayName**, the server SHOULD return the value "GeneralFailure".

6. The server constructs an Explicit Table as follows:
   - The server locates all the objects specified in the Explicit Table specified by the input value **pInMIds**. The server MUST ignore any Minimal Entry IDs that do not specify an object.
   - For each such object located, a row is inserted into the constructed Explicit Table.
   - The server MUST sort the rows in the constructed explicit table by the property specified in the **SortType** field of the input parameter **pStat**.

7. The server MUST return the constructed Explicit Table in the output parameter **ppOutMIds**.

8. The server MUST update the output parameter **pStat** as follows:
   - The **TotalRecs** field is set to the number of objects in the constructed Explicit Table.
   - If the object specified by the **CurrentRec** field of the input parameter **pStat** is not in the constructed Explicit Table, the **CurrentRec** field of the output parameter **pStat** is set to the value MID_BEGINNING_OF_TABLE and the **NumPos** field of the output parameter **pStat** is set to the value 0.
   - The server MUST NOT modify any other fields of the output parameter **pStat**.

9. If no other return values have been specified by these constraints, the server MUST return the return value "Success".

### 3.1.4.1.12 NspiCompareMIds (Opnum 10)

The **NspiCompareMIds** method compares the position in an address book container of two objects identified by Minimal Entry ID and returns the value of the comparison.

```c
long NspiCompareMIds(
    [in] NSPI_HANDLE hRpc,
    [in] DWORD Reserved,
    [in] STAT* pStat,
    [in] DWORD MId1,
    [in] DWORD MId2,
    [out] long* plResult
```
hRpc: An RPC context handle, as specified in section 2.2.10.

Reserved: A DWORD [MS-DTYP] value reserved for future use. Ignored by the server.

pStat: A reference to a STAT block that describes a logical position in a specific address book container.

MId1: A DWORD value containing a Minimal Entry ID.

MId2: A DWORD value containing a Minimal Entry ID.

plResult: A pointer to a long value which specifies the compare result of the NspiCompareMids method.

Return Values: The server returns a long value that specifies the return status of the method.

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

Server Processing Rules: Upon receiving this message, the server MUST process the data from the message subject to the following constraints:

1. If the CodePage field of the input parameter pStat contains the value CP_WINUNICODE, the server MUST return one of the return values specified in section 2.2.1.2. No further constraints are applied to server processing of this method; in this case server behavior is undefined. Note especially that there is no constraint on the data the server returns in any output parameter other than the return value, nor is there any constraint on how or if the server changes its state.

2. The server MAY make additional validations as described in section 5. If the server chooses to limit the visibility of data based on these validations, the server MUST proceed as if that data did not exist in the address book.

3. If the server is unable to locate the address book container specified by the ContainerID field in the input parameter pStat, the server MUST return the return value "InvalidBookmark".

4. If the server returns any return value other than "Success", the protocol does not constrain the value in the return parameter plResult.

5. If the server is unable to locate the objects specified by the input parameters MId1 or MId2 in the table specified by the ContainerID field of the input parameter pStat, the server MUST return the return value "GeneralFailure".

6. If the position of the object specified by MId1 comes before the position of the object specified by MId2 in the table specified by the field ContainerID of the input parameter pStat, the server MUST return a value less than 0 in the output parameter plResult.

7. If the position of the object specified by MId1 comes after the position of the object specified by MId2 in the table specified by the field ContainerID of the input parameter pStat, the server MUST return a value greater than 0 in the output parameter plResult.

8. If the position of the object specified by MId1 is the same as the position of the object specified by MId2 in the table specified by the ContainerID field of the input parameter pStat (that is, they specify the same object), the server MUST return a value of 0 in the output parameter plResult.

9. If no other return values have been specified by these constraints, the server MUST return the return value "Success".
3.1.4.1.13  NspiDNToMId (Opnum 7)

The NspiDNToMId method maps a set of DNs to a set of Minimal Entry ID.

```c
long NspiDNToMId(
    [in] NSPI_HANDLE hRpc,
    [in] DWORD Reserved,
    [in] StringsArray_r* pNames,
    [out] PropertyTagArray_r** ppMIds
);```

hRpc: An RPC context handle, as specified in section 2.2.10.

Reserved: A DWORD [MS-DTYP] value reserved for future use. Ignored by the server.

pNames: A StringsArray_r value. It holds a list of strings that contain DNs, as specified in [MS-OXOABK].

ppMIds: A PropertyTagArray_r value. On return, it holds a list of Minimal Entry IDs.

Return Values: The server returns a long value that specifies the return status of the method.

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

Server Processing Rules: Upon receiving this message, the server MUST process the data from the message subject to the following constraints:

1. If the server returns any return value other than "Success", the server MUST return the value NULL in the return parameter ppMIds.

2. The server MAY make additional validations as described in section 5). If the server chooses to limit the visibility of data based on these validations, the server MUST proceed as if that data did not exist in the address book.

3. If the server is unable to locate an appropriate mapping between a DN and a Minimal Entry ID, it MUST map the DN to a Minimal Entry ID with the value 0.

4. The server constructs a list of Minimal Entry IDs to return to the client, encoding the mappings. The list is in a one-to-one order preserving correspondence with the list of DNs in the input parameter pNames. The server MUST return the list in the output parameter ppMIds.

5. If no other return values have been specified by these constraints, the server MUST return the return value "Success".

3.1.4.1.14  NspiModProps (Opnum 11)

The NspiModProps method is used to modify the properties of an object in the address book. This protocol supports the PidTagUserX509Certificate ([MS-OXPROPS] section 2.1054) and PidTagAddressBookX509Certificate ([MS-OXPROPS] section 2.575) properties.

```c
long NspiModProps(
    [in] NSPI_HANDLE hRpc,
    [in] DWORD Reserved,
    [in] STAT* pStat,
    [in, unique] PropertyTagArray_r* pPropTags,
    [in] PropertyRow_r* pRow
);```
hRpc: An RPC context handle, as specified in section 2.2.10.

Reserved: A DWORD [MS-DTYP] value reserved for future use.

pStat: A reference to a STAT block that describes a logical position in a specific address book container.

pPropTags: The value NULL or a reference to a PropertyTagArray_r. Contains list of the proptags of the columns that client requests all values to be removed from.

pRow: A PropertyRow_r value. Contains an address book row.

Return Values: The server returns a long value that specifies the return status of the method.

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

Server Processing Rules: Upon receiving this message, the server MUST process the data from the message subject to the following constraints:

1. If the CodePage field of the input parameter pStat contains the value CP_WINUNICODE, the server MUST return one of the return values specified in section 2.2.1.2. No further constraints are applied to server processing of this method; in this case server behavior is undefined. Note especially that there is no constraint on the data the server returns in any output parameter other than the return value, nor is there any constraint on how or if the server changes its state.

2. If the server returns any return value other than "Success", the server MUST NOT modify any properties of any objects in the address book.

3. The server MAY make additional validations, as described in section 5. If the server chooses to limit the visibility of data based on these validations, the server MUST proceed as if that data did not exist in the address book.

4. If the Reserved input parameter contains any value other than 0, the server MUST ignore the value.

5. If the input parameter pPropTags is NULL, the server MUST return the value "InvalidParameter".

6. If the server is unable to locate the object specified by the CurrentRec field of the input parameter pStat, the server MUST return the value "InvalidParameter".

7. If the server is able to locate the object, but will not allow modifications to the object due to its display type, the server MUST return the value "InvalidObject".

8. The server MUST remove all values for all properties specified in the input parameter pPropTags from the object specified by the field CurrentRec in the input parameter pStat.

9. The server MUST remove all values for all properties specified in the input parameter pRow from the object specified by the field CurrentRec in the input parameter pStat.

10. The server SHOULD<7> add all values for all properties specified in the input parameter pRow to the object specified by the field CurrentRec in the input parameter pStat.

11. If the server is unable to apply the modifications specified for any other reason, the server MUST return the value "AccessDenied".

12. If no other return values have been specified by these constraints, the server MUST return the return value "Success".

3.1.4.1.15 NspiModLinkAtt (Opnum 14)
The **NspiModLinkAtt** method modifies the values of a specific property of a specific row in the address book. This protocol only supports modifying the value of the **PidTagAddressBookMember** property ([MS-OXOABK] section 2.2.6.1) of an address book object with display type DT_DISTLIST and the **PidTagAddressBookPublicDelegates** property ([MS-OXOABK] section 2.2.5.5) of an address book object with display type DT_MAILUSER.

```c
long NspiModLinkAtt(
    [in] NSPI_HANDLE hRpc,
    [in] DWORD dwFlags,
    [in] DWORD ulPropTag,
    [in] DWORD dwMId,
    [in] BinaryArray_r* lpEntryIds
);
```

**hRpc**: An RPC context handle, as specified in section 2.2.10.

**dwFlags**: A DWORD [MS-DTYP] value that contains a set of bit flags. The server MUST ignore values other than the bit flag **fDelete**.

**ulPropTag**: A DWORD value. Contains the proptag of the property that the client wants to modify.

**dwMId**: A DWORD value that contains the Minimal Entry ID of the address book row that the client wants to modify.

**lpEntryIds**: A BinaryArray_r value. Contains a list of **EntryIDs** to be used to modify the requested property on the requested address book row. These EntryIDs can be either Ephemeral Entry IDs or Permanent Entry IDs or both.

**Return Values**: The server returns a long value that specifies the return status of the method.

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

**Server Processing Rules**: Upon receiving this message, the server MUST process the data from the message subject to the following constraints:

1. If the server returns any return value other than Success (0x00000000), the server MUST NOT modify any properties of any objects in the address book.<8>

2. The server MAY make additional validations, as described in section 5). If the server chooses to limit the visibility of data based on these validations, the server MUST proceed as if that data did not exist in the address book.

3. If the input parameter **ulPropTag** does not specify a proptag the server recognizes, the server MUST return **NotFound**.

4. If the server is unable to locate the object specified by the input parameter **dwMId**, the server MUST return the value InvalidParameter (0x80070057).

5. If the server is able to locate the object, but will not allow modifications to the object due to its display type, the server MUST NOT modify any properties of any objects in the address book, and the server MUST return the value AccessDenied (0x80070005).

6. If the input parameter **dwFlags** contains the bit value **fDelete**, the server MUST remove all values specified by the input parameter **lpEntryIds** from the property specified by **ulPropTag** for the object specified by input parameter **dwMId**. The server MUST ignore any values specified by **lpEntryIds** that are not present on the object specified by **dwMId**.

7. If the input parameter **dwFlags** does not contain the bit value **fDelete**, the server MUST add all values specified by the input parameter **lpEntryIds** to the property specified by **ulPropTag** for the
object specified by the input parameter \textit{dwMId}. The server MUST ignore any values specified by \textit{lpEntryIDs} that are already present on the object specified by \textit{dwMId}.

8. If the server is unable to apply the modifications specified, the server MUST return the value AccessDenied (0x80070005).

9. If no other return values have been specified by these constraints, the server MUST return the return value Success (0x00000000).

3.1.4.1.16 \textbf{NspiResolveNames (Opnum 19)}

The \textit{NspiResolveNames} method takes a set of string values in an 8-bit character set and performs ANR (as specified in section 3.1.4.7) on those strings. The server reports the Minimal Entry ID that is the result of the ANR process. Certain property values are returned for any valid Minimal Entry IDs identified by the ANR process.

```c
long NspiResolveNames(
    [in] NSPI_HANDLE hRpc,
    [in] DWORD Reserved,
    [in] STAT* pStat,
    [in, unique] PropertyTagArray_r* pPropTags,
    [in] StringsArray_r* paStr,
    [out] PropertyTagArray_r** ppMIds,
    [out] PropertyRowSet_r** ppRows)
```

\textbf{hRpc}: An RPC context handle, as specified in section 2.2.10.

\textbf{Reserved}: A DWORD [MS-DTYP] reserved for future use.

\textbf{pStat}: A reference to a STAT block that describes a logical position in a specific \textit{address book container}.

\textbf{pPropTags}: The value NULL or a reference to a PropertyTagArray\_r value containing a list of the proptags of the columns that the client requests to be returned for each row returned.

\textbf{paStr}: A StringsArray\_r value. Specifies the values the client is requesting the server to do ANR on. The server MUST apply any necessary character set conversion as specified in section 3.1.4.3.

\textbf{ppMIds}: A PropertyTagArray\_r value. On return, contains a list of Minimal Entry IDs that match the array of strings, as specified in the input parameter \textit{paStr}.

\textbf{ppRows}: A reference to a PropertyRowSet\_r structure (section 2.2.4), which contains the address book container rows that the server returns in response to the request.

\textbf{Return Values}: The server returns a long value that specifies the return status of the method.

\textbf{Exceptions Thrown}: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [MS-RPCE].

\textbf{Server Processing Rules}: Upon receiving this message, the server MUST process the data from the message subject to the following constraints:

1. If the \textit{CodePage} field of the input parameter \textit{pStat} contains the value CP\_WINUNICODE, the server MUST return one of the return values specified in section 2.2.1.2. No further constraints are applied to server processing of this method; in this case server behavior is undefined. Note especially that there is no constraint on the data the server returns in any output parameter other than the return value, nor is there any constraint on how or if the server changes its state.
2. If the input parameter *Reserved* contains any value other than 0, the server MUST return one of the return values specified in section 2.2.1.2. No further constraints are applied to server processing of this method; in this case server behavior is undefined. Note especially that there is no constraint on the data the server returns in any output parameter other than the return value, nor is there any constraint on how or if the server changes its state.

3. If the server returns any return value other than "Success", the server MUST return the value NULL in the return parameters *ppMIds* and *ppRows*.

4. The server MAY make additional validations, as described in section 5. If the server chooses to limit the visibility of data based on these validations, the server MUST proceed as if that data did not exist in the *address book*.

5. If the server is unable to locate the address book container specified by the *ContainerID* field in the input parameter *pStat*, the server MUST return the return value "InvalidBookmark".

6. The server constructs a list of the Minimal Entry IDs specified in section 2.2.1.9 to return to the client. These Minimal Entry IDs are those that result from applying the ANR process, as specified in section 3.1.4.7, to the strings in the input parameter *paStr*. The server MUST return this list of Minimal Entry IDs in the output parameter *ppMIds*.

7. Subject to the prior constraints, the server MUST construct a *PropertyRowSet_r* structure to return to the client.

8. If no other return values have been specified by these constraints, the server MUST return the return value "Success".

### 3.1.4.1.17 NspiResolveNamesW (Opnum 20)

The *NspiResolveNamesW* method takes a set of string values in the *Unicode* character set and performs ANR (as specified in section 3.1.4.7) on those strings. The server reports the Minimal Entry IDs that are the result of the ANR process. Certain property values are returned for any valid Minimal Entry IDs identified by the ANR process.

```c
long NspiResolveNamesW(
    [in] NSPI_HANDLE hRpc,
    [in] DWORD Reserved,
    [in] STAT* pStat,
    [in, unique] PropertyTagArray_r* pPropTags,
    [in] WStringsArray_r* paWStr,
    [out] PropertyTagArray_r** ppMIds,
    [out] PropertyRowSet_r** ppRows
);
```

**hRpc**: An *RPC* context handle, as specified in section 2.2.10.

**Reserved**: A DWORD [*MS-DTYP*] value that is reserved for future use.

**pStat**: A reference to a STAT block that describes a logical position in a specific *address book container*.

**pPropTags**: The value NULL or a reference to a *PropertyTagArray_r* containing a list of the proptags of the columns that the client requests to be returned for each row returned.

**paWStr**: A *WStringsArray_r* value. Specifies the values on which the client is requesting that the server perform ANR. The server MUST apply any necessary character set conversion, as specified in section 3.1.4.3.

**ppMIds**: A *PropertyTagArray_r* value. On return, contains a list of Minimal Entry IDs that match the array of strings, as specified in the input parameter *paWStr*.
ppRows: A reference to a PropertyRowSet_r structure (section 2.2.4), which contains the address book container rows that the server returns in response to the request.

Return Values: The server returns a long value that specifies the return status of the method.

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

Server Processing Rules: Upon receiving this message, the server MUST process the data from the message subject to the following constraints:

1. If the CodePage field of the input parameter pStat contains the value CP_WINUNICODE, the server MUST return one of the return values specified in section 2.2.1.2. No further constraints are applied to server processing of this method; in this case server behavior is undefined. Note especially that there is no constraint on the data the server returns in any output parameter other than the return value, nor is there any constraint on how or if the server changes its state.

2. If the input parameter Reserved contains any value other than 0, the server MUST return one of the return values specified in section 2.2.1.2. No further constraints are applied to server processing of this method; in this case server behavior is undefined. Note especially that there is no constraint on the data the server returns in any output parameter other than the return value, nor is there any constraint on how or if the server changes its state.

3. If the server returns any return value other than "Success", the server MUST return the value NULL in the return parameters ppMIds and ppRows.

4. The server MAY make additional validations, as described in section 5. If the server chooses to limit the visibility of data based on these validations, the server MUST proceed as if that data did not exist in the address book.

5. If the server is unable to locate the address book container specified by the ContainerID field in the input parameter pStat, the server MUST return the return value "InvalidBookmark".

6. The server constructs a list of the Minimal Entry IDs specified in section 2.2.1.9 to return to the client. These Minimal Entry IDs are those that result from the ANR process, as specified in section 3.1.4.7, to the strings in the input parameter paWStr. The server MUST return this list of Minimal Entry IDs in the output parameter ppMIds.

7. Subject to the prior constraints, the server MUST construct a PropertyRowSet_r structure to return to the client.

8. If no other return values have been specified by these constraints, the server MUST return the return value "Success".

3.1.4.1.18 NspiGetTemplateInfo (Opnum 13)

The NspiGetTemplateInfo method returns information about template objects in the address book.

```c
long NspiGetTemplateInfo(
    [in] NSPI_HANDLE hRpc,
    [in] DWORD dwFlags,
    [in] DWORD ulType,
    [string, in, unique] char* pDN,
    [in] DWORD dwCodePage,
    [in] DWORD dwLocaleID,
    [out] PropertyRow_r** ppData
);
```

hRpc: An RPC context handle, as specified in section 2.2.10.
**dwFlags:** A DWORD [MS-DTYP] value that contains a set of bit flags. The server MUST ignore values other than the bit flags **TI_EMT**, **TI_SCRIPT** and **TI_TEMPLATE**.

**ulType:** A DWORD value. Specifies the display type of the template for which information is requested.

**pDN:** The value NULL or the DN of the template requested. The value is NULL-terminated.

**dwCodePage:** A DWORD value. Specifies the code page of the template for which information is requested.

**dwLocaleID:** A DWORD value. Specifies the LCID, as specified in [MS-LCID], of the template for which information is requested.

**ppData:** A reference to a PropertyRow_r value. On return, it contains the information requested.

**Return Values:** The server returns a long value that specifies the return status of the method.

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

**Server Processing Rules:** Upon receiving this message, the server MUST process the data from the message subject to the following constraints:

1. If the server returns any return value other than "Success", the server MUST return the value NULL in the return parameters ppData.

2. The server MAY make additional validations, as described in section 5. If the server chooses to limit the visibility of data based on these validations, the server MUST proceed as if that data did not exist in the address book.

3. If the codepage specified in the dwCodePage input parameter has the value CP_WINUNICODE, the server MUST return the value "InvalidCodePage".

4. If the server does not recognize the codepage specified in the dwCodePage input parameter as a supported code page, the server MUST return the value "InvalidCodePage".

5. The server locates the template for which it will return information as follows:

   - If the input parameter pDN is NULL, the server MUST choose an appropriate template object for the display type specified by the input parameter ulType and for the LCID specified by the input parameter dwLocaleID. The specific choice of an appropriate template object is defined by local policy, and is not constrained by the Exchange Server NSPI Protocol. For details, see [MS-OXOABKT].

   - If the input parameter pDN is not NULL, it specifies the DN of a template object in the address book. In this case, the server MUST ignore the input parameters ulType and dwLocaleID.

   - If the server is unable to locate a specific object based on these constraints, the server MUST return the value "InvalidLocale".

6. The server constructs a PropertyRow_r value. The property values in this PropertyRow_r are specified as follows:

   - If the input parameter dwFlags has the **TI_SCRIPT** bit set, the client is requesting the script data for the template, as specified in [MS-OXOABKT]. The server MUST place this data into the PropertyRow_r structure.
If the input parameter \textit{dwFlags} has the \texttt{TI_TEMPLATE} bit set, the client is requesting the user interface data for the template, as specified in the [MS-OXOABKT]. The server MUST place this data into the PropertyRow_r structure.

7. The server MUST return the constructed PropertyRow_r structure in the output parameter \textit{ppData}.

8. If no other return values have been specified by these constraints, the server MUST return the return value "Success".

### 3.1.4.2 Required Properties

For every object in the address book, the server MUST minimally maintain the following properties, which are defined in [MS-OXOABK]:

- \texttt{PidTagObjectType} ([MS-OXOABK] section 2.2.3.10)
- \texttt{PidTagInitialDetailsPane} ([MS-OXOABK] section 2.2.3.33)
- \texttt{PidTagAddressBookDisplayNamePrintable} ([MS-OXOABK] section 2.2.3.7)
- \texttt{PidTagAddressBookContainerId} ([MS-OXOABK] section 2.2.2.3)
- \texttt{PidTagEntryId} ([MS-OXOABK] section 2.2.3.2)
- \texttt{PidTagInstanceKey} ([MS-OXOABK] section 2.2.3.6)
- \texttt{PidTagSearchKey} ([MS-OXOABK] section 2.2.3.5)
- \texttt{PidTagRecordKey} ([MS-OXOABK] section 2.2.3.4)
- \texttt{PidTagAddressType} ([MS-OXOABK] section 2.2.3.13)
- \texttt{PidTagEmailAddress} ([MS-OXOABK] section 2.2.3.14)
- \texttt{PidTagDisplayType} ([MS-OXOABK] section 2.2.3.11)
- \texttt{PidTagTemplateId} ([MS-OXOABK] section 2.2.3.3)
- \texttt{PidTagTransmittableDisplayName} ([MS-OXOABK] section 2.2.3.8)
- \texttt{PidTagDisplayName} ([MS-OXOABK] section 2.2.3.1)
- \texttt{PidTagMappingSignature} ([MS-OXOABK] section 2.2.3.32)
- \texttt{PidTagAddressBookObjectDistinguishedName} ([MS-OXOABK] section 2.2.3.15)

The server MUST maintain the following properties, which are defined in [MS-OXOABK], for every object that has a \texttt{PidTagObjectType} property with a value of \texttt{DISTLIST}, value, as specified in [MS-OXOABK] section 2.2.3.10:

- \texttt{PidTagContainerContents} ([MS-OXOABK] section 2.2.6.3)
- \texttt{PidTagContainerFlags} ([MS-OXOABK] section 2.2.2.1)

If the server does not conform to the preceding rules, client behavior is undefined.

### 3.1.4.3 String Handling

A server holds values of properties for objects. Some of these values are strings. The Exchange Server NSPI Protocol allows string values to be represented as 8-bit character strings or Unicode strings. All
string valued properties held by a server are categorized as either natively of property type
PtypString or natively of property type PtypString8. Those properties natively of property type
PtypString8 are further categorized as either case-sensitive or case-insensitive.

3.1.4.3.1 Required Native Categorizations

Unless otherwise specified in this document, the Exchange Server NSPI Protocol does not constrain the
categorization of properties, and clients and servers MUST NOT require specific categorizations.
However, because the protocol intends for clients to be able to persist sorted string values across
multiple NSPI connections to a server, a server MUST NOT modify its native categorization for string
properties after the categorization has been determined, as doing so would lead to inconsistent
behavior of NSPI methods across multiple NSPI sessions.

The following table lists those properties categorization for which is specified by the Exchange Server
NSPI Protocol, and the categorization of those properties.

<table>
<thead>
<tr>
<th>Property name</th>
<th>String categorization</th>
</tr>
</thead>
<tbody>
<tr>
<td>PidTagDisplayName</td>
<td>PtypString</td>
</tr>
<tr>
<td>PidTagAddressBookPhoneticDisplayName</td>
<td>PtypString</td>
</tr>
<tr>
<td>PidTagAddressBookDisplayNamePrintable</td>
<td>PtypString8, case sensitive</td>
</tr>
</tbody>
</table>

3.1.4.3.2 Required Code Page Support

While processing an NSPI method, a server associates a code page with all strings expressed as
parameters in the method. The server MUST at a minimum be able to convert string representations
between the Unicode code page CP_WINUNICODE and the TELETEX code page CP_TELETEX. Clients specify a code page for 8-bit strings in input parameters to server methods. This protocol does
not specify conversion rules. However, because the protocol allows for clients to be able to reliably
access data that has been so converted, after a server uses an algorithm, it MUST NOT modify its
algorithm for converting between string representations in different code pages. Doing so would lead
to inconsistent behavior of NSPI methods across multiple NSPI sessions.

3.1.4.3.3 Conversion Rules for String Values Specified by the Server to the Client

When returning string values as output parameters for methods where the method allows for both
Unicode and 8-bit character representations, the server MUST adhere to the following conversion
rules.

If the native type of a property is PtypString and the client has requested that property with the type
PtypString8, the server MUST convert the Unicode representation to an 8-bit character
representation in the code page specified by the CodePage field of the pStat parameter, or the
dwCodePage parameter prior to returning the value.

If the native type of a property is PtypString and the client has requested that property with the type
PtypString, the server MUST return the Unicode representation unmodified.

If the native type of a property is PtypString8 and the client has requested that property with the type
PtypString, the server MUST convert the 8-bit character representation to a Unicode
representation prior to returning the value. The 8-bit character representation is considered to be in
the code page CP_TELETEX.

If the native type of a property is PtypString8 and the client has requested that property with the type
PtypString8, the server MUST return the 8-bit character representation unmodified.
Servers MAY undertake other conversions and substitutions for specific properties.

The following table lists NSPI methods that are capable of returning string values in both Unicode and 8-bit character representations, and the methods for which the conversion rules are applicable.

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NspiGetTemplateInfo</td>
<td>String values can be returned in the output parameter ppData.</td>
</tr>
<tr>
<td>NspiGetSpecialTable</td>
<td>String values can be returned in the output parameter ppRows.</td>
</tr>
<tr>
<td>NspiGetProps</td>
<td>String values can be returned in the output parameter ppRows.</td>
</tr>
<tr>
<td>NspiQueryRows</td>
<td>String values can be returned in the output parameter ppRows.</td>
</tr>
<tr>
<td>NspiGetMatches</td>
<td>String values can be returned in the output parameter ppRows.</td>
</tr>
<tr>
<td>NspiSeekEntries</td>
<td>String values can be returned in the output parameter ppRows.</td>
</tr>
<tr>
<td>NspiResolveNames</td>
<td>String values can be returned in the output parameter ppRows.</td>
</tr>
<tr>
<td>NspiResolveNamesW</td>
<td>String values can be returned in the output parameter ppRows.</td>
</tr>
</tbody>
</table>

### 3.1.4.3.4 Conversion Rules for String Values Specified by the Client to the Server

When accepting strings as input parameters for methods where the method allows for both Unicode and 8-bit character representations, the server MUST follow these conversion rules:

- If the native type of a property is PtypString8 and the client has specified a property value with the type PtypString, the server MUST convert the Unicode representation to an 8-bit character representation in the code page specified by the CodePage field of the pStat parameter prior to processing the method.

- If the native type of a property is PtypString8 and the client has specified a property value with the type PtypString8, the server MUST leave the 8-bit character representation unmodified while processing the method.

- If the native type of a property is PtypString and the client has specified a property value with the type PtypString8, the server MUST convert the 8-bit character representation to a Unicode representation prior to processing the method. The 8-bit character representation is considered to be in the code page specified by the CodePage field of the pStat parameter.

- If the native type of a property is PtypString and the client has specified a property value with the type PtypString, the server MUST leave the Unicode representation unmodified while processing the method.

The following table lists NSPI methods that are capable of specifying input parameters that contain string values in both Unicode and 8-bit character representations, and methods for which these conversion rules are applicable.

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NspiModProps</td>
<td>String values can be specified in the input parameter pRow.</td>
</tr>
<tr>
<td>NspiSeekEntries</td>
<td>String values can be specified in the input parameter pTarget.</td>
</tr>
<tr>
<td>NspiGetMatches</td>
<td>String values can be specified in the input parameter Filter.</td>
</tr>
<tr>
<td>NspiResolveNames</td>
<td>String values can be specified in the input parameter paStr.</td>
</tr>
</tbody>
</table>
### 3.1.4.3.5 String Comparison

Servers MUST implement comparison between string values. This comparison yields the normal semantics of **less than**, **equal to**, and **greater than**.

#### 3.1.4.3.5.1 Unicode String Comparison

Servers MUST compare **Unicode** representations of strings as specified in [MS-UCODEREF]. All methods in which a server is required to perform such Unicode string comparison include **LCID** as part of the input parameters. The server SHOULD compare the strings using the closest supported LCID. The Exchange Server NSPI Protocol does not constrain how a server chooses this closest supported LCID. However, because the protocol intends for clients to be able to persist sorted string values across multiple NSPI connections to a server, a server SHOULD NOT modify its algorithm for choosing the closest LCID after an algorithm has been implemented because doing so would lead to inconsistent behavior of NSPI methods across multiple NSPI sessions. The server MUST minimally support the **LCID NSPI_DEFAULT_LOCALE** flag, as specified in section 2.2.1.4. When making comparisons of Unicode string values, if the server uses LCID **NSPI_DEFAULT_LOCALE**, the server MUST also use the **NSPI_DEFAULT_LOCALE_COMPARE_FLAGS** flag for the comparison. Otherwise, the server MUST use the **NSPI_NON_DEFAULT_LOCALE_COMPARE_FLAGS** flag.

#### 3.1.4.3.5.2 8-Bit String Comparison

When making comparisons of 8-bit character string values, the server MUST compare according to the following series of steps:

- If the strings are categorized as case-sensitive, the server MUST implement a case-sensitive buffer comparison. If the strings are case-insensitive, the server MUST implement a case-insensitive buffer comparison. The Exchange Server NSPI Protocol does not constrain how a server implements these comparison functions. However, because the protocol intends for clients to be able to persist sorted string values across multiple NSPI connections to a server, a server MUST NOT modify its algorithm for choosing the closest LCID after an algorithm has been implemented because doing so would lead to inconsistent behavior of NSPI methods across multiple NSPI sessions.

- If the buffer representing one of the string values is shorter than the buffer representing the other string value, then the server considers the string value represented by the shorter buffer to be less than the string represented by the longer buffer. No further comparison steps are taken.

1. If the buffers representing the two string values have equal lengths, the comparison function implemented by the server MUST determine that one buffer is less than the other, or that the buffers are equal.

   If the comparison function determines that one of the buffers is less than the other, then the server considers the string value represented by the lesser buffer to be less than the string value represented by the greater buffer. No further comparison steps are taken.

   If the comparison function determines that the two buffers are equal, the server considers the two string values to be equal.

### 3.1.4.3.6 String Sorting

Every server MUST support sorting on **Unicode** string representations for the **PidTagDisplayName** property. If the server supports the **SortTypePhoneticDisplayName** sort order, it MUST also...
support sorting on Unicode string representation for the \texttt{PidTagAddressBookPhoneticDisplayName} property. The server MUST minimally support the \texttt{LCID NSPI_DEFAULT_LOCALE} flag. This sorting conforms to that specified in [MS-UCODEREF].

### 3.1.4.4 Tables

In order to achieve the primary goal of the Exchange Server NSPI Protocol (browsing \texttt{address book containers}), the protocol defines a data model based on tables. Two types of tables are used in the data model for the Exchange Server NSPI Protocol.

#### 3.1.4.4.1 Status-Based Tables

The first type of table specified by the Exchange Server NSPI Protocol is the Status-Based Table. This table directly represents an \texttt{address book container}. A Status-Based Table is specified in the protocol by the use of a \texttt{STAT} data structure. The data structure identifies an address book container, the order of objects in the address book container as exposed by the table, and positioning in the address book container.

The server is not required to maintain any state for a Status-Based Table; the state of the table is entirely specified by the fields of the \texttt{STAT} data structure, which is passed back and forth between the client and the server. Therefore, a single client can have multiple instances of an "open" address book container, each specified by a separate \texttt{STAT} structure.

#### 3.1.4.4.2 Explicit Tables

The second type of table specified by the Exchange Server NSPI Protocol is the Explicit Table. This table is implemented as a list of \texttt{Minimal Entry IDs}. The list is instantiated in the protocol either as an array of \texttt{DWORDs} or as a \texttt{PropertyTagArray_r} structure. This kind of table is used to implement Restriction-Based Explicit Tables and Property Value-Based Explicit Tables.

##### 3.1.4.4.2.1 Restriction-Based Explicit Tables

When a restriction on a table is specified to the server via the \texttt{NspiGetMatches} method, the server locates all the objects that meet the restriction criteria, and the list of the \texttt{Minimal Entry IDs} of those objects is constructed. This list is passed back to the client. Therefore, these Explicit Tables are "snapshots" of the base table. That is, if an object is placed in an Explicit Table, even if the object is deleted from the server, the Minimal Entry ID that specifies that object will still be in the Explicit Table.

##### 3.1.4.4.2.2 Property Value-Based Explicit Tables

When a specific object in the \texttt{address book} and a property on that object is specified to the server via the \texttt{NspiGetMatches} method, the server reads the values of that property and constructs a list of \texttt{Minimal Entry IDs} based on a mapping between the values and other objects in the address book. This is not possible on all properties, only on those properties for which the server can establish a reference between the value of the property and some object in the address book. The Exchange Server NSPI Protocol does not constrain how a server establishes this reference. Clients can identify the properties that the server can map by trying to obtain such a table. The server MUST return an error when it cannot make such a mapping, as specified in section 3.1.4.1.10.

#### 3.1.4.4.3 Specific Instantiations of Special Tables

The Exchange Server NSPI Protocol requires servers to maintain two special tables, in addition to any tables they maintain for normal browsing. The two required special tables are specified in the following two sections.

##### 3.1.4.4.3.1 Address Book Hierarchy Table

[MS-OXNSPI] - v20220215
Exchange Server Name Service Provider Interface (NSPI) Protocol
Copyright © 2022 Microsoft Corporation
Release: February 15, 2022
Each server MUST maintain an address book hierarchy table, as specified in [MS-OXOABK] section 3.1.4.1.

3.1.4.4.3.2 Address Creation Table

Each server MUST maintain an address creation table to clients, as specified in [MS-OXOABKT].

3.1.4.5 Positioning in a Table

In order to achieve the primary goal of the Exchange Server NSPI Protocol (browsing address lists), in addition to the concept of tables, a server MUST support the concept of position in Status-Based and Explicit Tables. Each such table has a Current Position, which specifies a specific row in the table. Methods such as NspiQueryRows return values based on the Current Position in the table, and methods such as NspiUpdateStat and NspiQueryRows modify the Current Position. Positioning in an Explicit Table is defined specifically in the semantics of the NSPI methods that operate on them.

When specifying position in a STAT structure–based table, the client sets the CurrentRec, Delta, ContainerID, SortType, and SortLocale fields of the STAT structure to specify to the server the Current Position in the table at the beginning of an NSPI method. The server sets the CurrentRec, NumPos, and TotalRecs fields to specify to the client the Current Position in the table at the end of an NSPI method. There are two ways for the client to specify position in a STAT structure–based table in the Exchange Server NSPI Protocol: Absolute Positioning and Fractional Positioning.

3.1.4.5.1 Absolute Positioning

The first form of specifying position in a STAT structure–based table is called Absolute Positioning. The client specifies this type of positioning by setting any value in the field CurrentRec field other than MID_CURRENT. The server uses the following steps to identify the Current Position specified by the client:

1. First, the server MUST determine the LCID that it supports that is closest to the LCID specified by SortLocale. The server MAY choose this closest LCID in any way.

2. The server sorts the objects in the address book container specified by ContainerID by the sort type specified in the SortType field and the LCID specified in step 1 of section 3.1.4.5.2.

3. The server identifies the number of objects in the sorted table. The server reports this in the TotalRecs field of the STAT structure.

4. The server locates the object specified by the CurrentRec field. If the server cannot locate the object, the Current Position in the table is undefined. The server MUST support the special Minimal Entry ID MID_BEGINNING_OF_TABLE and MID_END_OF_TABLE, as specified in section 2.2.1.8.

5. The server verifies that the object located in step 4 is in the container specified by the ContainerID field. If the server cannot verify this, the Current Position in the table is undefined.

6. The server moves the Current Position by the number of rows specified by the absolute value of the Delta field of the STAT structure. If the value of Delta is negative, the Current Position is moved toward the beginning of the table. If the value of Delta is positive, the Current Position is moved toward the end of the table. A Delta with the value 0 results in no change to the Current Position.

7. If applying the Delta as described in step 6 would move the Current Position to be before the first row of the table, the server sets the Current Position to the first row of the table and sets the CurrentRec to the Minimal Entry ID of the object that occupies the first row of the table.
8. If applying the Delta as described in step 6 would move the Current Position to be after the end of the table, the server sets the Current Position to a location one row past the last valid row of the table and sets the CurrentRec to the value MID_END_OF_TABLE.

9. The server sets the field CurrentRec to the Minimal Entry ID of the object occupying the row specified by the Current Position.

The server identifies the numeric row of the Current Position in the sorted table. This numeric row is 0-based. That is, the first valid row in the table has the numeric position 0. This is the Numeric Position of the Current Position of the table. The server reports this in the NumPos field of the STAT structure. The server MAY report an approximate value for the Numeric Position. Although the protocol places no boundary or requirements on the accuracy of the approximate value the server returns, it is recommended that implementations maximize the accuracy of the approximation to improve usability of the Exchange NSPI server for clients.

3.1.4.5.2 Fractional Positioning

The second form of specifying position in a STAT structure–based table is called Fractional Positioning. The client specifies this type of positioning by setting the field CurrentRec to the value MID_CURRENT. Fractional positioning is defined as only an approximation in the Exchange Server NSPI Protocol. The server MAY be inaccurate both in locating a row based on fractional positioning and in reporting the resultant actual fractional position. The server uses the following steps to identify the Current Position specified by the client:

1. First, the server identifies the LCID it supports that is closest to the LCID specified by the SortLocale field. The server MAY choose this closest LCID in any way.

2. The server sorts the objects in the address book container specified by the ContainerID field by the sort type specified in the SortType field and the LCID specified in step 1 of section 3.1.4.3.

3. The server identifies the number of objects in the sorted table. The server reports this in the TotalRecs field of the STAT structure.

4. The server calculates the Intended Numeric Position in the table as the TotalRecs reported by the server multiplied by the NumPos field of the STAT structure divided by the value of TotalRecs as specified by the client. The value is truncated to its integral part.

5. If the Intended Numeric Position thus calculated is greater than TotalRecs, the intended Intended Numeric Position is TotalRecs (that is, the last row in the table).

After the server has identified the Intended Numeric Position, the server sets Numeric Position to an approximation of that value. Although the protocol places no boundary or requirements on the accuracy of the approximation the server uses to set the Numeric Position, it is recommended that implementations maximize accuracy of the approximation to improve usability of the server for clients.


7. The server moves the Current Position by the number of rows specified by the absolute value of the Delta field of the STAT structure. If the value of Delta is negative, the Current Position is moved toward the beginning of the table. If the value of Delta is positive, the Current Position is moved toward the end of the table. A Delta field with the value 0 results in no change to the Current Position.

8. If applying the Delta as described in step 8 would move the Current Position to be before the beginning of the table, the server sets the Current Position to the beginning of the table and sets the CurrentRec field to the Minimal Entry ID of the object occupying the first row of the table.

9. If applying Delta as described in step 8 would move the Current Position to be after the end of the table, the server sets the Current Position to a location one row past the last valid row of the table and sets the CurrentRec to the value MID_END_OF_TABLE.
10. The server sets the field **CurrentRec** to the Minimal Entry ID of the object occupying the row specified by the Current Position.

11. The server identifies the numeric row of the Current Position in the sorted table. This numeric row is 0-based. That is, the first valid row in the table has the numeric position 0. This is the **Numeric Position** of the Current Position of the table. The server reports this in the **NumPos** field of the **STAT** structure.

### 3.1.4.6 Object Identity

Objects maintained by the server need to be identified in the Exchange Server NSPI Protocol. The Exchange Server NSPI Protocol makes use of the following three kinds of identifiers, differentiated primarily by their intended lifespan:

- **Permanent Identifier**: Specifies a specific object across all NSPI sessions. The display type of the object is included in the Permanent Identifier.

- **Ephemeral Identifier**: Specifies a specific object in a single NSPI session. The display type of the object is included in the Ephemeral Identifier. A server MUST NOT change an object's Ephemeral Identifier during the lifetime of an NSPI session. If a server uses the same NSPI session **GUID** (that is, the GUID returned by the server in the **pServerGuid** output parameter of the **NspiBind** method) for multiple NSPI sessions, the server MUST use the same Ephemeral Identifier for the same specific object in both sessions.

- **Minimal Identifier**: Specifies a specific object in a single NSPI session. A server MUST NOT change an object's **Minimal Entry ID** during the lifetime of an NSPI session. If a server uses the same NSPI session GUID (that is, the GUID returned by the server in the **pServerGuid** output parameter of the **NspiBind** method) for multiple NSPI sessions, the server MUST use the same Minimal Identifier for the same specific object in all sessions.

### 3.1.4.7 Ambiguous Name Resolution

**Ambiguous name resolution (ANR)** is a process by which a server maps a string to a specific object in a specific address book container. The string is provided by the client and is interpreted by the server as specified in section 3.1.4.3.

The specific algorithm used to map the string to an object is not prescribed by this protocol and is left to each server instance to define as local policy. The intended usage is an end user of a computer program entering free-form text and finding a unique object in an address book most closely matching the user's requirements. The specific result of an ANR process is a **Minimal Entry ID**. There are three possible outcomes to the ANR process:

1. If the server is unable to map the string to any objects in the address book, the result of the ANR process is the Minimal Entry ID with the value **MID_UNRESOLVED**.

2. If the server is able to map the string to more than one object in the address book, the result of the ANR process is the Minimal Entry ID with the value **MID_AMBIGUOUS**.

3. If the server is able to map the string to exactly one object in the address book, the result of the ANR process is the Minimal Entry ID with the value **MID_RESOLVED**.

The server MUST map the NULL string to the Minimal Entry ID **MID_UNRESOLVED**.

The server MUST map a zero-length string to the Minimal Entry ID **MID_UNRESOLVED**.
3.2 Client Details

3.2.1 Abstract Data Model
None.

3.2.2 Timers
None.

3.2.3 Initialization
None.

3.2.4 Message Processing Events and Sequencing Rules

In order to obtain any context handle to the server, the \texttt{NspiBind} method MUST be called initially. With the \texttt{contextHandle} parameter returned from this method, it is possible to call any associated methods on the handle, as described in section 4.

This protocol MUST indicate to the RPC runtime that it is to perform a strict NDR data consistency check at target level 6.0, as specified in \cite{MS-RPCE}.

This protocol MUST indicate to the RPC runtime via the \texttt{strict_context_handle} attribute that it is to reject use of context handles created by a method of a different RPC interface than this one, as specified in \cite{MS-RPCE}.

This protocol MUST indicate to the RPC runtime via the \texttt{type_strict_context_handle} attribute that it is to reject use of context handles created by a method that creates a different type of context handle, as specified in \cite{MS-RPCE}.

3.2.5 Timer Events
For details about any transport-level timers, see \cite{MS-RPCE}.

3.2.6 Other Local Events
None.
4 Protocol Examples

This section shows the call sequence for obtaining the address book hierarchy table at the NSPI layer. It further shows how a messaging client can use this table to retrieve properties of the Address Book objects by using the NspiQueryRows method.

It is assumed that the messaging client has established an RPC connection to the server.

**Note** Only parts of the details of client request parameters and server response parameters are documented, to show only the relevant information.

![Diagram of NSPI session message sequence example]

**Figure 3: NSPI session message sequence example**

The client initiates a session to the server by calling the NspiBind method. Messaging clients send the following values to the server.

**Note** Only relevant information, and not all parameters, are shown.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>dwFlags</td>
<td>0x00000000</td>
<td>unsigned long</td>
</tr>
<tr>
<td>pStat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hIndex</td>
<td>0x00000000</td>
<td>unsigned long</td>
</tr>
<tr>
<td>ContainerID</td>
<td>0x00000000</td>
<td>unsigned long</td>
</tr>
</tbody>
</table>
CurrentRec 0x00000000 unsigned long
Delta 0x00000000 long
NumPos 0x00000000 unsigned long
TotalRecs 0x00000000 unsigned long
CodePage 0x0000004e4 unsigned long
TemplateLocale 0x000000409 unsigned long
SortLocale 0x000000409 unsigned long

pServerGuid
<pointer to an array of 16 unsigned char to be returned by the server>

The server responds to the **NspiBind** method call with return code "Success" and a valid server GUID. Typical parameters are as follows.

```c
pServerGuid
[0x0]0xab 0xbc 0x8b 0x86 0x79 0x33 0xc4 0x48 0xa1 0xef
[0xa]0x1b 0x53 0xe6 0x3b 0xdc 0x46
```

contextHandle
<a token>

The client requests the address book hierarchy table from the server by calling the **NspiGetSpecialTable** method with `dwFlags` parameter typically set to the **NspiUnicodeStrings** bit flag. More importantly, the client does not set the **NspiAddressCreationTemplates** flag. Typical parameters are as follows.

```c
dwFlags 0x00000004 unsigned long
pStat
hIndex 0x00000000 unsigned long
ContainerID 0x00000000 unsigned long
CurrentRec 0x00000000 unsigned long
Delta 0x00000000 long
NumPos 0x00000000 unsigned long
TotalRecs 0x00000000 unsigned long
CodePage 0x0000004e4 unsigned long
TemplateLocale 0x000000409 unsigned long
SortLocale 0x000000409 unsigned long
ppRows <memory location that holds _PropertyRowSet_r** returned by the server>
```

The server responds to the **NspiGetSpecialTable** method call with return code "Success", and the rows of the address book hierarchy table typically have the following columns set, as described in [MS-OXOABK]:

- **PidTagEntryId**
- **PidTagContainerFlags**
- **PidTagDepth**
- **PidTagAddressBookContainerId**
- **PidTagDisplayName**
- **PidTagAddressBookIsMaster**.

In this example, the server did not return the optional **PidTagAddressBookParentEntryId** property. Typical parameters are as follows.
**Note** Only relevant information, and not all return parameters, are shown.

```
ppRows_PropertyRowSet_r **
{
  cRows=0x00000007
  aRow=<a pointer to an array of rows>
}
```

In this example, the server has returned a total of seven rows, denoted as [0x0]...[0x6], and each row typically looks as follows.

```
aRow[0x0] ... [0x6]_PropertyRow_r *
{
  Reserved=0x00000000
  cValues=0x00000006
  lpProps=<a pointer to an array of columns>
}
```

In this example, the server has returned a column set of six properties, and each column looks as follows.

```
[0x0]_PropertyValue_r
{
  ulPropTag=PidTagEntryId
  dwAlignPad=0x00000000
  Value={...}
}

[0x1]_PropertyValue_r
{
  ulPropTag=PidTagContainerFlags
  dwAlignPad=0x00000000
  Value={...}
}

[0x2]_PropertyValue_r
{
  ulPropTag=PidTagDepth
  dwAlignPad=0x00000000
  Value={...}
}

[0x3]_PropertyValue_r
{
  ulPropTag=PidTagAddressBookContainerId
  dwAlignPad=0x00000000
  Value={...}
}

[0x4]_PropertyValue_r
{
  ulPropTag=PidTagDisplayName
  dwAlignPad=0x00000000
  Value={...}
}

[0x5]_PropertyValue_r
{
  ulPropTag=PidTagAddressBookIsMaster
  dwAlignPad=0x00000000
  Value={...}
}
```

**Note** The client can invoke additional NSPI calls to access other information from the server before calling the `NspiUnbind` method.
Messaging clients call the **NspiQueryRows** method to retrieve various properties of Address Book objects. The following example shows the client requesting the server a total of two rows that contain the following properties:

- PidTagEntryId
- PidTagDisplayName
- PidTagSmtpAddress
- PidTagTitle.

Also, the client is requesting the server to use the `pStat` structure for table information by setting `lpETable` NULL and setting relevant values in the `pStat` structure. Typical parameters are as follows.

**Note** Only relevant information, and not all return parameters, are shown.

```c
pStat
  hIndex  0x00000000  unsigned long
  ContainerID  0x00000000  unsigned long
  CurrentRec  0x00000000  unsigned long
  Delta  0x00000000  long
  NumPos  0x00000000  unsigned long
  TotalRecs  0xffffffff  unsigned long
  CodePage  0x000004e4  unsigned long
  TemplateLocale  0x00000409  unsigned long
  SortLocale  0x00000409  unsigned long
  dwETableCount0  0x00000000  unsigned long
  lpETable  0x00000000  unsigned long *
  Count  0x00000002  unsigned long
  Flags  0x00000000  unsigned long
pPropTags_PropertyTagArray_r *
  cValues=0x00000004
  aulPropTag=<a pointer to an array of properties>
}
aulPropTag=an array of 4 PropTags>
  [0x0]PidTagEntryId  unsigned long
  [0x1]PidTagDisplayName  unsigned long
  [0x2]PidTagSmtpAddress  unsigned long
  [0x3]PidTagTitle  unsigned long
```

The server responds to the **NspiQueryRows** method call with return code "Success" and a row set. Typical return parameters are as follows.

**Note** Only relevant information, and not all parameters, are shown.

```c
dwFlags  0x00000000  unsigned long
pStat
  hIndex  0x00000000  unsigned long
  ContainerID  0x00000000  unsigned long
  CurrentRec  0x000001928  unsigned long
  Delta  0x00000000  long
  NumPos  0x00000002  unsigned long
  TotalRecs  0x00000016  unsigned long
  CodePage  0x000004e4  unsigned long
  TemplateLocale  0x00000409  unsigned long
  SortLocale  0x00000409  unsigned long
  dwETableCount  0x00000000  unsigned long
  lpETable  0x00000000  unsigned long *
  Count  0x00000002  unsigned long
pPropTags_PropertyRowSet_r * *
```

---

[MS-OXNSPI] - v20220215
Exchange Server Name Service Provider Interface (NSPI) Protocol
Copyright © 2022 Microsoft Corporation
Release: February 15, 2022
cRows=0x00000002
aRow=<a pointer to an array of rows>
}

In this example, the server has returned a total of 0x2 rows denoted as [0x0]...[0x1] equal to the number of rows requested by the client. Each row typically looks as follows.

aRow[0x0] ... [0x1]_PropertyRow_r *
| Reserved=0x00000000
| cValues=0x00000004
| lpProps=<a pointer to an array of columns>
}

In this example, the server has returned a column set of four properties equal to the number of properties requested by the client. Each column looks as follows.

[0x0]_PropertyValue_r
|
| ulPropTag= PidTagEntryId
| dwAlignPad=0x00000000
| Value={...}
}
[0x1]_PropertyValue_r
|
| ulPropTag= PidTagDisplayName
| dwAlignPad=0x00000000
| Value={...}
}
[0x2]_PropertyValue_r
|
| ulPropTag= PidTagSmtpAddress
| dwAlignPad=0x00000000
| Value={...}
}
[0x3]_PropertyValue_r
|
| ulPropTag= PidTagTitle
| dwAlignPad=0x00000000
| Value={...}
}

The client terminates the connection by calling the **NspiUnbind** method with a token that the server returned in response to the **NspiBind** method call.

contextHandleNSPI_HANDLE *
<a token>
dwFlags 0x00000000 unsigned long

The server responds with return code 0x00000001 and destroys the token that the client passed.
5 Security

5.1 Security Considerations for Implementers

The Exchange Server NSPI Protocol is not suitable for general administration of the data held by a server. It is suitable for client read access to data with limited modification of existing objects, not including address book container objects. Administration tasks the Exchange Server NSPI Protocol does not support include (but are not limited to) adding new objects to an address book, removing existing objects, and moving existing objects from one address book to another.

Beyond the basic support for address book browsing, a server can apply local security policies. When applying these security policies, the server can limit a client's access to data, either reading access and/or modification access. The simplest form of local security policy is the empty set; all data held by the server is accessible to all clients of the Exchange Server NSPI Protocol for both reading and modifying, regardless of the identity of the client. Local security policy is, with one exception, an implementation-specific detail and is not constrained by the Exchange Server NSPI Protocol. If local security policy allows a client read access to an object, the server is required to allow the client read access to the properties of the object specifying the objects identity. The following properties specify an object's identity:

- PidTagTransmittableDisplayName
- PidTagDisplayName
- PidTagAddressBookDisplayNamePrintable
- PidTagEmailAddress
- PidTagAddressType
- PidTagInitialDetailsPane
- PidTagInstanceKey
- PidTagAddressBookContainerId
- PidTagObjectType
- PidTagContainerContents
- PidTagContainerFlags
- PidTagDisplayType
- PidTagTemplateId
- PidTagEntryId
- PidTagMappingSignature
- PidTagRecordKey
- PidTagSearchKey

The protocol does not provide support for administration of local security policy or for client discovery of a server's security policy.

The protocol carries identity information from the client to the server in the form of an authenticated remote procedure call (RPC) connection. The client MUST create a secure RPC session such that
the server can identify and determine the authorization for the client. For details about secure RPC, see [MS-RPCE]. This requirement exists so that the server can implement its security model.

The server can use this information to apply local security policy. How the server uses this information is an implementation-specific detail and is not constrained by the protocol.

### 5.2 Index of Security Parameters

<table>
<thead>
<tr>
<th>Security parameter</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPC connection security</td>
<td>2.1</td>
</tr>
</tbody>
</table>
6 Appendix A: Full IDL

For ease of implementation, the following full IDL is provided, where "ms-dtyp.idl" refers to the IDL found in [MS-DTYP] Appendix A. The syntax uses the IDL syntax extensions defined in [MS-RPCE]. For example, as noted in [MS-RPCE], a pointer_default declaration is not required and pointer_default(unique) is assumed.

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

typedef long NTSTATUS;
typedef unsigned long DWORD;

[uuid (F5CC5A18-4264-101A-8C59-08002B2F8426),
 version(56.0)]
interface nspi {
    typedef struct {
        BYTE ab[16];
    } FlatUID_r;

typedef struct PropertyTagArray_r {
    DWORD cValues;
    [range(0, 100000)]
    [size_is(cValues + 1),
    length_is(cValues)] DWORD aulPropTag[];
} PropertyTagArray_r;

typedef struct Binary_r {
    [range(0, 2097152)] DWORD cb;
    [size_is(cb)] BYTE * lpb;
} Binary_r;

typedef struct ShortArray_r {
    [range(0, 100000)] DWORD cValues;
    [size_is(cValues)] short int * lpi;
} ShortArray_r;

typedef struct _LongArray_r {
    [range(0, 100000)] DWORD cValues;
    [size_is(cValues)] long * lpl;
} LongArray_r;

typedef struct _StringArray_r {
    [range(0, 100000)] DWORD cValues;
    [size_is(cValues)] [string] char ** lppszA;
} StringArray_r;

typedef struct _BinaryArray_r {
    [range(0, 100000)] DWORD cValues;
    [size_is(cValues)] Binary_r * lpbin;
} BinaryArray_r;

typedef struct _FlatUIDArray_r {
    [range(0, 100000)] DWORD cValues;
    [size_is(cValues)] FlatUID_r** lpguid;
} FlatUIDArray_r;

typedef struct _WStringArray_r {
    [range(0, 100000)] DWORD cValues;
    [size_is(cValues)] [string] wchar_t ** lppszW;
} WStringArray_r;

typedef struct _DateTimeArray_r {
    [range(0, 100000)] DWORD cValues;

```
typedef struct _PropertyRow_r {
    DWORD Reserved;
    [range(0, 100000)] DWORD cValues;
    [size_is(cValues)] PropertyValue_r * lpProps;
} PropertyRow_r;

typedef struct _PropertyRowSet_r {
    [range(0, 100000)] DWORD cRows;
    [size_is(cRows)] PropertyRow_r aRow[];
} PropertyRowSet_r;

typedef struct _Restriction_r Restriction_r;

typedef struct _AndOrRestriction_r {
    [range(0, 100000)] DWORD cRes;
    [size_is(cRes)] Restriction_r * lpRes;
} AndRestriction_r, OrRestriction_r;

typedef struct _NotRestriction_r {
    Restriction_r * lpRes;
} NotRestriction_r;

typedef struct _ContentRestriction_r {
    DWORD ulFuzzyLevel;
    DWORD ulPropTag;
    PropertyValue_r * lpProp;
} ContentRestriction_r;

typedef struct _BitMaskRestriction_r {
    DWORD relBMR;
    DWORD ulPropTag;
    DWORD ulMask;
} BitMaskRestriction_r;

typedef struct _PropertyRestriction_r {
    DWORD relop;
    DWORD ulPropTag;
    PropertyValue_r * lpProp;
} PropertyRestriction_r;

typedef struct _ComparePropsRestriction_r {
    DWORD relop;
    DWORD ulPropTag1;
    DWORD ulPropTag2;
} ComparePropsRestriction_r;

typedef struct _SubRestriction_r {
    DWORD ulSubObject;
    Restriction_r * lpRes;
} SubRestriction_r;

typedef struct _SizeRestriction_r {
    DWORD relop;
    DWORD ulPropTag;
    DWORD cb;
} SizeRestriction_r;

typedef struct _ExistRestriction_r {
    DWORD ulReserved1;
    DWORD ulPropTag;
    DWORD ulReserved2;
} ExistRestriction_r;
typedef [switch_type(long)] union _RestrictionUnion_r {
    [case (0x00000000)] AndRestriction_r resAnd;
    [case (0x00000001)] OrRestriction_r resOr;
    [case (0x00000002)] NotRestriction_r resNot;
    [case (0x00000003)] ContentRestriction_r resContent;
    [case (0x00000004)] PropertyRestriction_r resProperty;
    [case (0x00000005)] ComparePropsRestriction_r resCompareProps;
    [case (0x00000006)] BitMaskRestriction_r resBitMask;
    [case (0x00000007)] SizeRestriction_r resSize;
    [case (0x00000008)] ExistRestriction_r resExist;
    [case (0x00000009)] SubRestriction_r resSubRestriction;
} RestrictionUnion_r;

struct _Restriction_r {
    DWORD rt;
    [switch_is((long)rt)] RestrictionUnion_r res;
};

typedef struct PropertyName_r {
    FlatUID_r * lpguid;
    DWORD ulReserved;
    long lID;
} PropertyName_r;

typedef struct StringsArray {
    [range(0, 100000)] DWORD Count;
    [size_is(Count)] [string] char * Strings[];
} StringsArray_r;

typedef struct WStringsArray {
    [range(0, 100000)] DWORD Count;
    [size_is(Count)] [string] wchar_t * Strings[];
} WStringsArray_r;

typedef struct _STAT {
    DWORD SortType;
    DWORD ContainerID;
    DWORD CurrentRec;
    long Delta;
    DWORD NumPos;
    DWORD TotalRecs;
    DWORD CodePage;
    DWORD TemplateLocale;
    DWORD SortLocale;
} STAT;

typedef [switch_type(long)] union _PV_r {
    [case (0x00000002)] short int i;
    [case (0x00000003)] long l;
    [case (0x0000000B)] unsigned short int b;
    [case (0x0000001E)] [string] char * lpszA;
    [case (0x0000001F)] [string] wchar_t * lpszW;
    [case (0x00000048)] FlatUID_r * lpguid;
    [case (0x00000049)] FILETIME ft;
    [case (0x0000004A)] long err;
    [case (0x00001002)] ShortArray_r MVi;
    [case (0x00001003)] LongArray_r MVl;
    [case (0x0000101E)] StringArray_r MVszA;
    [case (0x00001048)] FlatUIDArray_r MVguid;
    [case (0x00001049)] WStringArray_r MVszW;
    [case (0x0000104A)] DateTimeArray_r MVft;
}
typedef [context_handle] void * NSPI_HANDLE;

//opnum 0
long
NspiBind{
[in] handle_t hRpc,
[in] DWORD dwFlags,
[in] STAT * pStat,
[in,out,unique] FlatUID_r * pServerGuid,
[out,ref] NSPI_HANDLE * contextHandle
};

//opnum 1
DWORD
NspiUnbind{
[in,out] NSPI_HANDLE * contextHandle,
[in] DWORD Reserved
};

//opnum 2
long
NspiUpdateStat{
[in] NSPI_HANDLE hRpc,
[in] DWORD Reserved,
[in,out] STAT * pStat,
[in,out,unique] long * plDelta
};

//opnum 3
long
NspiQueryRows{
[in] NSPI_HANDLE hRpc,
[in] DWORD dwFlags,
[in, out] STAT * pStat,
[in, range(0, 100000)] DWORD dwETableCount,
[in, unique, size_is(dwETableCount)] DWORD * lpETable,
[in] DWORD Count,
[in,unique] PropertyTagArray_r * pPropTags,
[out] PropertyRowSet_r ** ppRows
};

//opnum 4
long
NspiSeekEntries{
[in] NSPI_HANDLE hRpc,
[in] DWORD Reserved,
[in,out] STAT * pStat,
[in] PropertyValue_r * pTarget,
[in, unique] PropertyTagArray_r * lpETable,
[in,unique] PropertyTagArray_r * pPropTags,
[out] PropertyRowSet_r ** ppRows
};

//opnum 5
long
NspiGetMatches{
[in] NSPI_HANDLE hRpc,
[in] DWORD Reserved1,
[in,out] STAT * pStat,
[in, unique] PropertyTagArray_r * pReserved,
[in] DWORD Reserved2,
[in, unique] Restriction_r * Filter,

[in, unique] PropertyName_r * lpPropName,
[in] DWORD uiRequested,
[out] PropertyTagArray_r ** ppOutMIds,
[in, unique] PropertyTagArray_r * pPropTags,
[out] PropertyRowSet_r ** ppRows
);

//opnum 6
long NspiResortRestriction(
[in] NSPI_HANDLE hRpc,
[in] DWORD Reserved,
[in,out] STAT * pStat,
[in] PropertyTagArray_r * pInMIds,
[in,out] PropertyTagArray_r ** ppOutMIds
);

//opnum 7
long NspiDNToMId(
[in] NSPI_HANDLE hRpc,
[in] DWORD Reserved,
[in] StringsArray_r * pNames,
[out] PropertyTagArray_r ** ppOutMIds
);

//opnum 8
long NspiGetPropList(
[in] NSPI_HANDLE hRpc,
[in] DWORD dwFlags,
[in] DWORD dwMid,
[in] DWORD CodePage,
[out] PropertyTagArray_r ** ppPropTags
);

//opnum 9
long NspiGetProps(
[in] NSPI_HANDLE hRpc,
[in] DWORD dwFlags,
[in] STAT * pStat,
[in, unique] PropertyTagArray_r * pPropTags,
[out] PropertyRow_r ** ppRows
);

//opnum 10
long NspiCompareMIds(
[in] NSPI_HANDLE hRpc,
[in] DWORD Reserved,
[in] STAT * pStat,
[in] DWORD Mid1,
[in] DWORD Mid2,
[out] long * plResult
);

//opnum 11
long NspiModProps(
[in] NSPI_HANDLE hRpc,
[in] DWORD Reserved,
[in] STAT * pStat,
[in, unique] PropertyTagArray_r * pPropTags,
[in] PropertyRow_r * pRow
);

//opnum 12
long
NspiGetSpecialTable(
[in] NSPI_HANDLE hRpc,
[in] DWORD dwFlags,
[in] STAT * pStat,
[in, out] DWORD * lpVersion,
[out] PropertyRowSet_r ** ppRows
);

//opnum 13
long
NspiGetTemplateInfo(
[in] NSPI_HANDLE hRpc,
[in] DWORD dwFlags,
[in] DWORD ulType,
[in,unique] [string] char * pDN,
[in] DWORD dwCodePage,
[in] DWORD dwLocaleID,
[out] PropertyRow_r ** ppData
);

//opnum 14
long
NspiModLinkAtt(
[in] NSPI_HANDLE hRpc,
[in] DWORD dwFlags,
[in] DWORD ulPropTag,
[in] DWORD dwMId,
[in] BinaryArray_r * lpEntryIds
);

// opnum 15
void Opnum15NotUsedOnWire(void);

//opnum 16
long
NspiQueryColumns(
[in] NSPI_HANDLE hRpc,
[in] DWORD Reserved,
[in] DWORD dwFlags,
[out] PropertyTagArray_r ** ppColumns
);

// opnum 17
void Opnum17NotUsedOnWire(void);

// opnum 18
void Opnum18NotUsedOnWire(void);

//opnum 19
long
NspiResolveNames(
[in] NSPI_HANDLE hRpc,
[in] DWORD Reserved,
[in] STAT * pStat,
[in, unique] PropertyTagArray_r * pPropTags,
[in] StringsArray_r * paStr,
[out] PropertyTagArray_r ** ppMIds,
[out] PropertyRowSet_r ** ppRows
);

//opnum 20
long NspiResolveNamesW(
    [in] NSPI_HANDLE hRpc,
    [in] DWORD Reserved,
    [in] STAT * pStat,
    [in, unique] PropertyTagArray_r * pPropTags,
    [in] WStringsArray_r * pWStr,
    [out] PropertyTagArray_r ** ppMIds,
    [out] PropertyRowSet_r ** ppRows
);
7 Appendix B: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include updates to those products.

 Microsoft Exchange Server 2010
 Microsoft Exchange Server 2013
 Microsoft Exchange Server 2016
 Microsoft Exchange Server 2019
 Microsoft Office Outlook 2003
 Microsoft Office Outlook 2007
 Microsoft Outlook 2010
 Microsoft Outlook 2013
 Microsoft Outlook 2016
 Microsoft Outlook 2019
 Microsoft Outlook 2021

Exceptions, if any, are noted in this section. If an update version, service pack or Knowledge Base (KB) number appears with a product name, the behavior changed in that update. The new behavior also applies to subsequent updates unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

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


<2> Section 2.2.9.3: Exchange 2013, Exchange 2016, and Exchange 2019 do not follow the ABNF format that is specified in [MS-OXOABK] section 2.2.1.1.


<5> Section 3.1.4.1.7: Microsoft Exchange Server 2010 Service Pack 3 (SP3), Exchange 2013, Exchange 2016, and Exchange 2019 return "ErrorsReturned" (0x00040380).

<6> Section 3.1.4.1.11: Exchange 2013, Exchange 2016, and Exchange 2019 returns "Success".

Section 3.1.4.1.15: Exchange 2013, Exchange 2016, and Exchange 2019 return "GeneralFailure" (0x80004005) when modification of either the PidTagAddressBookMember property ([MS-OXOABK] section 2.2.6.1) or the PidTagAddressBookPublicDelegates property ([MS-OXOABK] section 2.2.5.5) is attempted.
8 Change Tracking

This section identifies changes that were made to this document since the last release. Changes are classified as Major, Minor, or None.

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

- A document revision that incorporates changes to interoperability requirements.
- A document revision that captures changes to protocol functionality.

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

The revision class None means that no new technical changes were introduced. Minor editorial and formatting changes may have been made, but the relevant technical content is identical to the last released version.

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

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Revision class</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1.4.1.11</td>
<td>NspiResortRestriction (Opnum 6)</td>
<td>Updated the endnote.</td>
</tr>
</tbody>
</table>
sequencing rules 39
String Handling method 68
Tables method 72
timers 39
Standards assignments 13
String Handling method 68

T

Tables method 72
Timer events
  client 76
Timers
  client 76
  server 39
Tracking changes 93
Transport 14

V

Vendor-extensible fields 13
Versioning 12