[MS-ADSO]:
Active Directory System Overview

This document provides an overview of the Active Directory System Overview Protocol Family. It is intended for use in conjunction with the Microsoft Protocol Technical Documents, publicly available standard specifications, network programming art, and Microsoft Windows distributed systems concepts. It assumes that the reader is either familiar with the aforementioned material or has immediate access to it.

A Protocol Family System Document does not require the use of Microsoft programming tools or programming environments in order to implement the Protocols in the System. Developers who have access to Microsoft programming tools and environments are free to take advantage of them.

Abstract

The Active Directory System provides the foundation for authentication services in a domain environment. Additionally, it is used as a component by other systems such as Group Policy and Print Services. Active Directory is a directory service that provides for the centralized storage of identity and account information, as well as storage for other forms of data such as group policies and printer location information. A directory service contains one or more servers, known as directory servers, in which directory objects can be created, queried for, modified, and deleted. Each directory object is a collection of attributes, each of which contains one or more values. One common directory object is the user object, which represents an account and stores information such as the user name and password.

This document describes the relationships of the system of protocols that make up the client-server behavior of the Active Directory System. It describes the intended client-server functionality of the Active Directory System and how the protocols in this system interact. In the context of the Active Directory System, "clients" can include both Microsoft Windows client and server operating systems, other Microsoft software, and third-party software and operating systems. This document does not describe server-to-server functionality, such as replication protocols. Nor does it describe the impact on clients of being "joined" to a domain or of domain-related interactions between the member computers and the AD DS domain controller, such as Kerberos support or certificate autoenrollment. These topics are discussed in separate documents.

This document does not restate the processing rules and other details that are specific to each protocol in the system. These details are described in the protocol specifications for each of the protocols and data structures that make up the Active Directory System.

Revision Summary

Date	Revision History	Revision Class	Comments
07/02/2009	0.1	Major	First Release.
08/14/2009	1.0	Major	Updated and revised the technical content.
09/25/2009	2.0	Major	Updated and revised the technical content.
11/06/2009	3.0	Major	Updated and revised the technical content.
12/18/2009	3.0.1	Editorial	Revised and edited the technical content.
01/29/2010	4.0	Major	Updated and revised the technical content.
03/12/2010	5.0	Major	Updated and revised the technical content.

Date	Revision History	Revision Class	Comments
04/23/2010	6.0	Major	Updated and revised the technical content.
06/04/2010	6.0.1	Editorial	Revised and edited the technical content.
07/16/2010	7.0	Major	Significantly changed the technical content.
08/27/2010	8.0	Major	Significantly changed the technical content.
10/08/2010	9.0	Major	Significantly changed the technical content.
11/19/2010	10.0	Major	Significantly changed the technical content.
01/07/2011	11.0	Major	Significantly changed the technical content.
02/11/2011	12.0	Major	Significantly changed the technical content.
03/25/2011	13.0	Major	Significantly changed the technical content.
05/06/2011	14.0	Major	Significantly changed the technical content.
06/17/2011	14.1	Minor	Clarified the meaning of the technical content.
09/23/2011	14.1	No change	No changes to the meaning, language, or formatting of the technical content.
12/16/2011	14.1	No change	No changes to the meaning, language, or formatting of the technical content.
03/30/2012	14.1	No change	No changes to the meaning, language, or formatting of the technical content.
07/12/2012	14.1	No change	No changes to the meaning, language, or formatting of the technical content.
10/25/2012	14.2	Minor	Clarified the meaning of the technical content.
01/31/2013	14.2	No change	No changes to the meaning, language, or formatting of the technical content.
08/08/2013	15.0	Major	Significantly changed the technical content.
11/14/2013	15.0	No change	No changes to the meaning, language, or formatting of the technical content.
02/13/2014	15.0	No change	No changes to the meaning, language, or formatting of the technical content.

Contents

1 Introduction

This Protocol Family System Document (PFSD) is primarily intended to cover the Protocol Family as a whole. In conjunction with Member Protocol Technical Documents (TDs), which are intended to cover Member Protocols, it presents the rules for information exchange relevant to those Member Protocols and the Protocol Family that are used to interoperate or communicate with a Windows operating system in its various environments.

Examples for this system PFSD are available at [SysDocCap].

The Active Directory System is a **directory service**. Directory services can be used to provide a central store for identity and **account** information as well as storage for other systems and applications. Use of the Active Directory System is appropriate when there is a need for a directory service. It is also appropriate when building another system that has a dependency on the Active Directory System. An example of such a system is the Windows **domain** security model, which is documented in the Domain Integration System Overview [MS-DISO] Defined Task System Document (DTSD).

This PFSD describes the family of protocols that comprise the Active Directory System. It also documents abstract state that is shared between the system's protocols. This document should be read by anyone interested in implementing the Active Directory System because it provides a high-level introduction to the functionality of the system and serves as documentation for which protocols must be supported by an implementation of the Active Directory System.

This PFSD does not duplicate or replace the content of the TDs that describe the individual protocols in the Active Directory System. An implementer must refer to those TDs for information on each protocol. Additionally, the Active Directory Technical Specification [MS-ADTS] contains vital information about the behavior of the directory service, such as the state model and processing rules, that is essential to the proper functioning of the system.

1.1 Glossary

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

access control entry (ACE)
access control list (ACL)
account
Active Directory
Active Directory Domain Services (AD DS)
Authentication Service (AS)
canonical name
certificate
client
directory
directory object
directory service (DS)
distinguished name (DN)
domain
domain controller (DC)
Domain Name System (DNS)
domain naming context (domain NC)
domain object
endpoint

garbage collection
global catalog (GC)
Interface Definition Language (IDL)
Kerberos
Key Distribution Center (KDC)
KRB_PRIV exchange
Lightweight Directory Access Protocol (LDAP)
mailslot
mutual authentication
NetBIOS
Network Data Representation (NDR)
object class
primary domain controller (PDC)
read-only domain controller (RODC)
relative distinguished name (RDN)
relative identifier (RID)
remote procedure call (RPC)
role
role change
root directory system agent-specific entry (rootDSE)
SASL
schema
secret object
Secure Sockets Layer (SSL)
security descriptor
security identifier (SID)
security principal
server
Server Message Block (SMB)
Service Principal Name (SPN)
service ticket
SOAP
SOAP fault
ticket-granting service (TGS)
ticket-granting ticket (TGT)
tombstone
Transmission Control Protocol (TCP)
Transport Layer Security (TLS)
trusted domain object (TDO)
User Datagram Protocol (UDP)
user principal name (UPN)
Web Services Description Language (WSDL)
XML

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

synthetic attribute

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

Active Directory Lightweight Directory Services (AD LDS)
application NC
attribute
deleted-object
domain functional level

flexible single master operation (FSMO)
forest
FSMO role
replica

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

service record

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

identity attribute
identity attribute type
identity attribute value
identity object

The following terms are defined in [WSA]:

endpoint reference
Web Service endpoint

The following terms are defined in [WSENUM]:

enumeration context

The following terms are specific to this document:

directory tree: A **directory service** is organized into a hierarchical tree structure where each **directory object** has exactly one parent **directory object** (except for one object that serves as the root of the tree) and zero or more child **directory objects**.

extended control: A mechanism for specifying extension information in a **Lightweight Directory Access Protocol (LDAP)** version 3 operation. It is documented in [RFC2251] section 4.1.12, Controls, where it is referred to as simply a "control".

KRB_PRIV: A **Kerberos** message, as specified in [RFC4120] section 3.5, The KRB_PRIV Exchange, that is part of a **KRB_PRIV exchange**.

Replica domain controller / replica directory server: a replica **domain controller** or directory server is a server that contains a replicated copy of the directory and is able to answer client requests over any protocol supported by the directory service.

WS-Addressing: The **WS-Addressing** protocol as defined in [WSA].

WS-AddressingAndIdentity: The **WS-AddressingAndIdentity** protocol as defined in [WSAIdentity].

WS-Policy: WS-Policy as defined in [WSPolicy1.5/FRMWK] and [WSPolicy1.5/Att].

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as described in [RFC2119]. Note that in [RFC2119] terms, most of these specifications should be imperative, to ensure interoperability. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

Any specification that does not explicitly use one of these terms is mandatory, exactly as if it used MUST.

The following protocol abbreviations are used in this document:

ADCAP: The Active Directory Web Services Custom Action Protocol as defined in [MS-ADCAP].

DS RPC: The Directory Replication Service (DRS) Remote Protocol as defined in [MS-DRSR].

DSSP: The Directory Services Setup Remote Protocol as defined in [MS-DSSP].

IMDA: The WS-Transfer Identity Management Operations for Directory Access Protocol Extensions as defined in [MS-WSTIM].

LDAP: **Lightweight Directory Access Protocol**, which can be either version 2 (defined in [RFC1777]) or version 3 (defined in [RFC3377]).

LSAD: The Local Security Authority (Domain Policy) Remote Protocol as defined in [MS-LSAD].

LSAT: The Local Security Authority (Translation Methods) Remote Protocol as defined in [MS-LSAT].

SAMR: The Security Account Manager (SAM) Remote Protocol as defined in [MS-SAMR].

WS-Enumeration: The WS-Enumeration protocol as defined in [WSENUM].

WS-Transfer: The WS-Transfer protocol as defined in [WXFR].

WSDS: The WS-Enumeration Directory Services Protocol Extensions as defined in [MS-WSDS].

WSPELD: WS-Transfer **Lightweight Directory Access Protocol (LDAP)** v3 Control Extension as defined in [MS-WSPELD].

1.2 References

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

A reference marked "(Archived)" means that the reference document was either retired and is no longer being maintained or was replaced with a new document that provides current implementation details. We archive our documents online [Windows Protocol].

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.

[MC-NMF] Microsoft Corporation, ".NET Message Framing Protocol".

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

[MS-ADA2] Microsoft Corporation, "Active Directory Schema Attributes M".

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

[MS-ADCAP] Microsoft Corporation, "Active Directory Web Services: Custom Action Protocol".

[MS-ADDM] Microsoft Corporation, "Active Directory Web Services: Data Model and Common Elements".

[MS-ADLS] Microsoft Corporation, "Active Directory Lightweight Directory Services Schema".

[MS-ADSC] Microsoft Corporation, "Active Directory Schema Classes".

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

[MS-AUTHSO] Microsoft Corporation, "Windows Authentication Services System Overview". (Archived)

[MS-CAESO] Microsoft Corporation, "Certificate Autoenrollment System Overview". (Archived)

[MS-CASO] Microsoft Corporation, "Certification Authority System Overview". (Archived)

[MS-DISO] Microsoft Corporation, "Domain Interactions System Overview".

[MS-DRSR] Microsoft Corporation, "Directory Replication Service (DRS) Remote Protocol".

[MS-DSSP] Microsoft Corporation, "Directory Services Setup Remote Protocol".

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

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

[MS-GPSO] Microsoft Corporation, "Group Policy System Overview". (Archived)

[MS-LSAD] Microsoft Corporation, "Local Security Authority (Domain Policy) Remote Protocol".

[MS-LSAT] Microsoft Corporation, "Local Security Authority (Translation Methods) Remote Protocol".

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

[MS-NAPSO] Microsoft Corporation, "Network Policy and Access Services System Overview".

[MS-NRPC] Microsoft Corporation, "Netlogon Remote Protocol".

[MS-PSSO] Microsoft Corporation, "Print Services System Overview". (Archived)

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

[MS-SAMR] Microsoft Corporation, "Security Account Manager (SAM) Remote Protocol (Client-to-Server)".

[MS-SCMR] Microsoft Corporation, "Service Control Manager Remote Protocol".

[MS-WSDS] Microsoft Corporation, "WS-Enumeration: Directory Services Protocol Extensions".

[MS-WSPELD] Microsoft Corporation, "WS-Transfer and WS-Enumeration Protocol Extension for Lightweight Directory Access Protocol v3 Controls".

[MS-WSPOL] Microsoft Corporation, "Web Services: Policy Assertions and WSDL Extensions".

[MS-WSTIM] Microsoft Corporation, "WS-Transfer: Identity Management Operations for Directory Access Extensions".

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

[RFC1034] Mockapetris, P., "Domain Names - Concepts and Facilities", STD 13, RFC 1034, November 1987, http://www.ietf.org/rfc/rfc1034.txt

[RFC1777] Yeong, W., Howes, T., and Kille, S., "Lightweight Directory Access Protocol", RFC 1777, March 1995, http://www.ietf.org/rfc/rfc1777.txt

[RFC1798] Young, A., "Connection-less Lightweight X.500 Directory Access Protocol", RFC 1798, June 1995, http://www.ietf.org/rfc/rfc1798.txt

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

[RFC2136] Thomson, S., Rekhter Y. and Bound, J., "Dynamic Updates in the Domain Name System (DNS UPDATE)", RFC 2136, April 1997, http://www.ietf.org/rfc/rfc2136.txt

[RFC2251] Wahl, M., Howes, T., and Kille, S., "Lightweight Directory Access Protocol (v3)", RFC 2251, December 1997, http://www.ietf.org/rfc/rfc2251.txt

[RFC2252] Wahl, M., Coulbeck, A., Howes, T., and Kille, S., "Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions", RFC 2252, December 1997, http://www.ietf.org/rfc/rfc2252.txt

[RFC2782] Gulbrandsen, A., Vixie, P., and Esibov, L., "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000, http://www.ietf.org/rfc/rfc2782.txt

[RFC3244] Swift, M., Trostle, J., and Brezak, J., "Microsoft Windows 2000 Kerberos Change Password and Set Password Protocols", RFC 3244, February 2002, http://www.ietf.org/rfc/rfc3244.txt

[RFC3377] Hodges, J., and Morgan, R., "Lightweight Directory Access Protocol (v3): Technical Specification", RFC 3377, September 2002, http://www.ietf.org/rfc/rfc3377.txt

[RFC3645] Kwan, S., Garg, P., Gilroy, J., et al., "Generic Security Service Algorithm for Secret Key Transaction Authentication for DNS (GSS-TSIG)", RFC 3645, October 2003, http://www.ietf.org/rfc/rfc3645.txt

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

[SOAP1.2-1/2003] Gudgin, M., Hadley, M., Mendelsohn, N., et al., "SOAP Version 1.2 Part 1: Messaging Framework", W3C Recommendation, June 2003, http://www.w3.org/TR/2003/REC-soap12-part1-20030624

[WSPolicy1.5/Att] Vedamuthu, A., Orchard, D., Hirsch, N., et al., "Web Services Policy 1.5 - Attachment", W3C Recommendation, September 2007, http://www.w3.org/Submission/WS-PolicyAttachment/

[WSA] Gudgin, M., Hadley, M., and Rogers, T., "Web Services Addressing 1.0 - Core", W3C Recommendation, May 2006, http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/

[WSAIdentity] Alexander, J., Della-Libera, G., Gudgin, M., et al., "Application Note: Web Services Addressing Endpoint References and Identity", August 2008, http://schemas.xmlsoap.org/ws/2006/02/addressingidentity/WS-AddressingAndIdentity.pdf

[WSASB] Gudgin, M., Hadley, M., and Rogers, T., "Web Services Addressing 1.0 - SOAP Binding", W3C Recommendation, May 2006, http://www.w3.org/TR/2006/REC-ws-addr-soap-20060509/

[WSDL] Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S., "Web Services Description Language (WSDL) 1.1", W3C Note, March 2001, http://www.w3.org/TR/2001/NOTE-wsdl-20010315

[WSENUM] Alexander, J., Box, D., Cabrera, L.F., et al., "Web Services Enumeration (WS-Enumeration)", March 2006, http://www.w3.org/Submission/2006/SUBM-WS-Enumeration-20060315/

[WSPolicy1.5/FRMWK] Vedamuthu, A., Orchard, D., Hirsch, N., et al., "Web Services Policy 1.5 - Framework", W3C Recommendation, September 2007, http://www.w3.org/TR/2007/REC-ws-policy-20070904/

[WXFR] Alexander, J., Box, D., Cabrera, L.F., et al., "Web Services Transfer (WS-Transfer)", September 2006, http://www.w3.org/Submission/2006/SUBM-WS-Transfer-20060927/

1.2.2 Informative References

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

[MS-WSO] Microsoft Corporation, "Windows System Overview". (Archived)

[SysDocCap] Microsoft Corporation, "Microsoft System Document Captures", 2006-2009, http://sysdoccap.codeplex.com/

[RFC792] Postel, J., "Internet Control Message Protocol", RFC 792, September 1981, http://www.ietf.org/rfc/rfc792.txt

[RFC793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981, http://www.ietf.org/rfc/rfc0793.txt

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts -- Communication Layers", STD 3, RFC 1122, October 1989, http://www.ietf.org/rfc/rfc1122.txt

[RFC2255] Howes, T., and Smith, M., "The LDAP URL Format", RFC 2255, December 1997, http://www.ietf.org/rfc/rfc2255.txt

[X501] ITU-T, "Information Technology - Open Systems Interconnection - The Directory: The Models", Recommendation X.501, August 2005, http://www.itu.int/rec/T-REC-X.501-200508-S/en

2 Overview

Section 1, "Introduction", describes this Protocol Family System Document. This section introduces the system that is being documented.

2.1 System Summary

The Active Directory System provides a directory service. A directory service is a service that stores and organizes **directory objects** in a centralized, hierarchical data store. This hierarchical organization of objects is called the **directory**. A directory object is an object that contains one or more **attributes**. Each attribute can have one or more values. Directory objects are identified by a name that is unique among all directory objects in the directory service. The directory objects are organized in a hierarchical manner with regards to other directory objects. For example, a directory service could have a container directory object named Users, the contents of which (referred to as child directory objects) are containers named for each logical division of users (for example, Accounting Department, Human Resources Department, Engineering Department, and so on). The contents of each of these containers, in turn, could be user objects, each of which represents one individual user and contains attributes that store information about that user, such as his or her user name, password, or telephone number.

The Active Directory System can operate in two distinct modes: as **Active Directory Domain Services (AD DS)**, and as **Active Directory Lightweight Directory Services (AD LDS)**. AD LDS consists of a directory service that is accessible via the **Lightweight Directory Access Protocol (LDAP)** versions 2 and 3. AD LDS is primarily intended for use by application software as a storage mechanism and is only available in certain versions of the Active Directory System.<1>

AD DS is also accessible via LDAP versions 2 and 3, but it extends the basic directory service to include additional capabilities (namely, the ability to host **domain naming contexts**) and additional protocols. This permits AD DS to store the account information for the users of a computer network. The collection of accounts stored in AD DS is referred to as a domain. Such account storage is a vital function of the Active Directory System, and in particular AD DS. However, the Active Directory System is not limited to storing such information. Any information that can be represented as a collection of attribute/value pairs (including the possibility of multivalued attributes) can be modeled as a directory object and stored in the Active Directory System.

Except where noted otherwise, information in this document applies to both AD DS and AD LDS. The Active Directory System encompasses both AD DS and AD LDS.

Physically, the Active Directory System consists of one or more computer servers running a directory service. In the case of both AD DS and AD LDS, these computers are referred to as **domain controllers**. Even though the directory service can be running on multiple computers, these computers replicate the contents of the directory such that a client sees a consistent view of the directory no matter which directory server or domain controller it communicates with.

Note that this document, like [MS-ADTS] and [MS-DRSR], uses the term "domain controller" to refer to a directory service running as either AD DS or AD LDS. Both AD DS and AD LDS are considered to be directory servers.

Directory objects can be created and deleted in the directory. Subsequent to its creation, the contents of a directory object can be modified by adding or removing attributes and their values, or by changing the values of existing attributes. Clients can retrieve the contents of directory objects either by reading the attributes of a specific object or by querying for any objects which match client-specified criteria.

The core protocol used by clients to perform these operations is LDAP version 3 as described in [MS-ADTS]. Clients can also communicate with the **Active Directory** System using the network protocols described in [MS-ADTS], [MS-DRSR], [MS-SAMR], [MS-LSAD], [MS-LSAT], and [MS-DSSP], as well as the Web Service protocols described in [MS-ADDM], [MS-WSTIM], [MS-ADCAP], [MS-WSDS], and [MS-WSPELD]. The set of protocols supported by the Active Directory System depends on whether the system is running as AD DS or AD LDS (see section 4.5).

While this document provides a description of the client-server functionality of the Active Directory System, there are additional specific benefits that a client obtains from being associated with a AD DS directory service ("joined to a domain") that are not described in this document. Nor does this document describe how being domain-joined affects the client or the domain-related interactions between the client and the server (such as the ability of the client to perform **Kerberos** security authentication by virtue of being joined to a domain). This information is instead provided in the Domain Integration System Overview document [MS-DISO]. Also, this document does not describe identity and security concepts that are defined as part of the Windows System Overview document [MS-WSO].

2.2 List of Member Protocols

The following are the member protocols and protocol extensions that make up the Active Directory System. Section 4.5 gives the details about which protocols (or protocol subsets) are supported in the different modes of operation.

ADCAP: The Active Directory Web Services Custom Action Protocol as defined in [MS-ADCAP]. It is a **SOAP**-based Web Services protocol for managing account and topology information.

DS RPC: The Directory Replication Service (DRS) Remote Protocol as defined in [MS-DRSR]. This includes only the drsuapi **remote procedure call (RPC)** interface. Methods on this interface provide a variety of functionality to clients, such as converting names between formats and retrieving information about AD DS domain controllers.

DSSP: The Directory Services Setup Remote Protocol as defined in [MS-DSSP]. This protocol can be used to retrieve information regarding the state of a computer in a domain or a non-domain workgroup.

IMDA: The WS-Transfer Identity Management Operations for Directory Access Protocol Extensions as defined in [MS-WSTIM]. This is a set of protocol extensions to WS-Transfer that allows directory objects to be manipulated at a finer level of granularity than unextended WS-Transfer.

LDAP: The Lightweight Directory Access Protocol, which can be either version 2 (defined in [RFC1777]) or version 3 (defined in [RFC3377]), and taking into account [MS-ADTS] section 3.1.1.3.1, LDAP Conformance, including the extensions to the LDAP protocol specified there. When a reference to LDAP is made without a version number, it is intended to refer to both versions 2 and 3. This is the primary protocol for accessing the directory. It includes operations to add, modify, remove, and query for directory objects.

LSAD: The Local Security Authority (Domain Policy) Remote Protocol as defined in [MS-LSAD]. Clients can use this protocol to retrieve security policy information.

LSAT: The Local Security Authority (Translation Methods) Remote Protocol as defined in [MS-LSAT]. Clients can use this protocol to translate **security identifiers (SIDs)** of **security principals** to human-readable names, and vice versa.

SAMR: The Security Account Manager (SAM) Remote Protocol as defined in [MS-SAMR]. Clients can use this protocol to perform account maintenance, such as creating and deleting accounts. Much of this functionality can also be performed using the LDAP protocol.

WS-Enumeration: The WS-Enumeration protocol as defined in [WSENUM]. This protocol allows directory objects to be queried using a SOAP-based Web Services protocol.

WS-Transfer: The WS-Transfer protocol as defined in [WXFR]. This protocol allows directory objects to be created, removed, modified, and read using a SOAP-based Web Services protocol.

WSDS: The WS-Enumeration Directory Services Protocol Extensions as defined in [MS-WSDS]. This is a set of protocol extensions to WS-Enumeration that, among other things, allows a client to request that the query results be sorted. It also specifies a query language that is used by clients to specify which directory objects are to be returned from the query.

WSPELD: WS-Transfer Lightweight Directory Access Protocol (LDAP) v3 Control Extension, as defined in [MS-WSPELD]. This is a protocol extension to WS-Transfer and WS-Enumeration. It permits LDAP **extended controls** to be attached to operations in the protocols that it extends.

2.3 Relevant Standards

The following standards are relevant to the Active Directory System.

LDAP: The Active Directory System conforms to the LDAP version 3 protocol as specified in [RFC3377]. Details of Active Directory's conformance are specified in [MS-ADTS] section 3.1.1.3.1, LDAP Conformance. The Active Directory System also supports LDAP version 2 [RFC1777], although clients are encouraged to use LDAP version 3 instead.

WS-Enumeration: The Active Directory System uses the WS-Enumeration protocol [WSENUM] to query for directory objects. The system extends the standard protocol with the extensions defined in [MS-WSDS] and [MS-WSPELD]. In the Active Directory System, the WS-Enumeration protocol operates over the **XML** data model described in [MS-ADDM].

WS-Transfer: The Active Directory System uses the WS-Transfer protocol [WXFR] to create, modify, remove, and read directory objects. The system extends the standard protocol with the extensions defined in [MS-WSTIM] and [MS-WSPELD]. As with WS-Enumeration, the WS-Transfer protocol operates over the XML data model described in [MS-ADDM] in the Active Directory System.

3 Foundation

This section describes the theoretical and practical information needed to understand this document and this system.

3.1 Background Knowledge and System-Specific Concepts

This section summarizes:

- Background knowledge required to understand this document.
- Concepts that are specific to this system.

Readers of this document should be familiar with the Windows domain model. For those who are not familiar with domains, [MS-DISO] section 3, Background Knowledge and System-Specific Concepts, has useful background material on this concept. Additionally, prior experience with directory services based on the X.500 model [X501], which includes directory services based on the LDAP protocol, although not required, can be helpful. The abstract data model of [MS-ADTS] section 3.1.1, Abstract Data Model, also provides essential background material.

A central concept in the directory service is the **directory tree**. A directory tree is an arrangement of directory objects into a tree structure. Each directory object has exactly one parent directory object, except for the object that serves as the root of the tree and has no parent. Each directory object can have zero or more child objects. Each directory object is assigned a name (the **relative distinguished name (RDN)**) that is unique among its sibling objects. Each directory object can be uniquely identified from among all the other objects in the directory service by its **distinguished name (DN)**, which is formed by concatenating the RDNs of the directory objects along the path from the root of the tree to the object in question.

For example, in the following diagram, the directory object at the root of the tree has the RDN "DC=Fabrikam".

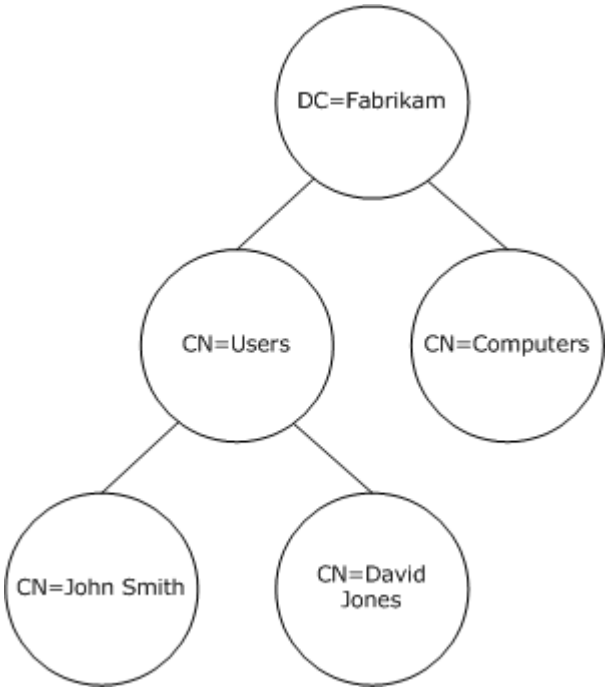

Figure 1: Directory tree

It has two child objects whose RDNs are "CN=Users" and "CN=Computers". The "CN=Users" object, in turn, has two child objects, with the RDNs "CN=John Smith" and "CN=David Jones". Alternatively, one could say that the parent object of the "CN=John Smith" and "CN=David Jones" objects is the object whose RDN is "CN=Users". The DN of the object whose RDN is "CN=Users" is "CN=Users,DC=Fabrikam", while the DNs of its child objects are "CN=John Smith,CN=Users,DC=Fabrikam" and "CN=David Jones,CN=Users,DC=Fabrikam".

A directory object is a collection of one or more attributes. Some attributes have exactly one value, while other attributes can have one or more values. If the last value is removed from an attribute, then that attribute is considered to no longer be part of that directory object. Therefore, no directory object ever has an attribute containing zero values. Examples of attributes include givenName, which specifies the given (first) name of a user, and member, which stores the membership of a group. The complete list of attributes in the Active Directory System is given in [MS-ADA1], [MS-ADA2], and [MS-ADA3] for AD DS, and in [MS-ADLS] for AD LDS. The attributes that are permitted to be included in the collection of a given directory object depends on the **object class** of that directory object. The list of object classes in the Active Directory System, and what attributes are permitted for each, is given in [MS-ADSC] for AD DS and [MS-ADLS] for AD LDS.

Directory objects and trees are the key concept behind many of the operations exposed by the protocols supported by the Active Directory System. Protocols contain operations to create, delete, read, modify, and query for directory objects. Object creation results in an object being added to the directory tree, while object deletion removes the object from the tree. The Technical Documents for the individual protocols describe the rules for operations against a directory object. For example, some protocol operations require that a directory object have no child objects in order to be deleted, while other operations automatically delete any child objects under the object being deleted.

While all the protocols in the Active Directory System ultimately operate against the directory tree, some protocols, such as SAMR, expose a non-hierarchical, or flat, view of the directory tree. They may also expose only some of the directory objects in the directory tree, and for those objects exposed by the protocol, only some of the attributes may be exposed (and those attributes exposed may be exposed under a different attribute name than used by other protocols). Section 5.1 discusses how the abstract data models of the different Active Directory System protocols map to each other, including how they map to the directory tree that forms the basis of the Active Directory abstract data model described in [MS-ADTS].

This section has provided a general overview of the concepts of trees and objects in the directory service. For a more formal definition of both of these concepts as they apply to the Active Directory abstract data model, see [MS-ADTS] section 3.1.1.1, State Model.

3.1.1 Active Directory Layout of the Directory Information Tree

This section provides a high-level logical view of the layout of Active Directory's Directory Information Tree. Readers of this section should be familiar with the domain model that is documented in [MS-DISO].

Directory objects that are inside of the **AD DS** and AD LDS directory information trees are structured according to the tree structure that is explained in section 3.1. The logical partitioning of an AD DS directory information tree is different from that of an AD LDS directory information tree. The logical partitioning of an AD LDS directory information tree is discussed later in this section.

3.1.1.1 Logical Partitioning for AD DS

Every AD DS directory information tree has the following **naming contexts**:

- Root Domain NC
- Configuration NC
- Schema NC

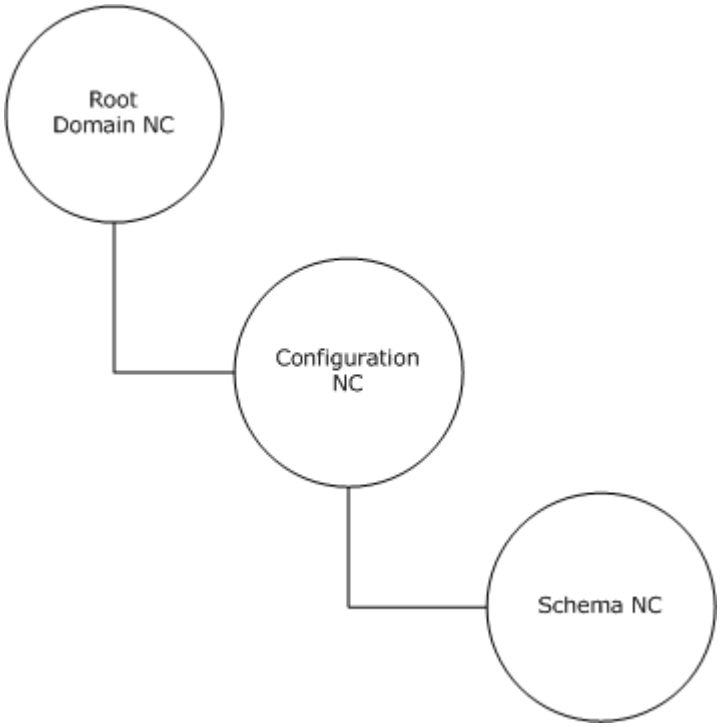

Figure 2: Mandatory naming contexts in an AD DS directory information tree

An AD DS directory information tree may also have the following naming contexts:

- One or more child domain NCs
- One or more application NCs

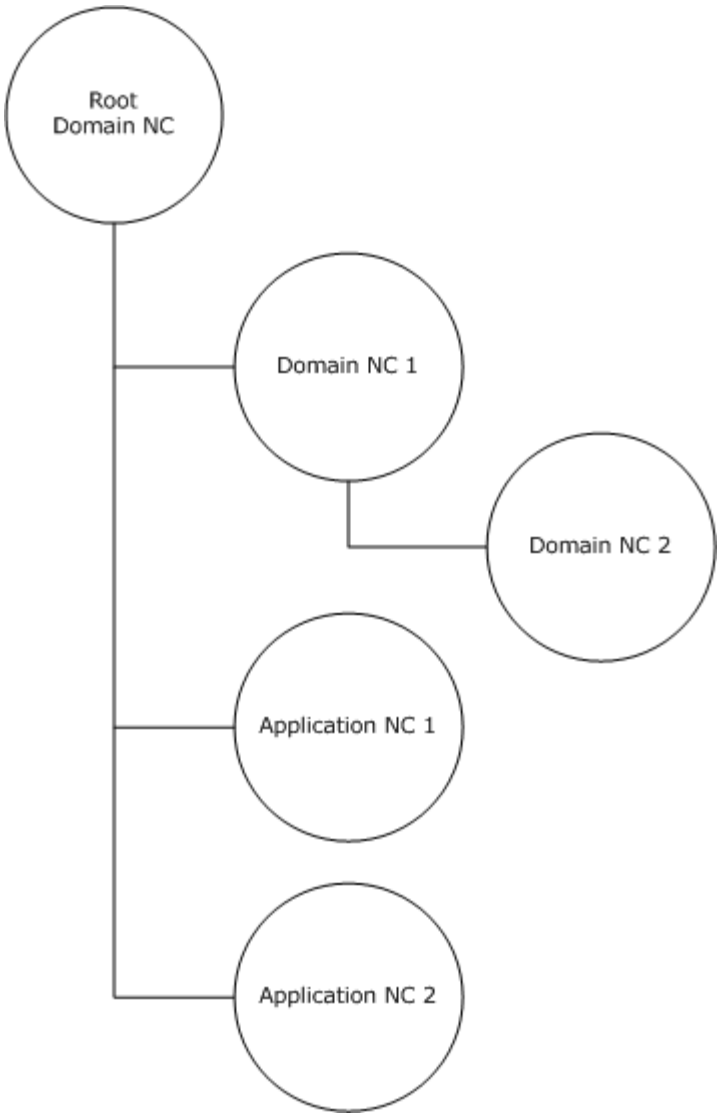

Figure 3: Root Domain NC and additional Domain and Application naming contexts in an AD DS directory information tree

The topmost (or root) object within a given naming context is called the NC root or NC head. This parent-child relationship that is outlined above is formed between naming contexts when a new naming context is created whose relative distinguished name (RDN) is suffixed with the distinguished name (DN) of an existing naming context. Technically these NCs are not related in the same way that a child object and its parent object are related within an NC; the parent relationship stops at the root of an NC. But their DNs are related in the same way as the DNs of a child object and its parent object within an NC. Given NCs with their corresponding DNs forming a child and parent relationship, it is convenient to refer to the NCs as the child NC and the parent NC.

An AD DS **forest** may have one or more domain NCs and zero or more application NCs. Both domain NCs and application NCs may be root objects in the AD DS directory information tree or child objects of existing naming contexts.

3.1.1.1.1 AD DS Naming Context Root Objects

- Every AD DS forest has exactly one forest **root domain** NC, zero or more additional domain NCs, and zero or more application NCs.
- The Configuration NC is a child of the forest root domain NC. The **RDN** of the Configuration NC is "CN=Configuration". The DN of the Configuration NC is the RDN concatenated with the forest root domain NC.
- The Schema NC is a child of the Configuration NC. The RDN of the schema NC is "CN=Schema". The DN of the Schema NC is the RDN concatenated with the Configuration NC.
- Each additional domain or application NC may be a child of the forest root domain NC or may be the root of a separate tree.

See [MS-ADTS] section 3.1.1.1.7, Forest, Canonical Name, for requirements that govern NCs in an Active Directory forest, and [MS-ADTS] section 3.1.1.5.2.6, NC Requirements, for NC requirements.

Consider an example of an AD DS directory tree below.

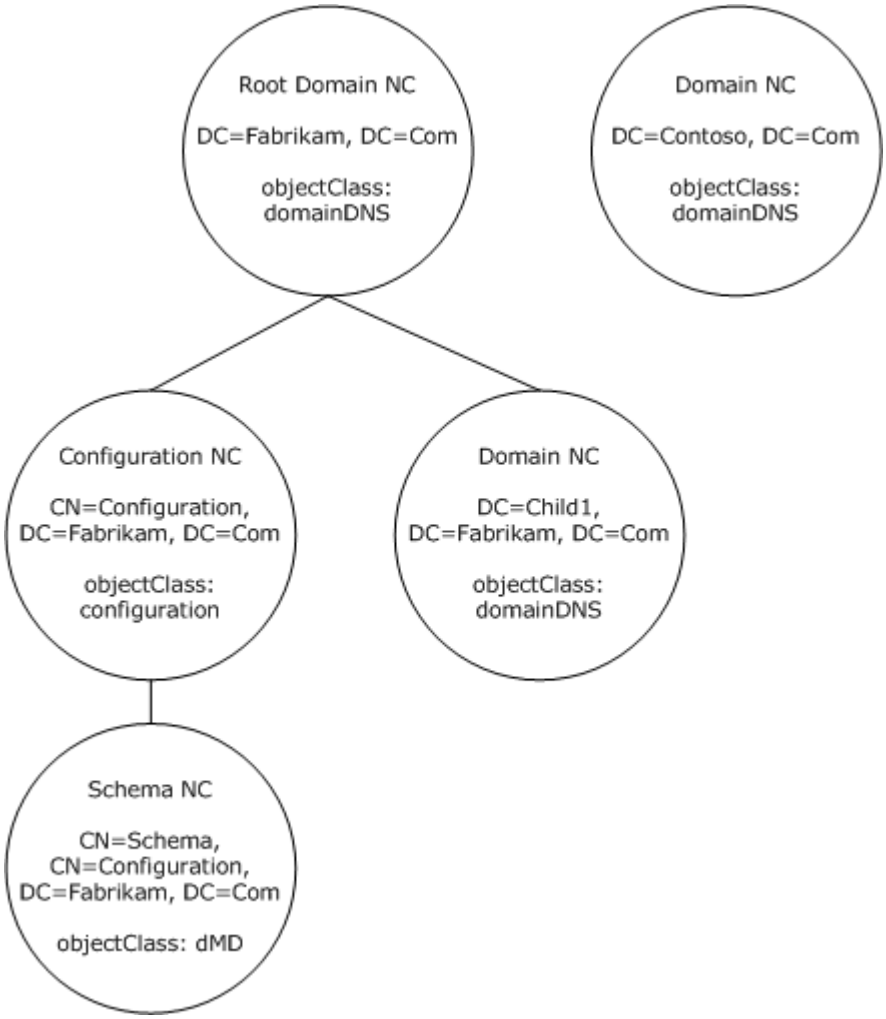

Figure 4: Example of an AD DS directory information tree

In the above example, the root domain NC object of the AD DS forest has the DN "DC=Fabrikam,DC=Com". The AD DS forest has a child domain, whose domain NC object is a child of the root domain NC and has the DN "DC=Child1,DC=Fabrikam,DC=Com".

The AD DS forest also has a third domain whose domain NC root object forms the root of an additional domain tree and has the DN "DC=Contoso,DC=Com".

3.1.1.1.2 Object Structure within the AD DS Naming Contexts

Within each AD DS naming context (NC) (excluding the schema NC), the following well-known system objects exist as child objects:

- Deleted Objects object with RDN "CN=Deleted Objects" and of type container. Objects within a naming context that are deleted are stored under this object (unless indicated otherwise within the schema by the systemFlags attribute of the object class).
- LostAndFound object of type LostAndFound. The LostAndFound objects under all of the NCs with the exception of the configuration NC have the RDN "CN=LostAndFound". The LostAndFound object under the configuration NC has the RDN "CN=LostAndFoundConfig". Objects that are orphaned as a result of the convergence of Add and Delete operations that originated on different AD DS **domain controllers (DCs)** are stored under this object.
- **NTDS** Quotas object with RDN "CN=NTDS Quotas" and of type msDS-QuotaContainer. Objects that represent quotas that restrict the number of objects that can be created by a specified security principal are stored under this object.

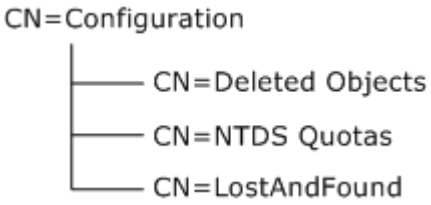

Figure 5: Well Known objects under a configuration naming context in an AD DS directory information tree.

In addition to the well-known objects that are listed above, the following well-known system objects exist within an AD DS Application NC and domain NC:

- Infrastructure object with RDN "CN=Infrastructure" and of class infrastructureUpdate. This object maintains a reference to the current Infrastructure Master role owner for the NC.

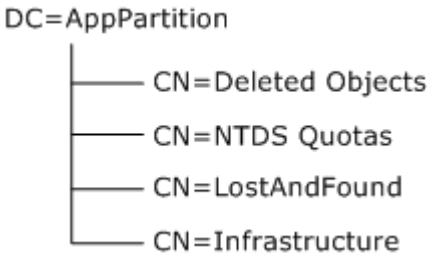

Figure 6: Well Known objects under an application naming context in an AD DS directory information tree

In addition to the well-known objects listed above, the following well-known system objects exist within an AD DS domain NC:

- Computers object with RDN "CN=Computers" and of class container. By default, new computer objects in the domain are stored under this object unless the computer object is explicitly created elsewhere.
- Domain Controllers object with RDN "OU=Domain Controllers" and of class organizationalUnit. Computer objects for AD DS domain controllers within the domain are stored under this object.
- ForeignSecurityPrincipals object with RDN "CN=ForeignSecurityPrincipals" and of class container. Objects that represent security principals from trusted domains that are external to the AD DS forest are stored under this object.
- Program Data object with RDN "CN=Program Data" and of class container. Application-specific data objects are stored under this object, for example, Microsoft application specific data is stored under a child object of Program Data, with RDN "CN=Microsoft".
- System object with RDN "CN=System" and of class container are stored under this object.
- Users object with RDN "CN=Users" and of class container. By default, new user objects in the domain are stored under this object unless the user object is explicitly created elsewhere.

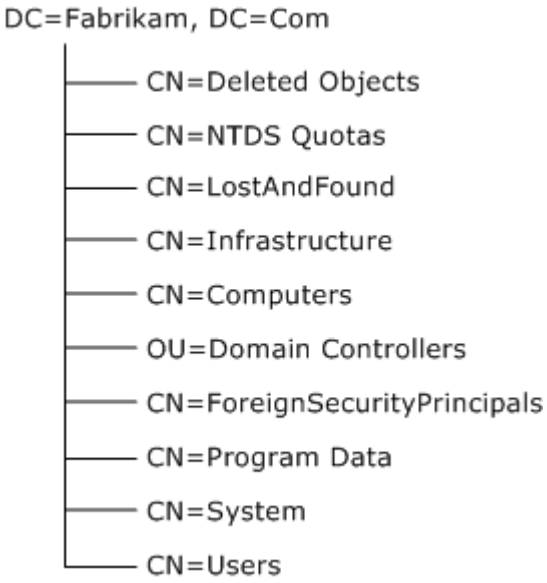

Figure 7: Well Known objects under a domain naming context in an AD DS directory information tree.

Refer to [MS-ADTS] section 6.1.1.4, Domain NC Root, for more details on the well-known system objects.

3.1.1.2 Logical Partitioning for AD LDS

The AD LDS directory information tree is structured in the same way as an AD DS directory information tree with the exception that an AD LDS directory information tree does not contain any domain NC objects.

Every AD LDS directory information tree has the following naming contexts:

- Configuration naming context
- Schema naming context

An AD LDS directory information tree may also have zero or more application naming contexts.

3.1.1.2.1 AD LDS Naming Context Root Objects

- The root object of the AD LDS directory information tree is the NC head of the configuration naming context and has the **DN** CN=Configuration, CN=<GUID Value>, where the <GUID Value> is randomly generated when the AD LDS "configuration set" is created.
- The NC root object of the schema naming context is a child object of the configuration NC. The RDN of the schema NC is "CN=Schema" and is of class dMD.
- The NC root object of the application naming context can be a new root object that is in the directory information tree or a child of another application NC. The application NC root objects may be of any structural classSchema type that is available in the schema of the AD LDS "configuration set".

Consider an example of an AD LDS directory tree below.

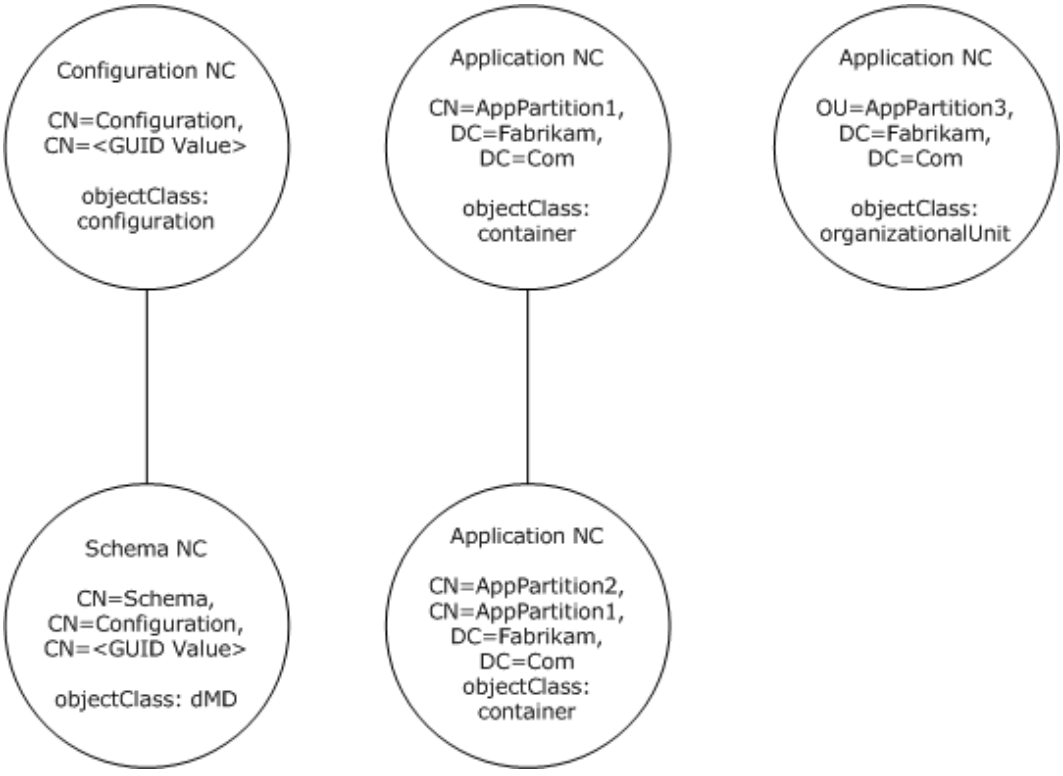

Figure 8: Example of an AD LDS directory information tree

In the above example, the AD LDS directory information tree has three application naming contexts with DNs "CN=AppPartition1,DC=Fabrikam,DC=Com",

"CN=AppPartition2,CN=AppPartition1,DC=Fabrikam,DC=Com", and "OU=AppPartition3,DC=Fabrikam,DC=Com".

3.1.1.2.2 Object Structure within the AD LDS Naming Contexts

Under each AD LDS naming context (excluding the schema NC), the following well-known system objects exist:

- Deleted Objects object with RDN "CN=Deleted Objects" and of type container. Objects within a naming context that are deleted are stored under this object, unless indicated otherwise within the schema by the systemFlags attribute of the object class.
- LostAndFound object of type LostAndFound. The LostAndFound objects under all of the NCs, with the exception of the configuration NC, have the RDN "CN=LostAndFound". The LostAndFound object under the configuration NC has the RDN "CN=LostAndFoundConfig". Objects that are orphaned as a result of the convergence of Add and Delete operations that originated on different **DCs** are stored under this object.
- NTDS Quotas object with RDN "CN=NTDS Quotas" and of class msDS-QuotaContainer. Objects that represent quotas that restrict the number of objects that can be created by a specified security principal are stored under this object.
- Roles object with RDN "CN=Roles" and of class container. Objects that represent the well-known AD LDS groups for the NC are stored under this object.

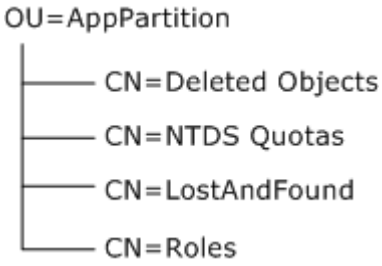

Figure 9: Well-Known objects under an application naming context in an AD LDS directory information tree.

In addition to the above listed well-known objects, the following well-known system objects exist under the AD LDS Configuration NC object:

- ForeignSecurityPrincipals object with RDN "CN=ForeignSecurityPrincipals" and of class container. Objects that represent security principals that are external to the AD LDS "configuration set" are stored under this object.

Figure 10: Well-Known objects under a configuration naming context in an AD LDS directory information tree.

See [MS-ADTS] 6.1.1.4 for more details on the well-known system objects.

3.2 System Purposes

The Active Directory System provides a directory service that applications can use to store and retrieve directory objects. When operating as AD DS, the directory service is used to provide services to domain-joined client computers as discussed in [MS-DISO]. Active Directory is used by other systems, such as Group Policy and Print Services, to store information in a manner that is discoverable to their clients.

The following diagram illustrates a high-level view of the Active Directory System:

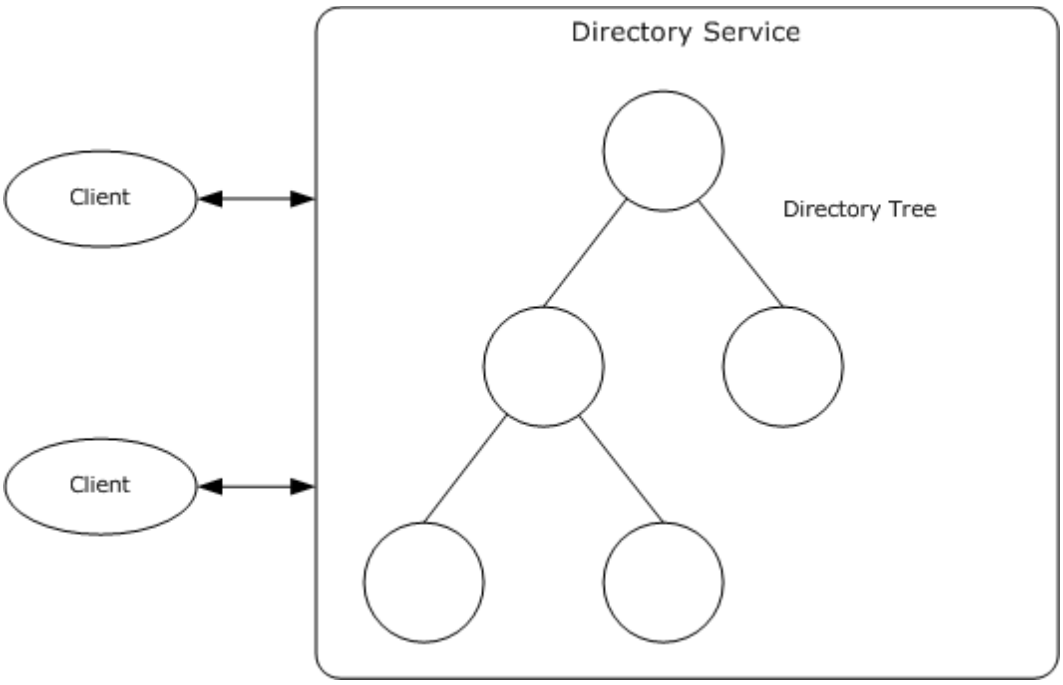

Figure 11: Active Directory System overview

The directory service is hosted on a computer known as a directory server. Clients send requests to the directory server using one of the supported protocol interfaces (see section 5.4). As shown in section 4.5, the interfaces that are supported for use by clients depend on the version of the Active Directory System and whether it is running as AD DS or AD LDS.

Clients manipulate and query for directory objects that are stored in a hierarchically-organized directory tree in the directory service. The Active Directory System enforces security checks to ensure that a client can read and write only those directory objects to which it has the appropriate access permissions. This requires the client to authenticate itself to the directory service (unless it is performing unauthenticated, that is, anonymous, access), and for the directory service to validate that client's authentication (unless the client is performing unauthenticated access).

In addition to providing a directory service to clients that provides for the storage and retrieval of directory objects, the Active Directory System provides the following capabilities.

3.2.1 Domain Integration

The Active Directory System, running as the AD DS service on an AD DS domain controller, provides services and benefits to domain-joined clients as described in [MS-DISO]. These benefits include support for the Kerberos authentication protocol, which domain-joined clients can use to authenticate to the AD DS domain controller and to each other. The benefits also include **certificate** autoenrollment, which automatically deploys certificates to domain-joined computers and to users whose accounts are stored in the directory service [MS-CAESO].

3.2.2 Message Security

Clients can protect requests that they send to a directory server from eavesdropping and tampering. The same mechanisms also protect the responses that the client receives from the directory. More information about security in the Active Directory System is available in section 7.

3.3 System Use Cases

3.3.1 Stakeholders and Interests Summary

The stakeholders in the Active Directory System are system architects, application architects, developers, testers, information technology operations personnel, and client applications.

3.3.1.1 System Architects

A system architect is responsible for the overall design of a system, making decisions about trade-offs in the design and managing technical risk.

The Active Directory protocol documentation is required by system architects who are designing a directory service system that is intended to be compatible with the Active Directory System. Centralized storage and management of identity information (such as user accounts and groups) and application-specific data is a common problem for many large organizations. The Active Directory System provides an approach to solving this problem that is supported by a suite of protocols that cover many common uses of such an identity management system.

A system architect can incorporate the Active Directory System as an element of a system that requires support for a directory service or that provides directory services to other systems.

The system also offers the system architect some additional advantages:

- Support is provided for the industry-standard and cross-platform LDAP protocol, providing compatibility with many directory-enabled applications, including those not specifically written to be compatible with the Active Directory System.
- When operating as AD DS, it provides the ability to integrate into a domain environment, as described in [MS-DISO], and to provide domain services to clients in that environment.
- An extensible data model (the directory **schema**) is available to applications. It permits those applications to declaratively define the structure of their stored data without requiring changes to the design or implementation of the Active Directory System. The structure of the stored data can be defined even after the system has been deployed and is operating in production.

3.3.1.2 Application Architects

An application architect is responsible for the design of software that must interact with the Active Directory System.

The Active Directory protocol documentation is required by application architects who are designing software that will interact with an implementation of the Active Directory System. The system is particularly useful for those who are designing applications that need to store data in a high-availability network-accessible store. This is because the Active Directory System provides a design for such a store, and the application architect can take advantage of it without having to design his or her own store. The Active Directory protocol documentation is also very useful for system architects who are designing applications that, while they might not store any data of their own, need to interact with data stored by other applications in the Active Directory System.

This document introduces the protocols that application architects will need to incorporate into their designs in order to communicate with the system. It also explains the state model of the system, which is useful since it is that state in which they will be storing their data or interacting with data stored by other systems.

For applications that require authentication, the Active Directory System can provide a central account management store. When operating as AD DS, this includes integrated support for authentication protocols such as Kerberos, as described in [MS-AUTHSO]. When operating as AD LDS, more limited support for authentication is provided, as described in [MS-ADTS] section 5.1.1.5, Supported Types of Security Principals.

3.3.1.3 Developers

Developers implement a design, whether that design is for an implementation of the Active Directory System itself or for an application that interacts with the Active Directory System.

Developers building an Active Directory System must understand the protocols documented in the Active Directory protocol documentation in order to build a server-side implementation of those protocols. Developers building an application that interacts with the Active Directory System must understand those same protocols so that they know how to make requests of the system and how to interpret the responses.

The system also offers developers some additional advantages:

- Since the Active Directory System's core protocol is industry-standard LDAP (except for the differences noted in [MS-ADTS] section 3.1.1.3.1, LDAP Conformance), and its data model is based on that of the LDAP data model, the developer can use any code libraries he or she already has for handling LDAP.
- Application developers can use the Active Directory System to store data that needs to be shared across multiple computers or multiple applications without having to implement their own sharing mechanisms and without having to deal with the implementation complexities of building such a system themselves (such as handling simultaneous or conflicting updates from multiple clients or verifying which portions of the store a given client is authorized to read or write).
- If the application requires authentication, the application developer can use the authentication mechanisms of the Active Directory System without having to implement the security protocols him- or herself and without having to understand the cryptographic basis of authentication protocols like Kerberos.

3.3.1.4 Testers

Testers (quality assurance personnel) are responsible for testing an implementation to ensure that it conforms to the design and functionality requirements.

A tester of an implementation of the Active Directory System, or of an application that uses the Active Directory System, must be knowledgeable about the protocols documented in the Active

Directory protocol documentation in order to ensure that the implementation conforms to the protocols. Use of the existing Active Directory protocols and designs means that the testers can focus on testing the implementation rather than on whether the protocol design satisfies the functionality requirements, as they would have to do if the design used an entirely new protocol. If testing an application that uses the Active Directory System for authentication or data storage, then they do not have to test those aspects of the application because it is leveraging an existing trusted system. If testing an implementation of the Active Directory System, testers can leverage the use cases and examples contained in this document to build test cases.

As an additional advantage, testers can use numerous existing tools to view any data stored in the Active Directory System. This permits them to quickly verify whether an application is storing the data it is supposed to store and whether it is storing it in the correct format.

3.3.1.5 Information Technology Operations Personnel

Operations personnel, such as network administrators, are responsible for deploying, configuring, and monitoring the system.

An understanding of the content in the Active Directory protocol documentation is useful to operations personnel because they will need to use many of these protocols to provision information into the system (such as creating and deleting user accounts) and to generate reports about the state of the system (such as verifying that no unexpected accounts are present).

Another advantage of the Active Directory System is that, by providing a centralized store for information used by applications in the enterprise, it reduces the number of distinct information stores that the operations personnel are responsible for maintaining. Instead of having to oversee one information store per application, the operations staff need only oversee a single store that spans all of their organization's applications.

3.3.1.6 Client Applications

Client Applications are used to access and manipulate data stored in the Active Directory System.

A client application uses the protocols documented in the Active Directory protocol documentation to communicate with and send requests to the directory.

3.3.2 Supporting Actors and System Interests Summary

The Active Directory System has the following supporting actor:

Windows Authentication Services: In order to authenticate clients, the Active Directory System uses the Windows Authentication Services [MS-AUTHSO]. The directory service uses the client's authenticated identity to make access-control decisions, such as whether a client has permission to perform a particular operation against a directory object. See section 4.3.1.

This supporting actor applies to all of the Active Directory System's use cases, since authentication and access control is required for proper functionality system-wide.

3.3.3 Use Case Diagrams

This section and the following section describe a set of use cases that span the functionality of the Active Directory System, excluding the domain integration use cases that are found in [MS-DISO].

Detailed descriptions for these use cases are provided in section 3.3.4.

Use case group	Use cases
Object Management	Create Directory Object in Application NC – Client Application Search for Directory Object – Client Application Modify Directory Object – Client Application Delete Directory Object – Client Application Create Organizational Unit – Client Application
Identity Lifecycle Management	Create a New Account – Client Application Reset an existing account's password – Client Application Change an existing account's password – Client Application Query an account's group membership – Client Application Delete an account – Client Application Create a security group – Client Application Update group member list – Client Application Query members of a group – Client Application
Schema Management	Add a new class to the schema - Client ApplicationAdd a new attribute to the schema - Client ApplicationAdd an attribute to a class - Client Application
Name Translation	Convert a SID to/from a human-readable format – Client Application

With the exception of the "SID Translation" use case within the "Name Translation" group which applies only to AD DS mode, all categories are applicable to both the AD DS and AD LDS modes.

The following use case diagrams illustrate the separate groups of use cases described in this section.

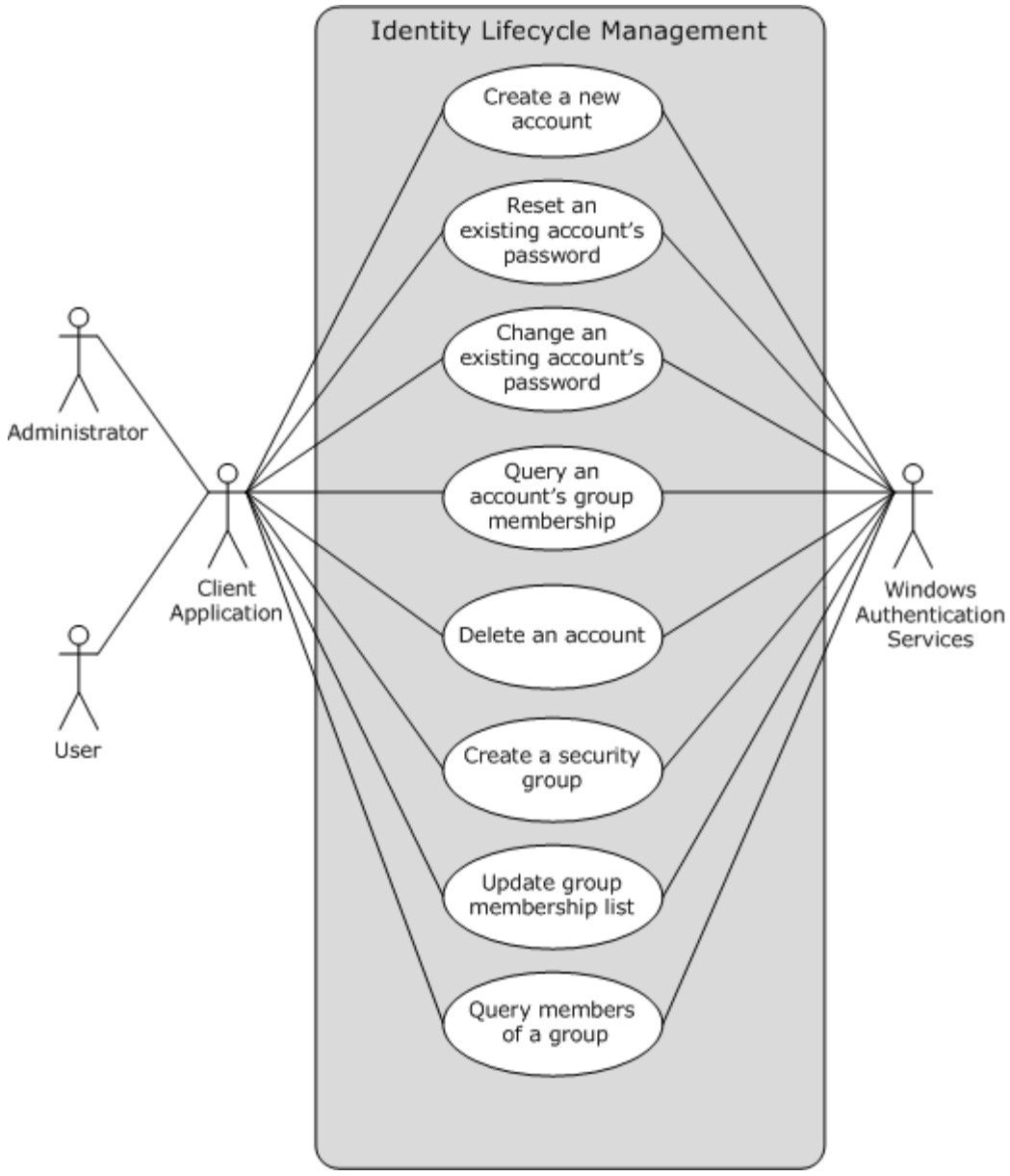

Figure 12: Use Case Group: Identity Lifecycle Management

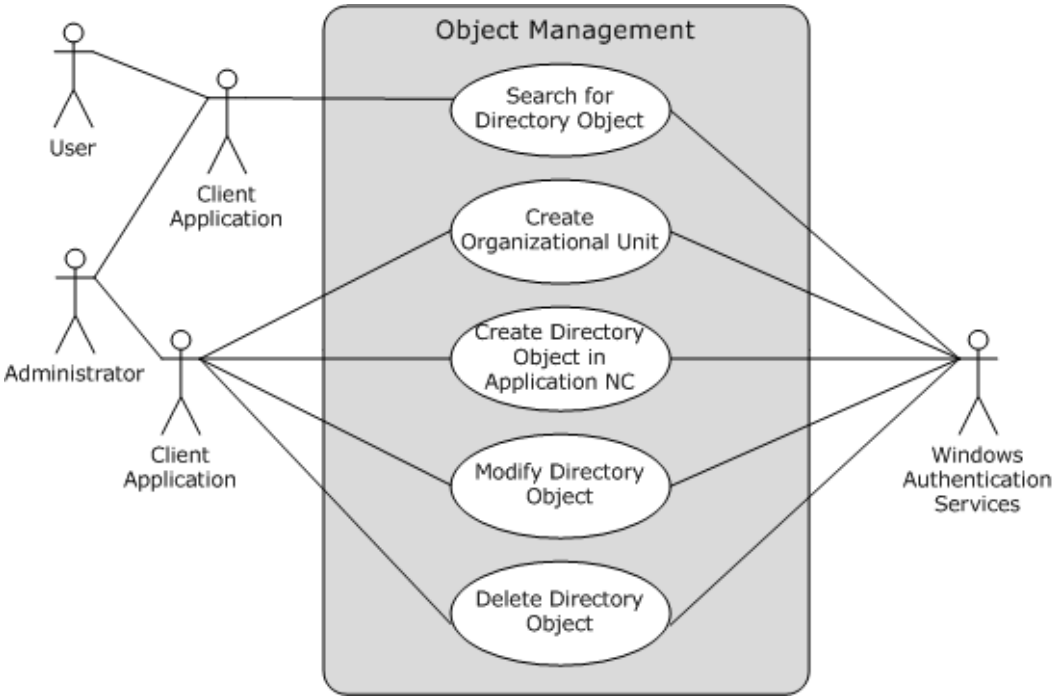

Figure 13: Use Case Group: Object Management

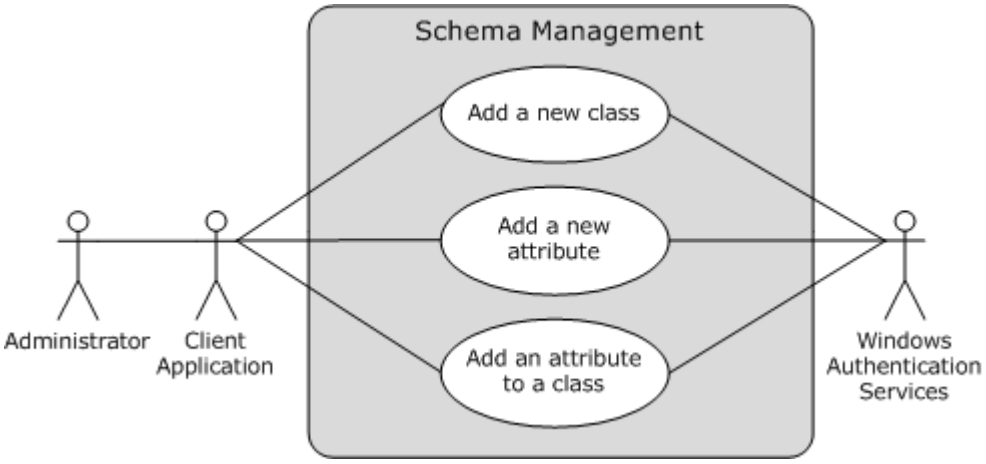

Figure 14: Use Case Group: Schema Management

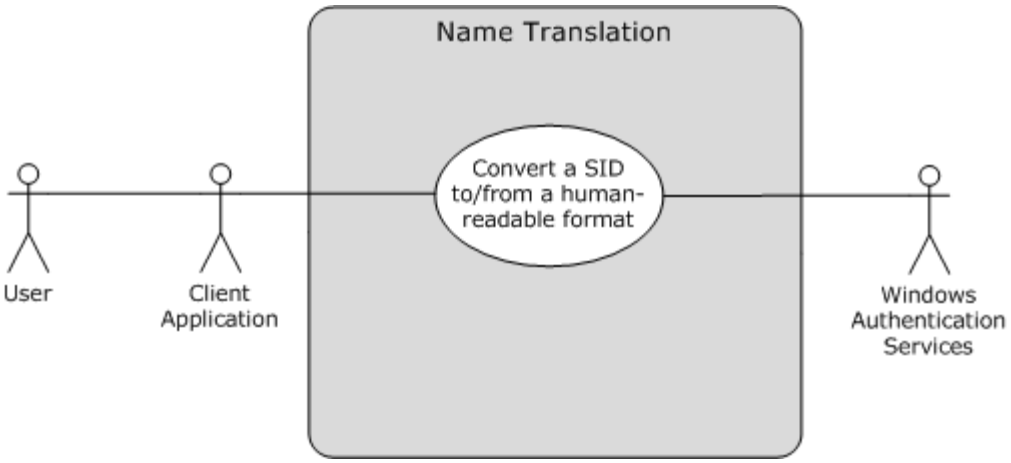

Figure 15: Use Case Group: Name Translation

3.3.4 Use Case Descriptions

This section contains a description of the individual use cases, organized by the four categories described in the previous section.

3.3.4.1 Object Management

The use cases in this category reflect the most fundamental of operations in the Active Directory System, namely, the storage and retrieval of directory objects. Client Applications perform such operations to store data in the directory service for their own use or to configure and manipulate other systems that rely on data that is stored in the directory service. The lifecycle of a directory object begins with its creation. Once created, clients can retrieve it by issuing a search to the directory service. Clients can also modify attributes of the directory object. When the object is no longer needed, it can be deleted.

3.3.4.1.1 Create Directory Object in Application NC - Client Application

Goal: Create a directory object on an **application NC** to store application related data.

Context of Use: An Administrator wants to create a new directory object in an existing application NC to store information that could be used by Client Applications. To achieve this, the Administrator launches the Client Application to interact with the Active Directory System. The Client Application establishes a connection to the Active Directory System. The Administrator creates the directory object with the Client Application.

Direct Actor: The direct actor is the Client Application.

Primary Actor: The primary actor is the Administrator that is creating the application directory object.

Supporting Actors: The supporting actors are the Windows Authentication Services [MS-AUTHSO]. Windows Authentication Services are responsible for authenticating the User's identity so that access control decisions can be made by the Active Directory System.

Stakeholders and Interests:

- Developer: The Developer relies on the correct behavior of the Active Directory System so that the Client Application functions as expected.
- Tester: The Tester needs to verify that the Client Application functions according to specification.
- Application architect: The application architect designs the Client Application and determines how data is stored in the application NC.

Preconditions:

- The Active Directory System MUST complete initialization as described in section 6.6. The system-wide preconditions specified in section 4.2 MUST be satisfied.
- The direct actor MUST have access to a directory server to which it can establish a connection (if it is not already connected) and send the request.
- There already exists an object class in the Active Directory System Schema that corresponds to the directory object to be created under the application NC. Section 3.3.4.3 describes Schema extensions in detail.
- The Active Directory System MUST host an application NC which the direct actor is configured to store its application data.

Minimal Guarantees: If the main success scenario does not successfully finish, the Active Directory System guarantees that the directory is left unchanged.

Success Guarantee:

- The new directory object is created under the application NC with the name specified by the Client Application and conforms to the Active Directory System's schema rules and constraints.
- The new directory object contains all the attribute values set by the Client Application.

Trigger: The Administrator provides input to the Client Application and invokes an operation that creates the application directory object.

Main Success Scenario:

1. The User launches the Client Application.
2. The Client Application establishes a connection to a directory server and provides credentials supplied by the User. Windows Authentication Services authenticates the User using the supplied credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task).
3. The User provides the name of the directory object to create and other pertinent information to the Client Application.
4. The Client Application sends a request to the directory server asking it to create a new application directory object and specifying details for the new object.
5. The directory server validates that the User has the necessary access-control rights to complete the operation ([MS-ADTS] section 5.1.3, Authorization).
6. The directory server creates an object under the application NC with the name and other attributes supplied by the client. The directory object is additionally populated with attributes that are mandated by the server's processing rules and constraints ([MS-ADTS] sections 3.1.1.5.1, General, and 3.1.1.5.2, Add Operation).

7. The directory server sends back a response to the Client Application that the new application directory object has been created successfully.

Extensions:

- If the User has insufficient access-control rights to create the new application directory object:

  1-4. Same as Main Success Scenario.

  5. The directory server sends a response back to the Client Application indicating that it has insufficient access-control rights to create the new application directory object.

- If the RDN value (directory object to be created) supplied by the User is not unique under the same parent container as required by [MS-ADTS] section 3.1.1.5.2.2, Constraints:

  1-5. Same as Main Success Scenario.

  6. The directory server sends a response back to the Client Application indicating that the object name provided already exists.

- If the directory object creation request does not contain all the mandatory attributes as required in [MS-ADTS] section 3.1.1.2.4.5, Content Rules:

  1-5. Same as Main Success Scenario.

  6. The directory server sends a response back to the Client Application indicating that the missing attribute is required in the request.

- If the directory object to be created under the application NC by the User is a security principal in AD DS:

  1-5. Same as Main Success Scenario.

  6. The directory server sends a response back to the Client Application indicating that the security principal can only be created in a domain NC.

System Details: Additional architectural details of this use case are covered in section 6.1.14.

3.3.4.1.2 Search for Directory Object - Client Application

Goal: Retrieve information from one or more directory objects in the Active Directory System.

Context of Use: An Administrator or User wants to inspect the attribute values for a given set of directory objects in order to make informed decisions concerning their Active Directory System. To achieve this, the Administrator or User launches the Client Application to interact with the Active Directory System. The Client Application establishes a connection to the Active Directory System. The Administrator or User then performs a search on the directory tree.

Direct Actor: The direct actor is the Client Application.

Primary Actor: The primary actor is the Administrator or User that is performing the directory search.

Supporting Actors: The supporting actor is Windows Authentication Services [MS-AUTHSO]. Windows Authentication Services are responsible for authenticating the Administrator's identity so that access control decisions can be made by the Active Directory System.

Stakeholders and Interests:

- Developer: The Developer relies on the correct behavior of the Active Directory System so that the Client Application functions as expected.
- Tester: The Tester needs to verify that the Client Application functions according to specification.
- Application architect: The application architect designs the Client Application and determines how data is accessed in the application NC.

Preconditions:

- The Active Directory System MUST complete initialization as described in section 6.6. The system-wide preconditions specified in section 4.2 MUST be satisfied.
- The direct actor MUST have access to a directory server to which it can establish a connection (if it is not already connected) and send the request.

Minimal Guarantees: The directory is left unchanged.

Success Guarantee:

- The requested information from the directory object(s) matching the Client Application's search criteria is returned, provided that the appropriate access-control rights exist. The directory is left unchanged.

Trigger: The Administrator or User provides input to the Client Application in order to perform a search on the directory tree.

Main Success Scenario:

1. The Administrator or User launches the Client Application.
2. The Client Application establishes a connection to a directory server and provides credentials supplied by the Administrator or User. Windows Authentication Services authenticates the Administrator or User using the supplied credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task).
3. The Administrator or User provides the search criteria for the directory object(s) that are of interest to the Client Application. The search criterion also specifies what information about each object is to be returned.
4. The Client Application sends a request to the directory server asking it to search for the directory object(s) and specifying the search criteria.
5. The directory server validates that the Administrator or User has the necessary access-control rights to complete the operation ([MS-ADTS] section 5.1.3, Authorization).
6. The directory server identifies all directory objects that match the criteria supplied by the Client Application. From the set of directory objects identified, the directory server extracts the information requested by the Client Application.

7. The directory server sends back a response to the Client Application containing the extracted information.

Extensions:

- If the Administrator or User has insufficient access-control rights to search for object(s) on the directory tree:

  1-4. Same as Main Success Scenario.

  5. If the Administrator or User does not have access-control rights to see the directory object at the base of the search the directory server sends a response back to the Client Application indicating that no such object exists in the directory tree. If the Administrator or User does not have privileges to read some of the objects in the search scope, those objects are not returned to the Client Application.

- If the search criteria supplied by the Administrator or User returns a result set that is larger than the configured MaxPageSize ([MS-ADTS] section 3.1.1.3.4.6, LDAP Policies):

  1-5. Same as Main Success Scenario.

  6. The directory server sends a response back to the Client Application indicating that it has exceeded the size limit for the request and returns all results up to the result size limitation.

- If the search criteria supplied by the Administrator or User would potentially return objects located on a different NC:

  1-2. Same as Main Success Scenario.

  3. The Administrator or User provides the search criteria for the directory object(s) that are of interest to the Client Application. The directory server that the Client Application is connected to does not host the objects specified by the search criteria.

  4. The Client Application sends a request to the directory server asking it to search for the directory object(s) and specifying the search criteria.

  5. The directory server determines that another server or NC is better suited to process the search request ([MS-ADTS] section 3.1.1.4.6, Referrals).

  6. The directory server sends back a response to the Client Application indicating that a referral error occurred.

System Details: Additional architectural details of this use case are covered in sections 6.1.6 and 6.1.7.

3.3.4.1.3 Modify Directory Object - Client Application

Goal: Modify a directory object in the Active Directory System.

Context of Use: A common activity for an Administrator is to make modifications on objects. Timely updates on these directory objects ensure the proper functioning of the Active Directory System and that the data in the system is current. To achieve this, the Administrator launches the

Client Application to interact with the Active Directory System. The Client Application establishes a connection to the Active Directory System. The Administrator performs a modification on an existing directory object with the Client Application.

Direct Actor: The direct actor is the Client Application.

Primary Actor: The primary actor is the Administrator that is performing the modification on the directory object.

Supporting Actors: The supporting actor is Windows Authentication Services [MS-AUTHSO]. Windows Authentication Services are responsible for authenticating the Administrator's identity so that access control decisions can be made by the Active Directory System.

Stakeholders and Interests:

- Developer: The Developer relies on the correct behavior of the Active Directory System so that the Client Application functions as expected.
- Tester: The Tester needs to verify that the Client Application functions according to specification.
- Application architect: The application architect designs the Client Application and determines how data is stored in the application NC.

Preconditions:

- The Active Directory System MUST complete initialization as described in section 6.6. The system-wide preconditions specified in section 4.2 MUST be satisfied.
- The direct actor MUST have access to a directory server to which it can establish a connection (if it is not already connected) and send the request.
- The directory object to be modified MUST exist in the Active Directory System.

Minimal Guarantees: If the main success scenario does not successfully finish, the Active Directory System guarantees that the directory is left unchanged.

Success Guarantee:

The Active Directory System guarantees that:

- The directory object is modified successfully and conforms to the Active Directory System's schema rules and constraints.
- All the specified modifications in the request are reflected on the object.

Trigger: The Administrator provides input to the Client Application and invokes an operation that modifies a directory object.

Main Success Scenario:

1. The Administrator launches the Client Application.
2. The Client Application establishes a connection to a directory server and provides credentials supplied by the Administrator. Windows Authentication Services authenticates the Administrator using the supplied credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task).

3. The Administrator provides the name of the directory object to modify to the Client Application including the attribute(s) being modified on the object and the list of modifications to be made to those attributes.

4. The Client Application sends a modify request to the directory server asking it to make the appropriate modifications on the directory object.

5. The directory server validates that the Administrator has the necessary access-control rights to complete the operation ([MS-ADTS] section 5.1.3, Authorization).

6. The directory server modifies the object as specified by the client and makes any additional modifications that are mandated by the server's processing rules and constraints; see [MS-ADTS] sections 3.1.1.5.1, General, 3.1.1.5.3, Modify Operation, and 3.1.1.5.4, Modify DN).

7. The directory server sends back a response to the Client Application indicating that the modifications were completed successfully.

Extensions:

- If the Administrator has insufficient access-control rights on the object to perform the operation:

  1-4. Same as Main Success Scenario.

  5. If the Administrator does not have access-control rights to see the object that is to be modified, the directory server sends a response back to the Client Application indicating that no such object exists. Otherwise, the directory server sends a response back to the Client Application indicating that it has insufficient access-control rights to modify the directory object.

- There are myriad failure scenarios when modifying a directory object in the Active Directory System. The operation must be validated against the server's processing rules and constraints as specified in [MS-ADTS] sections 3.1.1.5.1, General, and 3.1.1.5.3, Modify Operation.

System Details: Additional architectural details of this use case are covered in section 6.1.16.

3.3.4.1.4 Delete Directory Object - Client Application

Goal: Delete a directory object from the Active Directory System.

Context of Use: An Administrator can perform maintenance on an Active Directory System by removing objects that are no longer needed by the Client Application. To achieve this, an Administrator launches the Client Application to interact with the Active Directory System. The Client Application establishes a connection to the Active Directory System. The Administrator performs a delete on an existing directory object.

Direct Actor: The direct actor is the Client Application.

Primary Actor: The primary actor is the Administrator that is requesting the delete on the directory object.

Supporting Actors: The supporting actor is Windows Authentication Services [MS-AUTHSO]. Windows Authentication Services are responsible for authenticating the Administrator's identity so that access control decisions can be made by the Active Directory System.

Stakeholders and Interests:

- Developer: The Developer relies on the correct behavior of the Active Directory System so that the Client Application functions as expected.
- Tester: The Tester needs to verify that the Client Application functions according to specification.
- Application architect: The application architect designs the Client Application and determines how data is stored in the application NC.

Preconditions:

- The Active Directory System MUST complete initialization as described in section 6.6. The system-wide preconditions specified in section 4.2 MUST be satisfied.
- The direct actor MUST have access to a directory server to which it can establish a connection (if it is not already connected) and send the request.
- The directory object to be deleted MUST exist in the Active Directory System.

Minimal Guarantees: If the main success scenario does not successfully finish, the Active Directory System guarantees that the directory is left unchanged.

Success Guarantee:

The directory object is transformed into some form of a deleted object. The specific form of the deleted object (Tombstone, Deleted-Object or Recycled-Object) depends on whether the Recycle Bin optional feature is enabled or not, as described in [MS-ADTS] sections 3.1.1.5.5.1.1, Tombstone Requirements, 3.1.1.5.5.1.2, Deleted-Object Requirements, and 3.1.1.5.5.1.3, Recycled-Object Requirements.

Trigger: The Administrator interacts with the Client Application and selects the directory object to delete and submits the deletion request to the Active Directory System.

Main Success Scenario:

1. The Administrator launches the Client Application.
2. The Client Application establishes a connection to a directory server and provides credentials supplied by the Administrator. Windows Authentication Services authenticates the Administrator using the supplied credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task).
3. The Administrator provides the name of the directory object to delete to the Client Application.
4. The Client Application sends a delete request to the directory server asking it to delete the specified directory object.
5. The directory server validates that the Administrator has the necessary access-control rights to complete the operation ([MS-ADTS] section 5.1.3, Authorization).
6. The directory server deletes the object as specified by the client and makes any additional modifications that are mandated by the server's processing rules and constraints; see [MS-ADTS] sections 3.1.1.5.1, General, and 3.1.1.5.5, Delete Operation.
7. The directory server sends back a response to the Client Application indicating that the deletion was completed successfully.

Extensions:

- If the Administrator has insufficient access-control rights to delete the directory object:

1-4. Same as Main Success Scenario.

5. If the Administrator does not have access-control rights to see the object that is to be deleted, the directory server sends a response back to the Client Application indicating that no such object exists. Otherwise, the directory server sends a response back to the Client Application indicating that it has insufficient access-control rights to delete the directory object.

- If the Administrator attempted a delete on a non-leaf directory object:

1-5. Same as Main Success Scenario.

6. The directory server sends a response back to the Client Application indicating that it cannot delete a non-leaf object ([MS-ADTS] section 3.1.1.5.5.5, Constraints).

- If the Administrator attempted a delete on a directory object that is owned by the system ([MS-ADTS] section 3.1.1.5.5.3, Protected Objects):

1-5. Same as Main Success Scenario.

The directory server sends a response back to the Client Application indicating that it will not perform the operation.

System Details: Additional architectural details of this use case are covered in section 6.1.9.

3.3.4.1.5 Create Organizational Unit - Client Application

Goal: Create an organizational unit to facilitate data partitioning in the Active Directory System.

Context of Use: To streamline directory object management, an Administrator can partition directory data with the creation of organizational units. An organizational unit represents the smallest unit to which an Administrator can assign Group Policy settings or delegate administrative authority. To achieve this, the Administrator launches the Client Application to interact with the Active Directory System. The Client Application establishes a connection to the Active Directory System. The Administrator creates an organizational unit with the Client Application.

Direct Actor: The direct actor is the Client Application.

Primary Actor: The primary actor is the Administrator that is creating the new organizational unit.

Supporting Actors: The supporting actor is Windows Authentication Services [MS-AUTHSO]. Windows Authentication Services are responsible for authenticating the Administrator's identity so that access control decisions can be made by the Active Directory System.

Stakeholders and Interests:

- Developer: The Developer relies on the correct behavior of the Active Directory System so that the Client Application functions as expected.
- Tester: The Tester needs to verify that the Client Application functions according to specification.
- Application architect: The application architect designs the Client Application and determines how data is organized in the application NC.

Preconditions:

- The Active Directory System MUST complete initialization as described in section 6.6. The system-wide preconditions specified in section 4.2 MUST be satisfied.
- The direct actor MUST have access to a directory server to which it can establish a connection (if it is not already connected) and send the request.

Minimal Guarantees: If the main success scenario does not successfully finish, the Active Directory System guarantees that the directory is left unchanged.

Success Guarantee:

The Active Directory System guarantees that:

- The new organizational unit object is created and conforms to the Active Directory System's schema rules and constraints.
- The new organizational unit object contains all the attribute values set by the Client Application.

Trigger: The Administrator provides input to the Client Application and invokes an operation that creates the organizational unit object.

Main Success Scenario:

1. The Administrator launches the Client Application in order to create the new organizational unit.
2. The Client Application establishes a connection to a directory server and provides credentials supplied by the Administrator. Windows Authentication Services authenticates the Administrator using the supplied credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task).
3. The Administrator provides the name of the organizational unit to create to the Client Application.
4. The Client Application sends a request to the directory server asking it to create a new organizational unit and specifying the name for the new organizational unit.
5. The directory server validates that the Administrator has the necessary access-control rights to complete the operation ([MS-ADTS] section 5.1.3, Authorization).
6. The directory server creates an object in the directory representing the new organizational unit with the name and other attributes supplied by the client. The directory object is additionally populated with attributes that are mandated by the server's processing rules and constraints; see [MS-ADTS] sections 3.1.1.5.1, General, and 3.1.1.5.2, Add Operation.
7. The directory server sends back a response to the Client Application indicating that the new organizational unit has been created successfully.

Extensions:

- If the Administrator has insufficient access-control rights to create the new organizational unit:

  1-4. Same as Main Success Scenario.

  5. The directory server sends a response back to the Client Application indicating that it has insufficient access-control rights to create the new organizational unit.

- If the RDN value (organizational unit to be created) supplied by the Administrator is not unique under the same parent container as required by [MS-ADTS] section 3.1.1.5.2.2, Constraints:

  1-5. Same as Main Success Scenario.

  6. The directory server sends a response back to the Client Application indicating that the object name provided already exists.

- If the organizational unit creation request does not contain all mandatory attributes as required in [MS-ADTS] section 3.1.1.2.4.5, Content Rules:

  1-5. Same as Main Success Scenario.

  6. The directory server sends a response back to the Client Application indicating that the missing attribute is required in the request.

System Details: Additional architectural details of this use case are covered in section 6.1.13.

3.3.4.2 Identity Lifecycle Management

The use cases in this category represent the management of accounts in the Active Directory System. These accounts are used to gain access to the Active Directory System. An account's lifecycle begins with its creation. Once created, an account can be used to access the system and perform other operations based on the account's access-control rights. Accounts can be modified to change the state or attributes of the account. When accounts are no longer needed, they can be deleted. An account is a directory object (for example, a user object), so these use cases represent a specialization of the use cases of the previous section.

3.3.4.2.1 Create a New Account - Client Application

Goal: Create a new account in the directory.

Context of Use: An administrator wishes to create a new account in the directory to allow a user to access directory resources. The administrator launches a Client Application to create a new account. The Client Application establishes a connection to the Active Directory System.

Direct Actor: The Direct Actor is the Client Application.

Primary Actor: The Primary Actor is the administrator who creates the new account.

Supporting Actors: The supporting actor is Windows Authentication Services [MS-AUTHSO]. Windows Authentication Services are responsible for authenticating the administrator's identity so that access control decisions can be made by the Active Directory System.

Stakeholders and Interests:

- System architect: A system architect wants account storage centralized in the directory and to ensure that new accounts conform to the directory schema.
- Application architect: An application architect may have their application utilize the new account to access directory resources.

Preconditions:

- The Active Directory System MUST complete initialization, as described in section 6.6. The system-wide preconditions that are specified in section 4.2 MUST be satisfied.
- The Client Application MUST have connectivity to a directory server to which it can establish a connection (if it is not already connected) and send the request.

Minimal Guarantees: If the main success scenario does not successfully finish, the Active Directory System guarantees that the directory is left unchanged.

Success Guarantee:

- The new account is created in the directory.
- The account object is populated with attributes based on the input supplied to the Client Application.

Trigger: The administrator provides input to the Client Application that indicates that it wants to create a new account.

Main Success Scenario:

1. The administrator launches the client application. The administrator provides input to the Client Application that indicates that it wants to create a new account.
2. The Client Application establishes a connection to a directory server and provides credentials that are supplied by the Administrator. Windows Authentication Services authenticates the Administrator using the supplied credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task).
3. The Administrator provides the account name for the new account to the Client Application.
4. The Client Application sends a request to the directory server to ask it to create a new account and specifies the account name for the new account.
5. The directory server validates that the Administrator has the necessary access-control rights to complete the operation ([MS-ADTS] section 5.1.3, Authorization).
6. The directory server validates the constraints on the new account name, as described in [MS-SAMR] sections 3.1.1.6, Attribute Constraints for Originating Updates, and 3.1.1.8.4, sAMAccountName.
7. The directory server creates an object in the directory that represents the new account with the account name that is supplied by the client. The directory object is additionally populated with attributes that are mandated by the server's processing rules and constraints. ([MS-ADTS] section 3.1.1.5.1, General, and section 3.1.1.5.2, Add Operation, [MS-SAMR] section 3.1.1.8, Attribute Triggers for Originating Updates, and section 3.1.1.9, Additional Update Triggers).
8. The directory server sends a response back to the Client Application that indicates the new account has been created successfully.

Extensions:

- If the Administrator has insufficient access-control rights to create the new account:

  1-4. Same as Main Success Scenario.

5. The directory server sends a response back to the Client Application that indicates the administrator has insufficient access-control rights to create the new account

- If the account name that is supplied by the Administrator does not satisfy the account name constraints that are outlined in [MS-SAMR] section 3.1.1.6, Attribute Constraints for Originating Updates:

  1-5. Same as Main Success Scenario.

  6. The directory server sends a response back to the client that indicates that the supplied account name does not meet the constraints.

- If the account name that is supplied by the Administrator is not unique, as required by [MS-SAMR] section 3.1.1.8.4, sAMAccountName:

  1-5. Same as Main Success Scenario.

  6. The directory server sends a response back to the client that indicates that the supplied account name is already in use by an existing account.

System Details: Additional architectural details of this use case are covered in section 6.1.1 and section 6.1.2.

3.3.4.2.2 Reset an Existing Account's Password - Client Application

Goal: Reset the password on an account to a known value.

Context of Use: A User has forgotten the password for their account and contacts an Administrator. The Administrator wants to reset the account's password to a known value so they can communicate the new password to the User. The Administrator launches a Client Application to reset the password on an existing account. The Client Application establishes a connection to the Active Directory System.

Direct Actor: The direct actor is the Client Application.

Primary Actor: The primary actor is the Administrator who is resetting the password on the existing account.

Supporting Actors: The supporting actor is Windows Authentication Services [MS-AUTHSO]. Windows Authentication Services are responsible for authenticating the administrator's identity so that access control decisions can be made by the Active Directory System.

Stakeholders and Interests:

- System architect: A system architect wants to ensure that account passwords conform to the directory policy and constraints.
- Application architect: An application architect wants to use the new password so that their application can logon as the account.

Preconditions:

- The Active Directory System MUST complete initialization as described in section 6.6. The system-wide preconditions specified in section 4.2 MUST be satisfied.

- The Client Application MUST have connectivity to a directory server to which it can establish a connection (if it is not already connected) and send the request.

Minimal Guarantees: If the main success scenario does not successfully finish the Active Directory System guarantees that the directory is left unchanged.

Success Guarantee: The account's password is updated with the new value supplied by the Administrator.

Trigger: The Administrator provides input to the Client Application indicating that it wants to reset the password of an account.

Main Success Scenario:

1. The Administrator launches the Client Application.
2. The Client Application establishes a connection to a directory server and provides credentials supplied by the Administrator. Windows Authentication Services authenticates the Administrator using the supplied credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task).
3. The Administrator provides the account name of the existing account and new password to the Client Application.
4. The Client Application sends a request to the directory server asking it to reset the password of an existing account. This request includes the account name of the account and new password supplied by the Administrator.
5. The directory server validates that the Administrator has the necessary access-control rights to complete the operation. ([MS-ADTS] section 5.1.3, Authorization).
6. The directory server validates that the new password satisfies the password policy as outlined in [MS-SAMR] section 3.1.1.7.1, General Password Policy.
7. The directory server updates the password of the existing account with the new value supplied in the request. Additional attributes are updated as mandated by the server's processing rules and constraints. See [MS-ADTS] sections 3.1.1.5.1, General, and 3.1.1.5.3, Modify Operation, and [MS-SAMR] section 3.1.1.8.7, unicodePwd.
8. The directory server sends a response back to the Client Application indicating that the password has been updated successfully.

Extensions:

- If the Administrator has insufficient access-control rights to set the password on the account:

  1-4. Same as Main Success Scenario.

  5. The directory server sends a response back to the Client Application indicating that the Administrator has insufficient access-control rights to set the password on the account.

- If the password supplied by the Administrator does not satisfy the password constraints outlined in [MS-SAMR] section 3.1.1.7.1, General Password Policy:

  1-5. Same as Main Success Scenario.

6. The directory server sends a response back to the client indicating that the password supplied does not meet the constraints.

- If the account supplied by the Administrator does not exist:

1-4. Same as Main Success Scenario.

5. The directory server sends a response back to the client indicating that the account supplied does not exist.

System Details: Additional architectural details of this use case are covered in section 6.1.1.

3.3.4.2.3 Change an Existing Account's Password - Client Application

Goal: Change the password on an account to a known value.

Context of Use: A User wishes to change their existing password to a new value. The user launches a Client Application to change the password on their account. The Client Application establishes a connection to the Active Directory System.

Direct Actor: The direct actor is the Client Application.

Primary Actor: The primary actor is the User who is changing the password on their account.

Supporting Actors: The supporting actor is Windows Authentication Services [MS-AUTHSO]. Windows Authentication Services are responsible for authenticating the user's identity so that access control decisions can be made by the Active Directory System.

Stakeholders and Interests:

- System architect: A system architect wants to ensure that account passwords conform to the directory policy and constraints.
- Application architect: An application architect wants to use the new password so that their application can logon as the account.

Preconditions:

- The Active Directory System **MUST** complete initialization as described in section 6.6. The system-wide preconditions specified in section 4.2 MUST be satisfied.
- If not already connected, the Client Application MUST have connectivity to a directory server to which it can establish a connection (if it is not already connected) and send the request.
- The account that the password change is being performed on MUST exist.

Minimal Guarantees: If the main success scenario does not successfully finish, the Active Directory System guarantees that the directory is left unchanged.

Success Guarantee: The account's password is updated with the new value supplied by the User.

Trigger: The User provides their existing password the new value for their password to the Client Application in order to change the password of their account.

Main Success Scenario:

1. The User launches the Client Application.
2. The Client Application establishes a connection to a directory server and provides credentials supplied by the User. Windows Authentication Services authenticates the User using the supplied credentials, as specified in [MS-AUTHSO] section 4, Interactive Domain Logon Task.
3. The User provides the account name of the existing account, current password for the account, and new password to the Client Application.
4. The Client Application sends a request to the directory server asking it to change the password of an existing account. This request includes the account name of the account, current password, and new password supplied by the User.
5. The directory server validates that the User has the necessary access-control rights to complete the operation, as specified in [MS-ADTS] section 5.1.3, Authorization.
6. The directory server validates that the current password supplied by the User matches the account's password stored in the directory service.
7. The directory server validates that the new password satisfies the password policy, as specified in [MS-SAMR] section 3.1.1.7.1, General Password Policy.
8. The directory server updates the password of the existing account with the new value supplied in the request. Additional attributes are updated as mandated by the server's processing rules and constraints. See [MS-ADTS] sections 3.1.1.5.1, General, and 3.1.1.5.3, Modify Operation, [MS-SAMR] section 3.1.1.8.7, unicodePwd.
9. The directory server sends a response back to the Client Application indicating that the password has been updated successfully.

Extensions:

- If the User has insufficient access-control rights to set the password on the account:

  1-4. Same as Main Success Scenario.

  5. The directory server sends a response back to the Client Application indicating that the User has insufficient access-control rights to set the password on the account.

- If the current password supplied by the User does not match the password stored in the directory:

  1-5. Same as Main Success Scenario.

  6. The directory server sends a response back to the Client Application indicating that the supplied password is incorrect.

- If the new password supplied by the User does not satisfy the password constraints outlined in [MS-SAMR] section 3.1.1.7.1, General Password Policy.

  1-6. Same as Main Success Scenario.

7. The directory server sends a response back to the client indicating that the password supplied does not meet the constraints.

System Details: Additional architectural details of this use case are covered in section 6.1.3.

3.3.4.2.4 Query an Account's Group Membership - Client Application

Goal: Retrieve an account's group membership.

Context of Use: An Administrator wishes to display an account's group membership in order to determine the account's access rights. The Administrator launches a Client Application to query the group membership of a specified account. The Client Application establishes a connection to the Active Directory System.

Direct Actor: The direct actor is the Client Application.

Primary Actor: The primary actor is the Administrator who is retrieving the account's group membership.

Supporting Actors: The supporting actor is Windows Authentication Services [MS-AUTHSO]. Windows Authentication Services are responsible for authenticating the Administrator's identity so that access control decisions can be made by the Active Directory System.

Stakeholders and Interests:

- Application architect: An application architect wants their application to query the group membership of an account so that their application can determine access.

Preconditions:

- The Active Directory System MUST complete initialization as described in section 6.6. The system-wide preconditions specified in section 4.2 MUST be satisfied.
- The Client Application MUST have connectivity to a directory server to which it can establish a connection (if it is not already connected) and send the request.

Minimal Guarantees: The directory is left unchanged.

Success Guarantee: The account's group membership is determined and returned to the Client Application.

Trigger: The Administrator provides input to the Client Application that indicates it wishes to query the group membership of an account.

Main Success Scenario:

1. The Administrator launches the Client Application.
2. The Client Application establishes a connection to a directory server and provides credentials supplied by the Administrator. Windows Authentication Services authenticates the Administrator using the supplied credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task).
3. The Administrator provides the account name of the account to query to the Client Application.
4. The Client Application sends a request to the directory server asking it to retrieve group membership of the account.

5. The directory server validates that the administrator has the necessary access-control rights to complete the operation ([MS-ADTS] section 5.1.3, Authorization).

6. The directory server sends a response back to the Client Application containing the group membership for the specified account.

Extensions:

- If the User has insufficient access-control rights to retrieve group membership of the account:

  1-4. Same as Main Success Scenario.

  5. The directory server sends a response back to the Client Application. Group membership information is not returned to the Client Application.

- If the account supplied in the request does not exist:

  1-4. Same as Main Success Scenario.

  The directory server sends a response back to the Client Application indicating that the account supplied does not exist.

System Details: Additional architectural details of this use case are covered in section 6.1.4.

3.3.4.2.5 Delete an Account - Client Application

Goal: Delete an account in the directory.

Context of Use: An Administrator wishes to delete an account from the directory to prevent its further use. The Administrator launches a Client Application to delete an account. The Client Application establishes a connection to the Active Directory System.

Direct Actor: The direct actor is the Client Application.

Primary Actor: The primary actor is the Administrator who is deleting the account.

Supporting Actors: The supporting actor is Windows Authentication Services [MS-AUTHSO]. Windows Authentication Services are responsible for authenticating the administrator's identity so that access control decisions can be made by the Active Directory System.

Stakeholders and Interests:

- System architect: A system architect wants to ensure the directory is maintainable and accurate by removing accounts that are no longer needed.
- Application architect: An application architect no longer wishes for their application to utilize the account.

Preconditions:

- The Active Directory System MUST complete initialization as described in section 6.6. The system-wide preconditions specified in section 4.2 MUST be satisfied.

- The Client Application MUST have connectivity to a directory server to which it can establish a connection (if it is not already connected) and send the request.

Minimal Guarantees: If the main success scenario does not successfully finish, the Active Directory System guarantees that the directory is left unchanged.

Success Guarantee: The account is transformed into some form of a deleted object. The specific form of the deleted object (Tombstone, Deleted-Object or Recycled-Object) depends on whether the Recycle Bin optional feature is enabled or not, as described in [MS-ADTS] sections 3.1.1.5.5.1.1, Tombstone Requirements, 3.1.1.5.5.1.2, Deleted-Object Requirements, and 3.1.1.5.5.1.3, Recycled-Object Requirements.

Trigger: The Administrator provides input to the Client Application indicating that they wish to delete the account.

Main Success Scenario:

1. The Administrator launches the Client Application.
2. The Client Application establishes a connection to a directory server and provides credentials supplied by the Administrator. Windows Authentication Services authenticates the Administrator using the supplied credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task).
3. The Administrator provides the account name of the account to delete to the Client Application. The Client Application sends a request to the directory server asking it to delete the account.
4. The directory server validates that the Administrator has the necessary access-control rights to complete the operation ([MS-ADTS] section 5.1.3, Authorization).
5. The directory server deletes the object in the directory representing the account with the account name supplied by the client. Additional processing tasks that are mandated by the server's processing rules and constraints may occur; see [MS-ADTS] section 3.1.1.5.1, General, and section 3.1.1.5.3, Modify Operation.
6. The directory server sends back a response to the Client Application indicating that the account has been deleted successfully.

Extensions:

- If the Administrator has insufficient access-control rights to delete the new account:

  1-4. Same as Main Success Scenario.

  5. The directory server sends a response back to the Client Application indicating that the Administrator has insufficient access-control rights to delete the new account.

- If the supplied account does not exist:

  1-4. Same as Main Success Scenario.

  5. The directory server sends a response back to the Client Application indicating that the account does not exist.

System Details: Additional architectural details of this use case are covered in section 6.1.5.

3.3.4.2.6 Create a Security Group - Client Application

Goal: Create a new security group in the directory.

Context of Use: An administrator wishes to create a security group to be used for access control decisions. The Administrator launches a Client Application to create the new security group. The Client Application establishes a connection to the Active Directory System.

Direct Actor: The direct actor is the Client Application.

Primary Actor: The primary actor is the Administrator who is creating the new security group.

Supporting Actors: The supporting actor is Windows Authentication Services [MS-AUTHSO]. Windows Authentication Services are responsible for authenticating the Administrator's identity so that access control decisions can be made by the Active Directory System.

Stakeholders and Interests:

- System architect: A system architect wants group storage centralized in the directory and to ensure that new groups conform to the directory schema.
- Application architect: An application architect may have their application utilize the new group to control access to directory resources.

Preconditions:

- The Active Directory System MUST complete initialization as described in section 6.6. The system-wide preconditions specified in section 4.2 MUST be satisfied.
- The Client Application MUST have connectivity to a directory **server** to which it can establish a connection (if it is not already connected) and send the request.

Minimal Guarantees: If the main success scenario does not successfully finish the Active Directory System guarantees that the directory is left unchanged.

Success Guarantee: The new security group is created.

Trigger: The Administrator provides input to the Client Application indicating it wishes to create the new security group.

Main Success Scenario:

1. The Administrator launches the Client Application.
2. The Client Application establishes a connection to a directory server and provides credentials supplied by the Administrator. Windows Authentication Services authenticates the Administrator using the supplied credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task).
3. The Administrator provides the group name for the new security group to the Client Application.
4. The Client Application sends a request to the directory server asking it to create a new security group and specifies the group name for the new group.
5. The directory server validates that the Administrator has the necessary access-control rights to complete the operation ([MS-ADTS] section 5.1.3, Authorization).
6. The directory server validates the constraints on the new group name as specified in [MS-SAMR] sections 3.1.1.6, Attribute Constraints for Originating Updates, and 3.1.1.8.4, sAMAccountName.

7. The directory server creates an object in the directory representing the new security group with the group name supplied by the client. The directory object is additionally populated with attributes that are mandated by the server's processing rules and constraints; see [MS-ADTS] section 3.1.1.5.1, General, and section 3.1.1.5.2, Add Operation.

8. The directory server sends back a response to the Client Application indicating that the new security group has been created successfully.

Extensions:

- If the Administrator has insufficient access-control rights to create the new security group:

  1-4. Same as Main Success Scenario.

  5. The directory server sends a response back to the Client Application indicating that the Administrator has insufficient access-control rights to create the new security group.

- If the group name supplied by the Administrator does not satisfy the group name constraints as specified in [MS-SAMR] section 3.1.1.6, Attribute Constraints for Originating Updates:

  - 1-5. Same as Main Success Scenario.

  - 6. The directory server sends a response back to the client indicating that the group name supplied does not meet the constraints.

- If the group name supplied by the Administrator is not unique, as required by [MS-SAMR] section 3.1.1.8.4, sAMAccountName:

  1-5. Same as Main Success Scenario.

  6. The directory server sends a response back to the client indicating that the group name supplied is already in use by an existing account.

System Details: Additional architectural details of this use case are covered in section 6.1.8.

3.3.4.2.7 Update Group Member List - Client Application

Goal: Update the member list of an existing group.

Context of Use: An existing security group is used to control access to directory resources. An Administrator wishes to update the member list of that group so that a new account can access the controlled resources. The Administrator launches a Client Application to update the member list for an existing group. The Client Application establishes a connection to the Active Directory System.

Direct Actor: The direct actor is the Client Application.

Primary Actor: The primary actor is the Administrator who is modifying the member list of the group.

Supporting Actors: The supporting actor is Windows Authentication Services [MS-AUTHSO]. Windows Authentication Services are responsible for authenticating the Administrator's identity so that access control decisions can be made by the Active Directory System.

Stakeholders and Interests:

- System architect: A system architect wants accurate group membership to ensure that access control decisions are made correctly.
- Application architect: An application architect may have their application utilize the updated group membership to control access to directory resources.

Preconditions:

- The Active Directory System MUST complete initialization as described in 6.6. The system-wide preconditions specified in 4.2 MUST be satisfied.
- The Client Application MUST have connectivity to a directory server to which it can establish a connection (if it is not already connected) and send the request.

Minimal Guarantees: If the main success scenario does not successfully finish, the Active Directory System guarantees that the directory is left unchanged.

Success Guarantee: The group's member list is updated with the new values supplied by the Administrator.

Trigger: The Administrator provides input to the Client Application indicating that they wish to update the member list of the group.

Main Success Scenario:

1. The Administrator launches the Client Application.
2. The Client Application establishes a connection to a directory server and provides credentials supplied by the Administrator. Windows Authentication Services authenticates the Administrator using the supplied credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task).
3. The Administrator provides the group name for the group to be updated and the updates to the member list to the Client Application.
4. The Client Application sends a request to the directory server asking it to update the member list of the group specified. The updates for the member list are included in the request.
5. The directory server validates that the Administrator has the necessary access-control rights to complete the operation ([MS-ADTS] section 5.1.3, Authorization).
6. The directory server validates that the new member list satisfies the constraints specified in [MS-SAMR] section 3.1.1.8.9, member.
7. The directory object representing the group is updated with the new member list. Additional processing may occur specified in [MS-ADTS] sections 3.1.1.5.1, General, and 3.1.1.5.3, Modify Operation, and [MS-SAMR] section 3.1.1.8.9, member.
8. The directory server sends a response to the Client Application indicating that the member list has been updated.

Extensions:

- If the Administrator has insufficient access-control rights to update the member list of the group:

  1-4. Same as Main Success Scenario.

5. The directory server sends a response back to the Client Application indicating that the Administrator has insufficient access-control rights to update the member list of the group.

- If the member list supplied by the Administrator does not satisfy the constraints; see [MS-SAMR] section 3.1.1.8.9, member:

  1-5. Same as Main Success Scenario.

  6. The directory server sends a response back to the client indicating that the member list supplied does not meet the constraints.

- If the group supplied by the Administrator does not exist:

  1-4. Same as Main Success Scenario.

  5. The directory server sends a response back to the Client Application indicating that the supplied group does not exist.

System Details: Additional architectural details of this use case are covered in section 6.1.8.

3.3.4.2.8 Query Members of a Group - Client Application

Goal: Retrieve the member list of a group.

Context of Use: An Administrator wishes to view the members of a group to better determine which users have certain access control rights. The Administrator launches a Client Application to query the membership of a specified group. The Client Application establishes a connection to the Active Directory System.

Direct Actor: The direct actor is the Client Application.

Primary Actor: The primary actor is the Administrator who is retrieving the group's member list.

Supporting Actors: The supporting actor is Windows Authentication Services [MS-AUTHSO]. Windows Authentication Services are responsible for authenticating the Administrator's identity so that access control decisions can be made by the Active Directory System.

Stakeholders and Interests:

- Application architect: An application architect wants their application to query the members of a group so that their application can validate access control rights.

Preconditions:

- The Active Directory System MUST complete initialization as described in section 6.6. The system-wide preconditions specified in section 4.2 MUST be satisfied.
- The Client Application MUST have connectivity to a directory server to which it can establish a connection (if it is not already connected) and send the request.

Minimal Guarantees: The directory is left unchanged.

Success Guarantee: The group's member list is determined and returned to the Client Application.

Trigger: The Administrator provides input to the Client Application indicating they wish to query the group's member list.

Main Success Scenario:

1. The Administrator launches the Client Application.
2. The Client Application establishes a connection to a directory server and provides credentials supplied by the Administrator. Windows Authentication Services authenticates the Administrator using the supplied credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task).
3. The administrator provides the group name of the group to query to the Client Application.
4. The Client Application sends a request to the directory server asking it to retrieve the member list for the group.
5. The directory server validates that the Administrator has the necessary access-control rights to complete the operation ([MS-ADTS] section 5.1.3, Authorization).
6. The directory server sends a response back to the Client Application containing the member list for the specified group.

Extensions:

- If the User has insufficient access-control rights to retrieve the member list of the group:

  1-4. Same as Main Success Scenario.

  5. The directory server sends a response back to the Client Application. The member list of the group is not returned to the Client Application.

- If the group supplied by the Administrator does not exist:

  1-4. Same as Main Success Scenario.

  5. The directory server sends a response back to the Client Application indicating that the group does not exist.

System Details: Additional architectural details of this use case are covered in section 6.1.8.

3.3.4.3 Schema Management

When the set of **classes** and **attributes** in the base Active Directory schema do not meet the needs of the applications, the Administrator can extend the schema by adding a new class to the schema, adding a new attribute to the schema, or adding an attribute to a class. Schema classes and attributes are objects stored in Active Directory. Adding a new class to the schema equals to creating a new object of the classSchema class ([MS-ADTS] section 3.1.1.2.4.8, Class classSchema); adding a new attribute to the schema equals to creating a new object of the attributeSchema class ([MS-ADTS] section3.1.1.2.3, Attributes); adding an attribute to a class is done by modifying the classSchema object which defines the class to contain the attribute. After a new class or attribute is added to the schema successfully, users can create the objects of the newly defined or extended schema class.

3.3.4.3.1 Add a New Class to the Schema - Client Application

Goal: An Administrator adds a new class to the schema of the Active Directory System.

Context of Use: When the set of classes in the base Active Directory schema do not meet the needs of a Client Application, the Administrator can extend the schema by adding new objects of the classSchema class. After a new class is added to the schema successfully, the Administrator can create the objects of the new defined class.

Direct Actor: The direct actor is the Client Application.

Primary Actor: The primary actor is the Administrator who is adding a new class to the schema.

Supporting Actors: The supporting actor is Windows Authentication Services [MS-AUTHSO]. Windows Authentication Services are responsible for authenticating the Administrator's identity so that access control decisions can be made by the Active Directory System.

Stakeholders and Interests:

- System architect: The System architect is interested in avoiding redundancy in schema extensions and in ensuring that the correct extensions are deployed to the appropriate directory services.
- Application architect: The application architect designs a data model to store their application's data and designs a set of schema extensions corresponding to this data model.
- Developer: The Developer implements the Client Application to make use of the schema extensions so that the Client Application functions as expected to store data.
- Tester: The Tester needs to verify that the Client Application functions according to the design specification.

Preconditions:

- The Active Directory System MUST complete initialization as described in section 6.6. The system-wide preconditions specified in section 4.2 MUST be satisfied.
- The Client Application MUST have connectivity to the Directory Server to which it can establish a connection (if not already connected) and send the request.

Minimal Guarantees: If the main success scenario does not successfully finish, the Active Directory System guarantees that the directory is left unchanged.

Success Guarantee:

- The new class is added to the schema of the Active Directory System.
- The new class has set all the attributes specified by the Client Application successfully.

Trigger: The Administrator launches the Client Application to add a new class to the schema.

Main Success Scenario:

1. The Administrator launches the Client Application.
2. The Client Application establishes a connection to a Directory Server holding the Schema FSMO role ([MS-ADTS] section 3.1.1.5.1.8, Updates Performed Only on FSMOs) and provides credentials supplied by the Administrator. Windows Authentication Services authenticates the

Administrator using the supplied credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task).

3. The Administrator provides the mandatory attributes ([MS-ADTS] section 3.1.1.2, Active Directory Schema) for the new class to the Client Application.

4. The Client Application sends a request to a Directory Server asking it to create a new class and specifying the values of the attributes present on the classSchema object for the new class.

5. The Directory Server validates that the Administrator has the necessary access-control rights to complete the operation ([MS-ADTS] section 3.1.1.2.5, Schema Modifications).

6. The Directory Server validates that it owns the Schema Master FSMO role ([MS-ADTS] section 3.1.1.2.5, Schema Modifications) to perform the operation.

7. The Directory Server validates the constraints on the new class attributes (as outlined in [MS-ADTS] section 3.1.1.2.5, Schema Modifications).

8. The Directory Server creates an object in the directory representing the new class with the attributes supplied by the Client Application. The directory object is additionally populated with attributes that are mandated by the server's processing rules and constraints; see [MS-ADTS] section 3.1.1.5.1, General, section 3.1.1.5.2, Add Operation, and section 3.1.1.2.5, Schema Modifications.

9. The Directory Server sends back a response to the Client Application indicating that the new class has been added to the schema successfully.

Extensions:

- If the Administrator has insufficient access-control rights to add the new schema class:

  1-4. Same as Main Success Scenario.

  5. The Directory Server sends a response back to the Client Application indicating that it has insufficient access-control rights to add the new class to the scheme.

- If the Directory Server the Client Application connects to does not own the Schema Master FSMO role ([MS-ADTS] section 3.1.1.2.5, Schema Modifications):

  1-4. Same as Main Success Scenario.

  5. The Directory Server sends a response back to the Client Application indicating that a referral is returned from the server.

- If the class name supplied by the Administrator is not unique:

  1-4. Same as Main Success Scenario.

  5. The Directory Server sends a response back to the Client Application indicating that the object name to be created is already in use.

- If the attributes the Client Application provides do not meet the consistency checks ([MS-ADTS] section 3.1.1.2.5.1.1, Consistency Checks):

  1-4. Same as Main Success Scenario.

  5. The Directory Server sends a response back to the Client Application indicating that it will not perform the operation.

System Details: Additional architectural details of this use case are covered in 6.1.10.

3.3.4.3.2 Add a New Attribute to the Schema – Client Application

Goal: An Administrator adds a new attribute to the schema of the Active Directory System by creating an object of class attributeSchema.

Context of Use: When the set of attributes in the base Active Directory schema do not meet the needs of a Client Application, the Administrator can extend the schema by adding new objects of the class attributeSchema. After a new attribute is added to the schema successfully, the Administrator can then add the attribute to a class and hence create objects of the class with the new attribute.

Direct Actor: The direct actor is the Client Application.

Primary Actor: The primary actor is the Administrator who is adding a new attribute to the schema.

Supporting Actors: The supporting actor is Windows Authentication Services [MS-AUTHSO]. Windows Authentication Services are responsible for authenticating the Administrator's identity so that access control decisions can be made by the Active Directory System.

Stakeholders and Interests:

- System architect: The System architect is interested in avoiding redundancy in schema extensions and in ensuring that the correct extensions are deployed to the appropriate directory services.
- Application architect: The application architect designs a data model to store their application's data and designs a set of schema extensions corresponding to this data model.
- Developer: The Developer implements the Client Application to make use of the schema extensions so that the Client Application functions as expected to store data.
- Tester: The Tester needs to verify that the Client Application functions according to the design specification.

Preconditions:

- The Active Directory System MUST complete initialization as described in section 6.6. The system-wide preconditions specified in section 4.2 MUST be satisfied.
- The Client Application MUST have connectivity to the Directory Server to which it can establish a connection (if not already connected) and send the request.

Minimal Guarantees: If the main success scenario does not successfully finish, the Active Directory System guarantees that the directory is left unchanged.

Success Guarantee:

- The new attribute (an object of class attributeSchema) is added to the schema of the Active Directory System.
- The new attribute (an object of class attributeSchema) contains all the attribute values specified by the Client Application.
- The Client Application can add this new attribute to a class.

Trigger: The Administrator launches the Client Application to add a new attribute to the schema.

Main Success Scenario:

1. The Administrator launches the Client Application.
2. The Client Application establishes a connection to a Directory Server holding the Schema FSMO role ([MS-ADTS] section 3.1.1.5.1.8, Updates Performed Only on FSMOs) and provides credentials supplied by the Administrator. Windows Authentication Services authenticates the Administrator by using the supplied credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task).
3. The Administrator provides the mandatory attributes ([MS-ADTS] section 3.1.1.2, Active Directory Schema) for the new object of class attributeSchema to the Client Application.
4. The Client Application sends a request to a Directory Server asking it to create a new attribute (an object of class attributeSchema) and specifying the values of the attributes present on the attributeSchema object.
5. The Directory Server validates that the Administrator has the necessary access-control rights to complete the operation ([MS-ADTS] section 3.1.1.2.5, Schema Modifications).
6. The Directory Server validates that it owns the Schema Master FSMO role ([MS-ADTS] section 3.1.1.2.5, Schema Modifications) to perform the operation.
7. The Directory Server validates the constraints on the new attributeSchema object attributes, as specified in ([MS-ADTS] section 3.1.1.2.5, Schema Modifications).
8. The Directory Server creates an object of class attributeSchema in the directory representing the new attribute with the values of the attributes supplied by the Client Application. The directory object is additionally populated with attributes that are mandated by the server's processing rules and constraints; see [MS-ADTS] section 3.1.1.5.1, General, section 3.1.1.5.2, Add Operation, and section 3.1.1.2.5, Schema Modifications.
9. The Directory Server sends back a response to the Client Application indicating that the new attribute has been added to the schema successfully.

Extensions:

- If the Administrator has insufficient access-control rights to add the new attribute to the schema:

  1-4. Same as Main Success Scenario.

  5. The Directory Server sends a response back to the Client Application indicating that it has insufficient access-control rights to add the new attribute to the scheme.

- If the Directory Server the Client Application connects to does not own the Schema Master FSMO role ([MS-ADTS] section 3.1.1.2.5, Schema Modifications):

1-4. Same as Main Success Scenario.

5. The Directory Server sends a response back to the Client Application indicating that a referral is returned from the server.

- If the attribute name supplied by the Administrator is not unique:

1-4. Same as Main Success Scenario.

5. The Directory Server sends a response back to the Client Application indicating that the object name to be created is already in use.

- If the attributes the Client Application provides do not meet the consistency checks ([MS-ADTS] section 3.1.1.2.5.1.1):

1-4. Same as Main Success Scenario.

5. The Directory Server sends a response back to the Client Application indicating that it will not perform the operation.

System Details: Additional architectural details of this use case are covered in section 6.1.11.

3.3.4.3.3 Add an Attribute to a Class - Client Application

Goal: An Administrator adds an attribute to a class.

Context of Use: When a class in the base Active Directory schema lacks an attribute needed by the Client Application, the Administrator can extend the schema by adding an attribute to an existing class. After the attribute is added to the class successfully, the Administrator can create objects of the extended class with the new added attribute.

Direct Actor: The direct actor is the Client Application.

Primary Actor: The primary actor is the Administrator who is adding an attribute to a class.

Supporting Actors: The supporting actor is Windows Authentication Services [MS-AUTHSO]. Windows Authentication Services are responsible for authenticating the Administrator's identity so that access control decisions can be made by the Active Directory System.

Stakeholders and Interests:

- System architect: The System architect is interested in avoiding redundancy in schema extensions and in ensuring that the correct extensions are deployed to the appropriate directory services.
- Application architect: The application architect designs a data model to store their application's data, and designs a set of schema extensions that correspond to this data model.
- Developer: The Developer implements the Client Application to make use of the schema extensions so that the Client Application functions as expected to store data.

- Tester: The Tester needs to verify that the Client Application functions according to the design specification.

Preconditions:

- The Active Directory System MUST complete initialization, as described in section 6.6. The system-wide preconditions that are specified in section 4.2 MUST be satisfied.
- The Client Application MUST have connectivity to a Directory Server to which it can establish a connection (if not already connected) and send the request.

Minimal Guarantees: If the main success scenario does not successfully finish, the Active Directory System guarantees that the directory is left unchanged.

Success Guarantee:

- The attribute is added to the class.
- The Client Application can create an object of the class and specify the value of the attribute for the object.

Trigger: The Administrator launches the Client Application to add an attribute to a class.

Main Success Scenario:

1. The Administrator launches the Client Application.
2. The Client Application establishes a connection to a Directory Server that holds the Schema FSMO role ([MS-ADTS] section 3.1.1.5.1.8, Updates Performed Only on FSMOs) and provides the credentials that are supplied by the Administrator. Windows Authentication Services authenticates the Administrator that is using the supplied credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task).
3. The Administrator provides the attribute name and the class name to the Client Application.
4. The Client Application sends a request to a Directory Server that asks it to add a new attributeSchema object and specifies the attribute name and the class for this operation.
5. The Directory Server validates that the Administrator has the necessary access-control rights to complete the operation ([MS-ADTS] section 3.1.1.2.5, Schema Modifications).
6. The Directory Server validates that it owns the Schema Master FSMO role ([MS-ADTS] section 3.1.1.2.5, Schema Modifications) to perform the operation.
7. The Directory Server validates the constraints on the attribute and the class, as outlined in [MS-ADTS] section 3.1.1.2.5, Schema Modifications.
8. The Directory Server adds the attribute to the class.
9. The Directory Server sends back a response to the Client Application indicating that the attribute has been added to the class successfully.

Extensions:

- If the Administrator has insufficient access-control rights to modify the class:

1-4. Same as Main Success Scenario.

5. The Directory Server sends a response back to the Client Application indicating that it has insufficient access-control rights to add the attribute to the class.

- If the Directory Server to which the Client Application connects does not own the Schema Master FSMO role ([MS-ADTS] section 3.1.1.2.5, Schema Modifications):

1-4. Same as Main Success Scenario.

5. The Directory Server sends a response back to the Client Application indicating that a referral is returned from the server.

- If the attribute to be added or the class to be modified is not defined in the schema:

1-4. Same as Main Success Scenario.

5. The Directory Server sends a response back to the Client Application indicating that the attribute or the class is not defined in the schema.

- If the Active Directory schema has the attribute to be added already defined in the class:

1-4. Same as Main Success Scenario.

5. The Directory Server sends a response back to the Client Application indicating that the value to be added is already exist.

- If the attributes the Client Application provides do not meet the constraints outlined ([MS-ADTS] section 3.1.1.2.5, Schema Modifications):

1-4. Same as Main Success Scenario.

5. The Directory Server sends a response back to the Client Application indicating that it will not perform the operation.

System Details: Additional architectural details of this use case are covered in section 6.1.12.

3.3.4.4 Name Translation

The use case in this category is used to check how an object is secured in the directory, or to modify the way it is secured.

3.3.4.4.1 Convert a SID to/from a Human-Readable Format - Client Application

This use case is applicable to Active Directory Domain Services only.

Goal: Translate an object's **SID** to or from another format or type of name.

Context of Use: An IT Administrator uses a Client Application to secure access to a directory object. The application displays the human-readable names of the security principals in the **Access**

Control Entries securing the object. The Administrator specifies human-readable names of security principals when securing the object.

Direct Actor: The direct actor is the Client Application.

Primary Actor: The primary actor is the Administrator using the Client Application to secure an object.

Supporting Actors: The supporting actor is Windows Authentication Services [MS-AUTHSO]. Windows Authentication Services are responsible for authenticating the Administrator's identity so that access control decisions can be made by the Active Directory System.

Other Stakeholders and Interests: None.

Preconditions:

- The Active Directory System MUST complete initialization as described in section 6.6. The system-wide preconditions specified in section 4.2 MUST be satisfied.
- The application MUST have network connectivity to a Directory Server that meets the requirements described in section 4.1, to which it can establish a connection (if it is not already connected) and send the request.

Minimal Guarantees: The directory is left unchanged.

Success Guarantee: The server returns to the client the translated names. The directory is left unchanged.

Trigger: Following an action by the Administrator, the Client Application needs to display human-readable names that correspond to the SIDs in the ACLs that secure the object with which the Administrator is interacting with. Alternatively, the Administrator provides a human-readable name to the application to set an ACL on an object, and the application needs to retrieve the SID of the object corresponding to that name for doing so.

Main Success Scenario:

1. The Administrator launches the Client Application.
2. The Client Application establishes a connection to a Directory Server and provides credentials supplied by the Administrator. Windows Authentication Services authenticates the Administrator using the supplied credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task).
3. The Client Application sends a request to the AD DS domain controller to perform name translation between the SID and a human-readable name of directory objects of interest to the Administrator that is interacting with the application.
4. The AD DS domain controller identifies the directory objects for which name translation has to be performed.
5. From the set of directory objects so identified, the AD DS domain controller extracts the requested name format.
6. The AD DS domain controller sends a response back to the client containing the names in the requested format.

Extensions:

- The SID or the name supplied by the Administrator is mis-formatted, as defined in [MS-DTYP] section 2.4.2, SID, and [MS-LSAT] section 3.1.4.5, LsarLookupNames4 (Opnum 77), respectively:

  1-3. Same as Main Success Scenario.

  4. The Directory Server sends a response back to the Client Application indicating that an invalid parameter was used in the request, as indicated in [MS-LSAT] section 3.1.4.9, LsarLookupSids3 (Opnum 76).

- No object exists in the directory with the SID or the name provided:

  1-3. Same as Main Success Scenario.

  4. The Directory Server sends a response back to the Client Application indicating that no object exists with the SID or the name provided.

- Not all SIDs in the request could be translated to names:

  1-3. Same as Main Success Scenario.

  4. The Directory Server sends a response back to the Client Application indicating that only some of the SIDs were translated to names.

- Not all names in the request could be translated to SIDs:

  1-3. Same as Main Success Scenario.

  4. The Directory Server sends a response back to the Client Application indicating that only some of the names were translated to SIDs.

- The client does not have necessary permissions to read the object whose SID or name was supplied:

  1-3. Same as Main Success Scenario.

  4. The Directory Server sends a response back to the Client Application indicating that it has insufficient access-control rights to perform the name translation.

System Details: Additional architectural details of this use case are covered in section 6.1.15.

4 System Context

This section describes the relationship between this system and its environment.

4.1 System Environment

The Active Directory System requires a durable storage system to maintain the state of the directory and meet the transactional guarantees specified in [MS-ADTS] section 3.1.1.5.1.4, Transactional Semantics. This storage system SHOULD provide a means of securing the contents of the storage system from unauthorized access.

The Active Directory System also requires a networking system that clients can use to send requests to the directory server and receive a response. This networking system MUST support the **TCP**, **UDP**, and **Server Message Block (SMB)** transports. This networking system MUST provide an accessible name resolution service through a **Domain Name System (DNS)** service. The DNS service MUST be capable of storing and resolving SRV [RFC2782], A [RFC1034], and CNAME [RFC1034] resource records. It is recommended that an implementation of the Active Directory system use dynamic updates [RFC2136] and [RFC3645] to update DNS with the records as described in [MS-ADTS] section 6.3.2, DNS Record Registrations,<2> but any alternative method that creates the DNS records described there is permissible. DNS can be used by clients of the Active Directory System in order to locate directory servers using the algorithms described in [MS-ADTS] section 6.3, Publishing and Locating a Domain Controller.

Several of the Active Directory System protocols are RPC-based and so depend on the availability of an RPC runtime that implements an RPC mechanism as described in [MS-RPCE].

4.2 System Assumptions and Preconditions

The following assumptions and preconditions MUST be satisfied for the Active Directory System to start operation successfully:

- For AD DS, the server MUST be configured ("promoted") to act as an AD DS domain controller. This is accomplished by having the server host the Active Directory service in AD DS mode. This includes the installation and configuration of any additional services required of an AD DS domain controller as described in [MS-DISO].
- For AD DS, member clients MUST be configured as described in [MS-DISO].
- For AD LDS, the server MUST be configured to host the Active Directory service operating in AD LDS mode.
- A network that provides transport for communications between the directory server and its clients MUST be available. As specified in the previous section, this network MUST supply access to DNS and MUST support the TCP, UDP, and SMB transports.
- The transport protocol for that network MUST be available and configured (for example, the TCP transport must be configured with a valid IP address).
- Support for all authentication mechanisms indicated in each Member Protocol Technical Document of the Active Directory system MUST be available.
- The durable storage system used to store the Active Directory system's state MUST be available to the Active Directory system.
- The directory MUST contain at least the required directory objects and naming contexts described in [MS-ADTS] section 6.1, Special Objects and Forest Requirements.

- The directory's schema MUST contain at least the attribute and class schema definitions described in [MS-ADA1], [MS-ADA2], [MS-ADA3], and [MS-ADSC] (for AD DS); or [MS-ADLS] (for AD LDS) to be compliant with the protocol described in [MS-ADTS]. However, Active Directory currently does not make use of all the attributes and classes defined in the schema definitions.

4.3 System Relationships

This section describes the relationships between the system and external components, system dependencies, and other systems influenced by this system.

4.3.1 Black Box Relationship Diagram

The following figures present the Active Directory System, in context with a client and the Windows Authentication Service and the **DNS** service. The System consists of a directory service, running on a set of replica domain controllers that contains a directory tree to which access is exposed by a variety of protocols to a client. In this section, all protocols and protocol extensions are included in the discussion. However, as described in section 4.5, the actual subset of protocols and protocol extensions that are supported varies depending on the system's version and mode of operation. The following diagram shows the protocols supported by an Active Directory server. Not shown, but supported, are the IMDA, WSDS, and WSPELD protocol extensions to the WS-Transfer and WS-Enumeration protocols.

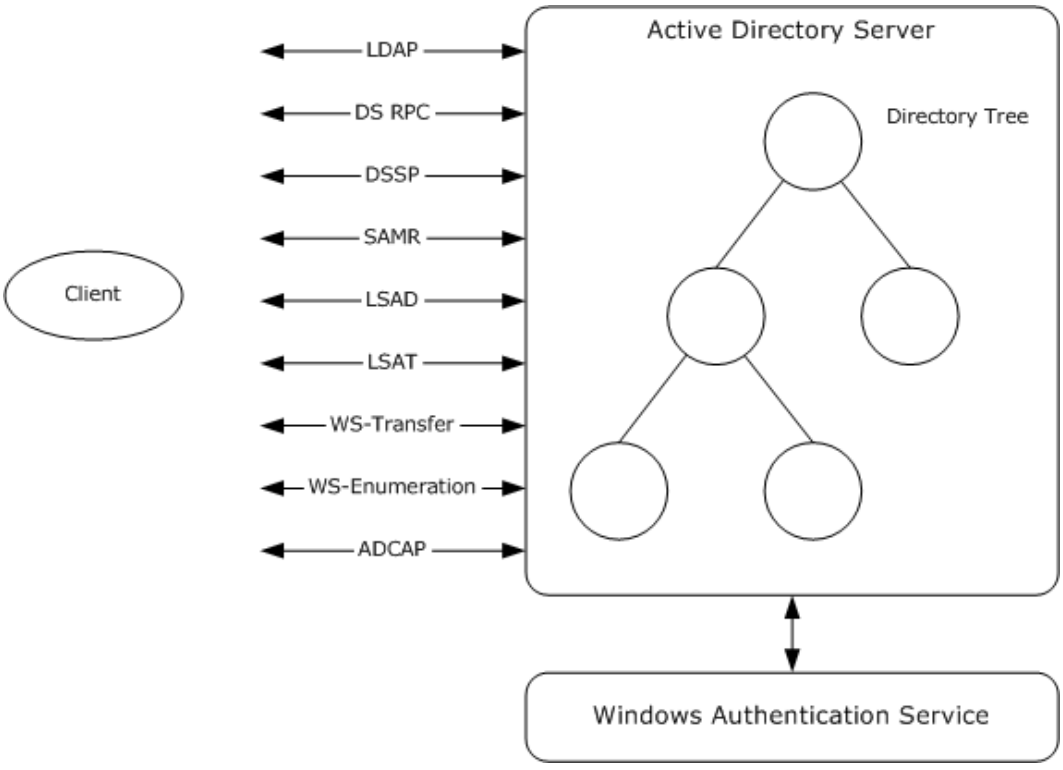

Figure 16: Active Directory server in context with client and Windows Authentication Service

The following figure presents the typical interactions between a client, a DNS server, the Windows Authentication Service, and the 1 or more (3 in this example) Active Directory server(s) constituting the Directory Service.

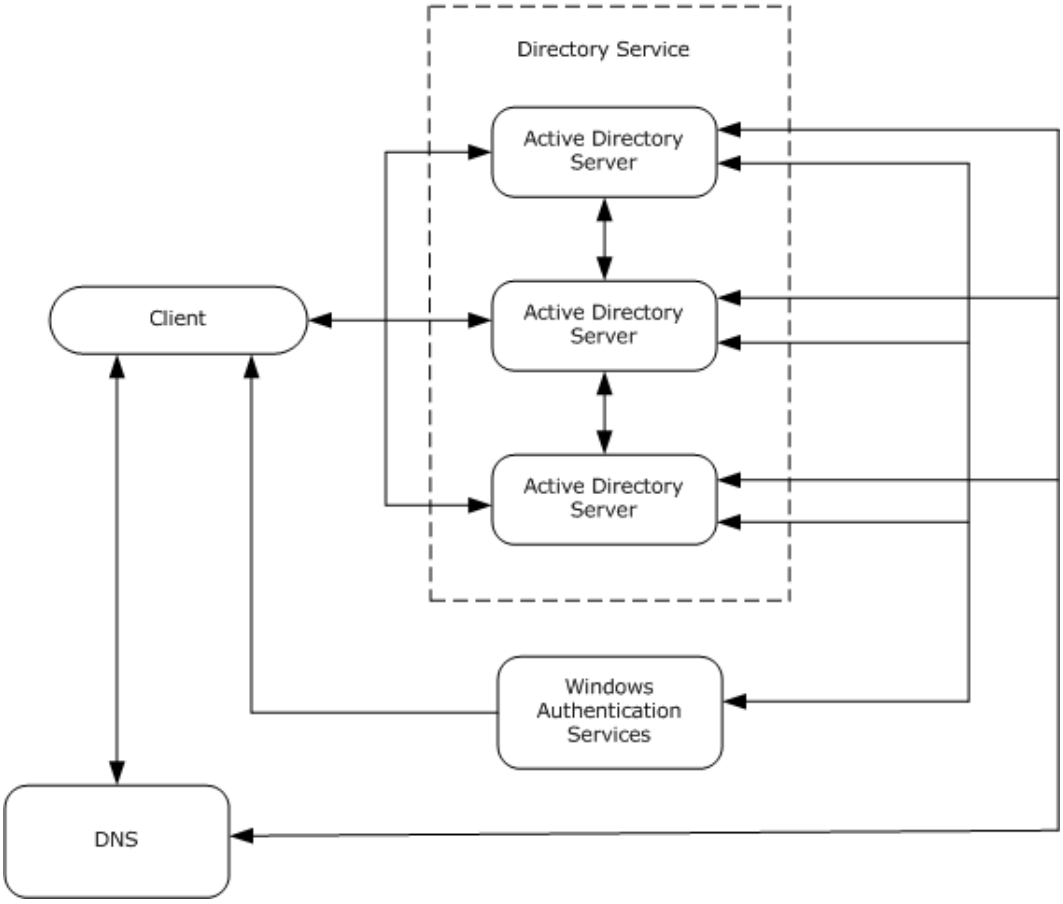

Figure 17: Interaction between the Client, the DNS service, the Windows Authentication Services, and the Directory Service, constituted of one or more directory servers

The actors in play in this model are one or several clients: a DNS Server, the Windows Authentication Service, and the Directory Service.

The directory servers forming the Directory Service implement the abstract data models and support the protocols relevant to the version and mode (AD DS or AD LDS) of the Active Directory System in use. See section 4.5 for a list. The directory service is provided by one or multiple Active Directory servers that are part of a distributed system that performs replication via an implementation-specific server to server replication protocol. Replication of directory data between the domain controllers offers a number of benefits:

- Adding new replicas enables handling the load from an increasing number of clients, making the directory scalable.
- Updates to the directory resulting from management operations performed on one domain controller replicate to other domain controllers; thus, the directory service is centrally manageable.

- Adding more replicas also increases reliability and availability of the system; in case one domain controller fails, clients can fail over to another replica, avoiding outages, and having more replicas available for responding to requests increases the availability of the overall service.

Other systems such as Group Policy are integrated with the Active Directory System by using it as a highly available and scalable store. Systems which integrate with the Active Directory System are listed in section 4.4.

Clients send requests to the directory servers using one of the protocols supported by the Active Directory servers, such as the industry-standard **LDAP** protocol (see [MS-ADTS] section 3.1.1.3, LDAP). A client can be located on the same computer as the directory server or on a different computer. Though Active Directory servers do replicate data, some clients may on occasion have specific requirements as to which directory server they need to communicate with (as an example, clients may want to target the directory server that hosts the Schema **FSMO role** in order to perform a schema update; see section 3.3.4.3). They can locate directory servers meeting their requirements as specified in [MS-ADTS] section 6.3.6, Locating a Domain Controller, and [MS-DISO] section 5, Locating a Domain Controller. Locating a directory server may in particular involve communication with a DNS server. On the other side, domain controllers also interact with DNS servers to register their records, as indicated in [MS-ADTS] section 6.3.2, DNS Record Registrations.

The Active Directory System requires the client to authenticate to the directory service (unless the client is performing unauthenticated, that is, anonymous, access) and the directory service to validate the client's authentication (except when the client is performing unauthenticated access). The system uses the Windows Authentication Services [MS-AUTHSO] to authenticate the clients to the directory server. The Windows Authentication Services supplies one or more means of authentication that is supported by both the client and the directory service. It must provide a mechanism for the client to obtain authentication material that can be used in the protocol by which the client intends to communicate with the directory service. It must also provide a mechanism by which the directory service, upon receipt of that authentication material, can validate that material to confirm that the client really is who it claims to be. The authenticated identity of the client is then used by the directory service to make authorization decisions for the purpose of access control; for example, deciding whether a given client has permission to read or write a particular attribute of a particular directory object, or to otherwise perform an operation exposed by a protocol (see section 7.1 for more information on access check methodology in the Active Directory System).

As shown in the following figure, the client sends a request to the directory service using one of the protocols supported by that service. The service processes the request in accord with the processing rules specified in the protocol document for the protocol being used and in the other supporting documents such as [MS-ADTS] and its appendixes ([MS-ADSC], [MS-ADLS], [MS-ADA1], [MS-ADA2], and [MS-ADA3]). All the protocols operate against the same directory tree. As a result of this, some of the protocols share a subset of the directory tree state. In this manner, the abstract data models of the individual protocols overlap, such that a change made to the directory tree via a request sent by one protocol is visible through all other protocols whose state models include the portion of the directory tree that was changed. This relationship between the state models of the different protocols is specified in section 5.1.

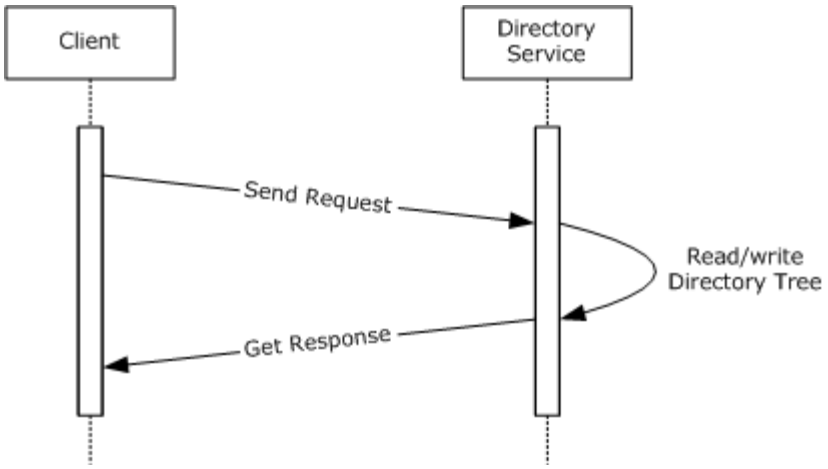

Figure 18: Generalized request/response operation sequence

4.3.2 System Dependencies

This section lists the systems that depend on the interfaces provided by the Active Directory System, as well as other systems that depend on this system. Other systems take dependencies on this system, and in particular on AD DS, not only by making use of this system's external interfaces, but also by sharing state with this system; that is, there are overlapping abstract data models. The specification of state shared between different protocols' abstract data models and how that state maps from one abstract data model to another is specified in section 5.1.

The Active Directory System depends on an environment that meets the criteria enumerated in section 4.1 for durable storage, network support, and an RPC runtime.

As illustrated in the black box diagrams in the preceding section, the Active Directory System also depends on the Windows Authentication Services [MS-AUTHSO] to authenticate clients that are accessing the system. The system controls access based on the identity of the client.

4.3.2.1 External Interfaces from this System

The following systems have dependencies on the protocol interfaces provided by the Active Directory System. All of these systems apply to Active Directory operating as AD DS. Note that while these systems depend on the Active Directory System, the Active Directory System does not, in turn, depend on them. In other words, any of these systems can be omitted from the environment, and the Active Directory System will continue to function. The exception is domain integration as documented in [MS-DISO], which is vital to the functioning of the Active Directory System in the AD DS mode of operation.

Domain Integration: Clients joined to a domain make use of the services of the Active Directory System as described in [MS-DISO].

As described in [MS-DISO], many systems rely on domain integration, and thus in turn the Active Directory Service, for authentication and authorization. Additionally, several of the systems leverage the directory service for additional purposes. These services include the following:

Certification Authority System: The Certification Authority System, described in [MS-CASO], MUST have access to the directory service to store and retrieve certificate templates when running in "Enterprise CA" mode.

Group Policy System: The Group Policy System, as described in [MS-GPSO], stores group policies in the directory service. Group policy clients discover and retrieve the policies to use from the directory service.

Network Policy and Access Services System: The Network Policy and Access Services System, described in [MS-NAPSO], retrieves information about users and computers from the directory service and uses that information to make access-control decisions.

4.3.2.2 Shared State from This System

As noted earlier, when operating as AD DS, the Active Directory System directly (without use of network protocols) shares some of its internal state represented in its abstract data model (documented in [MS-ADTS]) with other systems. These dependent systems make use of the Active Directory state in their own protocols, but unlike the systems described in the previous section, they do not obtain access to this state via any of the Active Directory System's protocols. This state-sharing happens entirely within the bounds of the server computer running the Active Directory System. Interoperability requirements do not put any constraints on how an implementation of one of these systems obtains its state from an implementation of the Active Directory System, except as documented in [MS-DISO] section 4.3.2, Domain Join State.

Windows Authentication Services (including the Kerberos **Key Distribution Center**): The Authentication Services use the directory service provided by AD DS to store user name and credential information, such as the information used by the Kerberos Key Distribution Center to service Kerberos ticket requests. The interaction between AD DS and the security mechanisms provided in a domain environment is described in [MS-DISO].

4.3.3 System Influences

The presence of the Active Directory System influences the behavior of the following systems by enabling optional functionality in them to be active. Unlike the systems in section 4.3.2, the systems in this section can continue to function (with reduced functionality) in the absence of the Active Directory System.

Message Queuing System: The Message Queuing System, as described in [MS-MQSO], optionally makes use of the directory service to store information such as queue and system metadata.

Print Services System: The Print Services System, documented in [MS-PSSO], can optionally publish information about shared printers in the directory. Domain-joined clients discover shared printers that are available to them by querying the directory for this information.

Additionally, because the Active Directory System supports the industry-standard LDAP protocol, a wide range of applications are capable of using the system to store and retrieve directory information. For example, Microsoft Exchange depends on the Active Directory System to store information such as a user's email addresses and the mailing lists to which he or she subscribes. Individual deployments can also contain applications written by the customer or by third-party software vendors that make use of Active Directory for storing and retrieving information. These systems can also make use of the Active Directory System operating as AD LDS, which is intended to be a "pure" LDAP directory server that does not have the domain integration functionality described in [MS-DISO] and has support for fewer protocols (see section 4.5).

4.4 System Applicability

The Active Directory System provides a centralized directory service with the ability to integrate into the Windows domain security model. It should be used for the following purposes:

- For storage of data that is a good fit to the data model used by LDAP and the Active Directory System, namely, hierarchically organized objects that consist of a collection of attributes.
- For storage of relatively static data that is expected to be read at a significantly higher rate than it is updated.
- For use in scenarios where the domain integration capabilities of [MS-DISO] are required. When deploying the Active Directory System to provide these capabilities, the AD DS mode of operation MUST be used.
- For use in scenarios where other systems that have a dependency on the Active Directory System, such as Group Policy [MS-GPSO] or Message Queuing [MS-MQSO], are to be deployed. When deploying the Active Directory System in support of these other systems, care should be taken to choose the appropriate mode of operation (typically, AD DS) for the Active Directory System.
- As a directory service for use by applications, such as web portals that need to store information about their registered users. In scenarios where the domain integration capabilities are not required, the AD LDS mode of operation can be a particularly good choice for this as it does not require support for protocols such as SAMR, LSAD, or LSAT that will not be used by the client application in this scenario.

The Active Directory System should not be used for the following purposes:

- As a replacement for a file system. Directory services such as the Active Directory System are not intended for storing highly volatile data, and emphasize read performance over write performance. They are also not designed for storing large amounts of unstructured data, such as storing a multi-megabyte value in a single attribute of a directory object.
- As a means of passing transient messages between clients. The Active Directory System is not intended to be a message-passing system. Applications that require such a system are encouraged to investigate the use of a system designed for that purpose, such as [MS-MQSO].

There is no interoperability requirement that an implementation of the Active Directory System support both the AD DS and AD LDS modes of operation. An implementer is free to implement either or both modes of operation, depending on his or her requirements for and intended use of the Active Directory System.

4.5 System Versioning and Capability Negotiation

There are two distinct modes of operation of the Active Directory System: Active Directory Domain Services (AD DS) and Active Directory Lightweight Directory Services (AD LDS). Additionally, some versions of AD DS and AD LDS include support for Web Services protocols. A summary of the different modes along with the protocols (or protocol subsets) and directory schemas supported by each is given in the table below. Information on which versions of AD DS and AD LDS support Web Services protocols is given in the following product behavior note:<3> The Technical Documents for the individual protocols specify additional versioning information; that is, not all versions of the Active Directory System support every method of a protocol listed below.

Modes and Protocols Supported

Mode	Protocols supported	Additional protocols of which a subset is supported	Schemas implemented
AD DS (without	DSSP LDAP	DS RPC: All methods of the dsaop interface are supported. All methods of the drsuapi interface are	[MS-ADA1] [MS-ADA2]

Mode	Protocols supported	Additional protocols of which a subset is supported	Schemas implemented
Web Services)	LSAD LSAT SAMR	supported except for the following: ▪ IDL_DRSInitDemotion ▪ IDL_DRSFinishDemotion	[MS-ADA3] [MS-ADSC]
AD DS (with Web Services)	ADCAP DSSP LDAP LSAD LSAT SAMR WS-Enumeration WS-Transfer *Protocol Extensions* IMDA WSDS WSPELD	DS RPC: All methods of the dsaop interface are supported. All methods of the drsuapi interface are supported except for the following: ▪ IDL_DRSInitDemotion ▪ IDL_DRSFinishDemotion	[MS-ADA1] [MS-ADA2] [MS-ADA3] [MS-ADSC]
AD LDS (without Web Services)	LDAP	DS RPC: All methods of the drsuapi interface are supported except for the following: ▪ IDL_DRSAddSidHistory ▪ IDL_DRSDomainControllerInfo ▪ IDL_DRSRemoveDsDomain ▪ IDL_DRSGetNT4ChangeLog ▪ IDL_DRSGetMemberships ▪ IDL_DRSInterDomainMove ▪ IDL_DRSGetMemberships2 ▪ IDL_DRSQuerySitesByCost ▪ IDL_DRSWriteSpn No methods of the dsaop interface are supported. DSSP: Supported in the same manner as any member server or stand-alone server on which the Active Directory System is not running.	[MS-ADLS]
AD LDS (with Web Services)	LDAP WS-Enumeration WS-Transfer *Protocol*	ADCAP: All methods of the AccountManagement port type are supported. The following methods of the TopologyManagement port type are supported: ▪ MoveADOperationMasterRole	[MS-ADLS]

Mode	Protocols supported	Additional protocols of which a subset is supported	Schemas implemented
	Extensions IMDA WSDS WSPELD	▪ ChangeOptionalFeature DS RPC: All methods of the drsuapi interface are supported except for the following: ▪ IDL_DRSAddSidHistory ▪ IDL_DRSDomainControllerInfo ▪ IDL_DRSRemoveDsDomain ▪ IDL_DRSGetNT4ChangeLog ▪ IDL_DRSGetMemberships ▪ IDL_DRSInterDomainMove ▪ IDL_DRSGetMemberships2 ▪ IDL_DRSQuerySitesByCost ▪ IDL_DRSWriteSpn No methods of the dsaop interface are supported. DSSP: Supported in the same manner as any member server or stand-alone server on which the Active Directory System is not running.	

The state model, constraints, processing rules, and so on, in [MS-ADTS] apply to both AD DS and AD LDS, except as otherwise noted in [MS-ADTS]. [MS-ADDM] applies to the Web Service-enabled versions of both AD DS and AD LDS.

4.6 System Vendor-Extensible Fields

This system has no vendor-extensible fields other than any that are specified in the protocol documents.

5 System Architecture

This section describes the basic structure of the system and the interrelationships among its parts, consumers, and dependencies.

5.1 Abstract Data Model

This section discusses the abstract data model of the Active Directory System. The system does not have a data model of its own. Rather, the system's data model is the union of the data models of the protocols that make up the system and the data models described in [MS-ADTS] and its appendixes ([MS-ADSC], [MS-ADLS], [MS-ADA1], [MS-ADA2], and [MS-ADA3]).

Portions of these abstract data models share state; that is, implementations of the protocols MUST share state such that a change in the abstract data model of one protocol is immediately visible to another protocol. An example of such overlapping state is that if a user account object is created via the SAMR protocol it is accessible via the LDAP protocol, and vice versa.

This sharing of state happens internal to the server implementation. Compatible implementations can use any mechanism of their choice to perform the sharing, as long as it satisfies the requirements of this section and the protocols' Technical Documents. For example, an implementation could use a single store that is shared across all the protocols, or it could use multiple stores (one per protocol) and build a synchronization mechanism to maintain consistency across all the stores, as long as that synchronization mechanism ensures that all atomicity requirements are observed and that two protocols that share state always see an identical view of that shared state.

It is the purpose of this section to document such shared state between the abstract data models of the Active Directory System protocols. For each protocol, the state that that protocol's abstract data model shares with other protocols' data models is specified.

In some cases, there is a unidirectional sharing of state: a data field in one protocol's abstract data model is influenced by the state in another protocol's abstract data model but cannot, in turn, influence the second protocol's state. Such data fields are typically read-only data fields in the first protocol's state. In other cases, there is a bidirectional sharing of state. Account modification via SAMR and LDAP is an example of such bidirectional sharing: any modification of the account via either one of the protocols is reflected in the other protocol's state.

Finally, the Web services protocols operate on an XML data representation defined in [MS-ADDM], which is mapped onto the abstract data model of the Active Directory System. Section 5.1.3 describes the utilization of that data model by the Web services protocols.

5.1.1 Coherency Requirements

Coherency requirements for shared state between protocols are specified in the sections below for each state mapping between protocols. As a general requirement, when multiple protocols share state and one of the protocols performs an operation on that shared state that is specified as atomic in that protocol's Technical Documents, other protocols sharing that state MUST honor the atomicity of that transaction. They MUST NOT attempt to perform operations on the uncommitted transaction or expose the uncommitted transaction to the client's view. For example, because SAMR and LDAP share state, the SAMR protocol methods to retrieve information about users can be used to view users created by LDAP as well as by SAMR. Since the creation of a directory object via LDAP is specified as an atomic transaction ([MS-ADTS] section 3.1.1.5.1.4, Transactional Semantics), a user that is in the process of being created by LDAP MUST NOT be visible via SAMR until the atomic transaction is committed.

The system MUST expose the same view of the shared state via all the protocols that share that state. This means that all committed changes to the shared state are visible (subject to access-control restrictions) through all protocols that share that state, and no uncommitted changes are visible through any of the protocols that share that state. This does not require that all state be visible through all protocols, but rather that if a piece of state is specified as being visible through two or more protocols, then that state MUST be the same when viewed through any of the protocols (subject to access-control restrictions).

5.1.2 Mapping Between Active Directory System Protocols' Abstract Data Models

For each protocol in the Active Directory System other than the Web Services protocols (which are discussed instead in section 5.1.3), the subsections of this section specify the state in that protocol's abstract data model that is shared with the data models of other protocols in the Active Directory System. For each data field of shared state in the protocol, it specifies the mapping of that data field to the data field(s) of the other protocol's data model with which state is shared. Where possible, this will be done by mapping the shared state in that protocol's abstract data model to state in the directory tree that forms the basis of the abstract data model of [MS-ADTS]. These mappings only apply to the protocols supported in the mode in which Active Directory is operating (as specified in section 4.5).

Sharing of state is transitive. If one abstract data model has a field that shares state with a field in a second abstract data model, and a third abstract data model also shares state with that field in the second model, then transitively the first and third abstract data models share state.

5.1.2.1 SAMR Protocol

With the exception of the abstract data fields specified in [MS-SAMR] section 3.1.1.5, Password Settings Attributes for Originating Update Constraints, the SAMR protocol does not define its own abstract data model. Instead, its protocol specification directly makes use of the abstract data model defined in [MS-ADTS]. It does, however, define its own terminology that it uses to refer to directory objects. For example, a SAMR "alias" object is a directory object of object class group with a particular value for the groupType attribute. Additionally, not all directory objects are accessible via the SAMR protocol, and for those objects that are accessible, not all attributes of the object are accessible via SAMR. [MS-SAMR] section 3.1.1.4, Object Class List, specifies the object classes of directory objects that the SAMR protocol uses, and [MS-SAMR] section 3.1.1, Abstract Data Model, specifies how these object classes map to the SAMR object terminology. [MS-SAMR] section 3.1.1.3, Attribute Listing, lists those attributes that are used by the SAMR protocol. The SAMR protocol ignores the hierarchical structure of the directory tree and treats these directory objects as if they were contained in a single flat list.

For the abstract data fields specified in [MS-SAMR] section 3.1.1.5, Password Settings Attributes for Originating Update Constraints, that section also specifies how those data fields map to the abstract data model (the directory tree) specified in [MS-ADTS]. The sharing of state is unidirectional from the abstract data model of [MS-ADTS] to these fields, since SAMR treats them as read-only. They are read atomically from the directory object: since creation and modification of directory objects is atomic ([MS-ADTS] section 3.1.1.5.1.4, Transactional Semantics), the state of these data fields in SAMR will never reflect a partially created or partially updated directory object.

5.1.2.2 LSAD Protocol

For each portion of the LSAD abstract data model (given in [MS-LSAD] section 3.1.1, Abstract Data Model), the data field shown below shares state with other protocols' abstract data models. This sharing of state is as specified in this section.

Note that despite their name, LSAD account objects ([MS-LSAD] section 3.1.1.3, Account Object Data Model) do not correspond to accounts in the Active Directory System. LSAD account objects are not represented in the directory tree and do not share state with any other protocols in the Active Directory System, and therefore are not further discussed in this section.

Policy Object Data Model ([MS-LSAD] section 3.1.1.1, Policy Object Data Model)

LSAD abstract data model name	LSAD structure and field	Shares state with abstract data model field	Abstract data model field defined in
Server Role Information	(POLICY_LSA_SERVER_ROLE_INFO.LsaServerRole)	DomainServerRoleInformation (DOMAIN_SERVER_ROLE_INFORMATION.DomainServerRole)	[MS-SAMR] section 3.1.5.5.1.2, DomainServerRoleInformation

The Server Role Information state is defined in terms of the output of the SamrQueryInformationDomain2 method of the SAMR protocol. This means that the contents of the Server Role Information abstract data fields, at any point in time, MUST equal what would be returned by a SamrQueryInformationDomain2 method call made at that point in time with the DomainInformationClass parameter set equal to DomainServerRoleInformation.

The sharing of state information is bidirectional. A change to the Server Role Information is treated as equivalent to a call to the SamrSetInformationDomain method with the DomainInformationClass parameter set equal to DomainServerRoleInformation. Any changes of the Server Role Information state MUST conform to the constraints specified in [MS-SAMR] section 3.1.5.6.1.1, DomainServerRoleInformation.

Secret Object Data Model ([MS-LSAD] section 3.1.1.4, Secret Object Data Model)

Secret objects in the LSAD abstract data model can be classified into several different types; see the behavior notes for [MS-LSAD] section 3.1.1.4, Secret Object Data Model). Those which are classified as global secret objects correspond to directory objects of the following class in the directory tree:

- secret

The Name fields of such secret objects in the LSAD abstract data model always start with "G$".

Global secret objects are specific to and are contained in individual domains. In the directory tree, the directory objects corresponding to LSAD global secret objects are located in the System container in the domain naming context ([MS-ADTS] section 6.1.1.4.11, System Container).

LSAD abstract data model field	Shares state with abstract data model field	Abstract data model field defined in
Name	cn attribute of the corresponding directory object, subject to the following mapping: Let the Name field of the LSAD secret object be "G$name". Then, the value of the cn attribute will be "nameSecret".	[MS-ADTS]
Security Descriptor	None -- access to this abstract data field is rejected in LSAD message processing for LsarQuerySecurityObject and LsarSetSecurityObject	

LSAD abstract data model field	Shares state with abstract data model field	Abstract data model field defined in
Old Set Time	priorSetTime attribute of the corresponding directory object	[MS-ADTS]
Old Value	priorValue attribute of the corresponding directory object	[MS-ADTS]
New Set Time	lastSetTime attribute of the corresponding directory object	[MS-ADTS]
New Value	currentValue attribute of the corresponding directory object	[MS-ADTS]

The sharing of state is bidirectional: updates to the global secret object in the LSAD abstract data model are immediately reflected in the corresponding directory object, and vice versa. The sharing of state is also atomic: there is no point in time at which a change in the secret object state will only be partially reflected in the directory object state, and vice versa.

Trusted Domain Object Data Model ([MS-LSAD] section 3.1.1.5, Trusted Domain Object Data Model)

Trusted domain objects in the LSAD abstract data model correspond to objects of the following class in the directory tree:

- trustedDomain

There is a one-to-one mapping between LSAD trusted domain objects and directory objects of the above class. The LSAD abstract data model ignores the hierarchical structure of the directory tree and treats these directory objects as if they were contained in a single flat list.

For information on the mapping between the state of trusted domain objects in the LSAD abstract data model and trustedDomain objects in the directory tree, see [MS-LSAD] section 3.1.1.5, Trusted Domain Object Data Model, where each abstract data field is linked to the appropriate subsection of [MS-ADTS] section 6.1.6.7, Essential Attributes of a Trusted Domain Object.

As with the secret objects, the sharing of state is bidirectional and atomic.

5.1.2.3 LSAT Protocol

For each portion of the LSAT abstract data model (given in [MS-LSAT] section 3.1.1, Abstract Data Model), the data fields shown below share state with other protocols' abstract data models. This sharing of state is as specified in this section.

Configurable Translation Database and Corresponding View ([MS-LSAT] section 3.1.1.1.2, Configurable Translation Database and Corresponding View)

The state in this view is shared with the abstract data model of the Service Control Manager Remote Protocol ([MS-SCMR] section 3.1.1, Abstract Data Model). The state mapping is given in [MS-LSAT] section 3.1.1.1.2, Configurable Translation Database and Corresponding View. The sharing of information is unidirectional from the Service Control Manager Remote Protocol abstract data model to LSAT's abstract data model, since LSAT treats this information as read-only. The sharing of state is atomic at the level of the **service record** in the [MS-SCMR] abstract data model: LSAT will never see a partially complete or partially updated service record.

Builtin Domain Principal View ([MS-LSAT] section 3.1.1.1.3, Builtin Domain Principal View)

Rows in this view in the LSAT abstract data model correspond to objects of the following class in the directory tree:

- group

Directory objects of classes that derive from this class are excluded. Additionally, only directory objects that correspond to the criteria given in [MS-LSAT] section 3.1.1.1.3, Builtin Domain Principal View, are used. There is a one-to-one mapping between rows in this LSAT view and directory objects of the above class that satisfy the criteria. The LSAT view ignores the hierarchical structure of the directory tree and treats these directory objects as if they were contained in a single flat list.

LSAT abstract data model field	Shares state with abstract data model field	Abstract data model field defined in
Security Principal SID	objectSID attribute of the corresponding directory object	[MS-ADTS]
Security Principal Type	sAMAccountType attribute of the corresponding directory object, using the mapping table from [MS-LSAT] section 3.1.1.1.3, Builtin Domain Principal View	[MS-ADTS]
Security Principal Name	sAMAccountName attribute of the corresponding directory object	[MS-ADTS]

The sharing of information is unidirectional from the abstract data model of [MS-ADTS] to LSAT, since LSAT treats this information as read-only. The sharing of state is atomic at the level of the directory object: LSAT will never see a partially complete or partially updated directory object.

Account Domain Principal View ([MS-LSAT] section 3.1.1.1.4, Account Domain Principal View)

Rows in this view in the LSAT abstract data model correspond to objects of the following classes in the directory tree:

- user
- group
- computer
- inetOrgPerson
- msDS-ManagedServiceAccount

Only directory objects that correspond to the criteria given in [MS-LSAT] section 3.1.1.1.4, Account Domain Principal View, are used. There is a one-to-one mapping between rows in this LSAT view and directory objects of the above class that satisfy the criteria. The LSAT view ignores the hierarchical structure of the directory tree and treats these directory objects as if they were contained in a single flat list.

LSAT abstract data model field	Shares state with abstract data model field	Abstract data model field defined in
Security Principal SID	objectSID attribute of the corresponding directory object	[MS-ADTS]
Security Principal Name	sAMAccountName attribute of the corresponding directory object	[MS-ADTS]

LSAT abstract data model field	Shares state with abstract data model field	Abstract data model field defined in
Security Principal Type	sAMAccountType attribute of the corresponding directory object, using the mapping table from [MS-LSAT] section 3.1.1.1.3, Builtin Domain Principal View	[MS-ADTS]
Default User Principal Names	sAMAccountName of the corresponding directory object, using the algorithm given in [MS-LSAT] section 3.1.1.1.4, Account Domain Principal View	[MS-ADTS]

The sharing of information is unidirectional from the abstract data model of [MS-ADTS] to LSAT, since LSAT treats this information as read-only. The sharing of state is atomic at the level of the directory object: LSAT will never see a partially complete or partially updated directory object.

Forest Principal View ([MS-LSAT] section 3.1.1.1.7, Forest Principal View)

Rows in this view in the LSAT abstract data model correspond to objects of the following classes in the directory tree:

- user
- group
- computer
- inetOrgPerson
- msDS-ManagedServiceAccount

Those directory objects that correspond to the criteria given in [MS-LSAT] section 3.1.1.1.7, Forest Principal View, are used. Directory objects from all domains in the forest are used. There is a one-to-one mapping between rows in this LSAT view and directory objects of the above class that satisfy the criteria. The LSAT view ignores the hierarchical structure of the directory tree and treats these directory objects as if they were contained in a single flat list.

LSAT abstract data model field	Shares state with abstract data model field	Abstract data model field defined in
User Principal Name	userPrincipalName attribute of the corresponding directory object	[MS-ADTS]
Security Principal SID History	sidHistory attribute of the corresponding directory object	[MS-ADTS]
Security Principal SID	objectSID attribute of the corresponding directory object	[MS-ADTS]
Security Principal Name	sAMAccountName attribute of the corresponding directory object	[MS-ADTS]
Security Principal Type	sAMAccountType attribute of the corresponding directory object, using the mapping table from [MS-LSAT] section 3.1.1.1.3, Builtin Domain Principal View	[MS-ADTS]
Default **User**	sAMAccountName of the corresponding directory object, using	[MS-ADTS]

LSAT abstract data model field	Shares state with abstract data model field	Abstract data model field defined in
Principal Name	the algorithm given in [MS-LSAT] section 3.1.1.1.5, Account Domain Information View	

The sharing of information is unidirectional from the abstract data model of [MS-ADTS] to LSAT, since LSAT treats this information as read-only. The sharing of state is atomic at the level of the directory object: LSAT will never see a partially-complete or partially-updated directory object.

Domain Database Information ([MS-LSAT] section 3.1.1.2, Domain Database Information)

When the LSAT protocol is used to translate information for an account stored in a domain database which is an Active Directory database ([MS-LSAT] section 3.1.1, Abstract Data Model), the following mapping is used.

LSAT abstract data model field	Shares state with abstract data model field	Abstract data model field defined in
Domain DNS Name	DNS Domain Information (LSAPR_POLICY_DNS_DOMAIN_INFO.DnsDomainName)	[MS-LSAD] section 3.1.1.1, Policy Object Data Model
Domain **NetBIOS** Name	DNS Domain Information (LSAPR_POLICY_DNS_DOMAIN_INFO.Name)	[MS-LSAD] section 3.1.1.1, Policy Object Data Model
Domain SID	DNS Domain Information (LSAPR_POLICY_DNS_DOMAIN_INFO.Sid)	[MS-LSAD] section 3.1.1.1, Policy Object Data Model

When the LSAT protocol is used to translate information for an account stored in a domain database which is not an Active Directory database, the following mapping is used.

LSAT abstract data model field	Shares state with abstract data model field	Abstract data model field defined in
Domain DNS Name	None -- left empty	
Domain NetBIOS Name	Account Domain Information (LSAPR_POLICY_ACCOUNT_DOM_INFO.DomainName)	[MS-LSAD] section 3.1.1.1, Policy Object Data Model
Domain SID	Account Domain Information (LSAPR_POLICY_ACCOUNT_DOM_INFO.DomainSid)	[MS-LSAD] section 3.1.1.1, Policy Object Data Model

The sharing of information is unidirectional from LSAD to LSAT, since LSAT treats this information as read-only. The LSAT protocol accesses the three pieces of state listed in the tables above as a single atomic read.

Trusted Domains and Forest Information ([MS-LSAT] section 3.1.1.3, Trusted Domains and Forests Information)

The LSAT abstract data model uses information from the Trusted Domain Object Data Model portion of the LSAD abstract data model, given in [MS-LSAD] section 3.1.1.5, Trusted Domain Object Data Model. The LSAT abstract data model only incorporates those LSAD trusted **domain objects** that meet the criteria given in [MS-LSAT] section 3.1.1.3, Trusted Domains and Forests Information. For those trusted domain objects that do meet that criteria, the mapping between the data fields of the LSAT state and the LSAD trusted domain object data fields is as shown in the following table.

LSAT abstract data model field	Shares state with abstract data model field	Abstract data model field defined in
Trusted Domain DNS Name	Name	[MS-LSAD] section 3.1.1.5, Trusted Domain Object Data Model
Trusted Domain NetBIOS Name	Flat Name	[MS-LSAD] section 3.1.1.5, Trusted Domain Object Data Model
Domain SID	Security Identifier	[MS-LSAD] section 3.1.1.5, Trusted Domain Object Data Model
Trusted Direction	Trust Direction	[MS-LSAD] section 3.1.1.5, Trusted Domain Object Data Model
Trust Type	Trust Type	[MS-LSAD] section 3.1.1.5, Trusted Domain Object Data Model
Trust Attributes	Trust Attributes	[MS-LSAD] section 3.1.1.5, Trusted Domain Object Data Model
Forest Trust Information	Forest Trust Information	[MS-LSAD] section 3.1.1.5, Trusted Domain Object Data Model

The sharing of state information is unidirectional from LSAD to LSAT, since LSAT treats the trust information in its abstract data model as read-only. The sharing of state is atomic at the level of the LSAD trusted domain object: LSAT will never see a partially complete or partially updated trusted domain object.

5.1.2.4 DS RPC Protocol

The DS RPC protocol does not define its own abstract data model. Its protocol specification directly makes use of the abstract data model defined in [MS-ADTS], including utilizing the same typographic conventions to represent access to that abstract data model. Therefore, there is no further mapping between the DS RPC protocol and the model of [MS-ADTS] beyond what is defined in [MS-DRSR].

5.1.2.5 LDAP Protocol

The LDAP protocol operates directly on the directory tree-based abstract state model of [MS-ADTS] and does not have a separate abstract data model of its own. Therefore, there is no further mapping between the LDAP protocol and the model of [MS-ADTS].

5.1.3 Web Services Protocols

In the Active Directory System, the WS-Transfer and WS-Enumeration protocols operate on the XML data model specified in [MS-ADDM] section 2.3, XML Data Model. That document also specifies the mapping between that XML representation of the data and the directory objects and directory tree that are the basis of the abstract data model specified in [MS-ADTS].

The IMDA protocol extension [MS-WSTIM] also operates on this same XML representation. In the Active Directory System, the IMDA **identity object** is a directory object, and the **identity attributes** are the attributes of the directory object as well as the **synthetic attributes** described in [MS-ADDM] section 2.3.3, Synthetic Attributes. Each **identity attribute value** corresponds to exactly one value of the directory attribute (or synthetic attribute), represented using the syntax given in [MS-ADDM] section 2.3.4, Syntax Mapping, which is the same syntax used by the system for the WS-Enumeration and unextended WS-Transfer protocols. The dialect used for the **identity attribute types** is the XPath 1.0-derived Selection Language specified in [MS-ADDM] section 2.4, XPath 1.0-Derived Selection Language. This dialect is also used for the WSDS extensions to the WS-Enumeration protocol for the SelectionProperty and the SortingProperty [MS-WSDS].

The WSPELD protocol extension permits LDAP extended controls to be attached to WS-Transfer and WS-Enumeration operations, including WS-Transfer operations that use the IMDA extensions. The Active Directory System permits all the extended controls specified in [MS-ADTS] section 3.1.1.3.4.1, LDAP Extended Controls, to be used with WSPELD, except for the following:

- LDAP_SERVER_NOTIFICATION_OID
- LDAP_SERVER_SHUTDOWN_NOTIFY_OID

An attempt to use the above extended controls will be rejected with a **SOAP fault**.

The ADCAP protocol does not define its own abstract data model. Its protocol specification directly makes use of the directory tree-based abstract data model of [MS-ADTS]. Therefore, there is no further mapping between the ADCAP protocol and the abstract data model of [MS-ADTS].

5.2 White Box Relationships

As shown in the black box diagrams of section 4.3.1, an Active Directory server exposes a collection of protocols which access state shared between those protocols (where this sharing of state is as documented in section 5.1). The following diagram expands upon the interior of the Active Directory server to show the logical design of an Active Directory server. For simplicity, the dependencies on DNS and the Windows Authentication System are not repeated in this diagram.

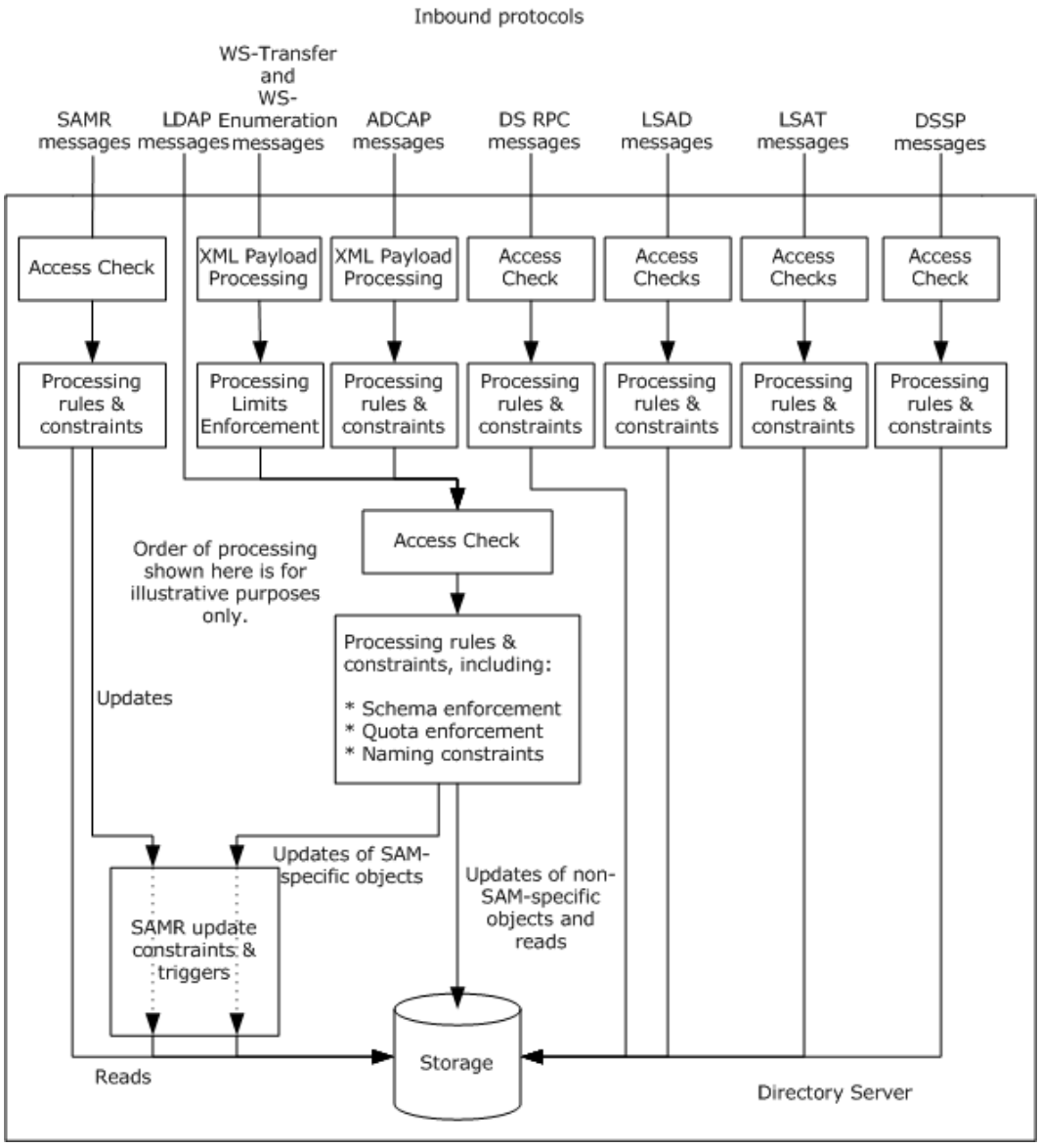

Figure 19: Logical Design of the Active Directory server

For each of the protocols, the processing for that protocol is described in the protocol's associated Technical Document. For example, for LSAD or LSAT, the access checks and processing rules and constraints are described in [MS-LSAD] and [MS-LSAT], respectively.

As shown in the diagram, several of the protocols have a complex interrelationship that goes beyond merely sharing state at the storage level. Updates to SAM-specific objects (defined in [MS-ADTS] section 3.1.1.5.2.3, Special Classes and Attributes) performed using the LDAP protocol are also subject to the rules of [MS-SAMR] sections 3.1.1.6, Attribute Constraints for Originating Updates, through 3.1.1.9, Additional Update Triggers. Thus, any implementation of the Active Directory system MUST guarantee a consistent enforcement of these rules regardless of whether the originating update occurs via the SAMR or LDAP protocol. Note that while the diagram depicts this SAMR update processing as occurring after the SAMR- and LDAP-specific processing, this is for

illustrative purposes only and any processing order is permissible provided that the final result of that processing is consistent with the processing rules and constraints of [MS-SAMR] and [MS-ADTS].

Also shown in the diagram is a link between the processing rules enforced by the WS-Transfer and WS-Enumeration protocols (and their protocol extensions, such as IMDA) and the processing rules and constraints defined in [MS-ADTS] and used by LDAP. Thus, when accessing the directory using the WS-Transfer and WS-Enumeration protocols, these protocols obey all of the processing rules, constraints and access checks of the LDAP protocol, with the XML data model used by WS-Transfer and WS-Enumeration mapped onto the LDAP data model defined in [MS-ADTS] using the mapping specified in [MS-ADDM] section 2.3, XML Data Model.

The protocols are depicted as sharing an internal storage subsystem. This internal storage subsystem stores the state maintained by each protocol (as defined in the abstract data model of the individual protocols). This does not mean that all the abstract state of one protocol is accessible to the other protocols. Rather, the sharing of abstract state between the protocols is as documented in section 5.1.

The exact storage subsystem used is an implementation detail of the server. However, it MUST maintain the persistent data and state of the protocols across system reboots. It MUST be capable of meeting any transactional requirements specified by the individual protocols.

5.3 Member Protocol Functional Relationships

This section provides information on how each member protocol is used by the system and on how the protocols can be logically grouped into distinct areas of functionality.

5.3.1 Member Protocol Roles

All of the member protocols in the Active Directory System, as described in this Protocol Family System Document, are used for communication between the client and the directory service. The purpose played by each protocol in the system is best understood in relationship to the protocol groupings described in the next section. Each protocol grouping specified in that section will explain the role of the protocols in that grouping.

The protocols can be divided into approximately three distinct roles:

- General-purpose directory tree access: These are the protocols used by the client for generic read and write access to directory objects in the directory tree.
- Limited directory tree access: Like the general-purpose directory access protocols, these protocols provide access to objects in the directory tree. However, the access provided is more limited, providing access to only a subset of the objects in the tree. These protocols were originally implemented in versions of Windows that predate the Active Directory System (for example, they were supported by Windows NT operating system 4.0 domain controllers). The Active Directory System continues to support these protocols to maintain backwards compatibility.
- Task-oriented directory tree access: These protocols operate against directory objects in the directory tree, but they are focused on performing a specific task using those objects rather than providing generic read and write access to the objects.
- Specialized access: These protocols operate on state that is not directly accessible via the general-purpose or limited directory tree access protocols. Such protocols, although they may share state with the abstract data models of protocols in the former roles, typically have their own separate abstract data model as well in which they maintain their own state.

The protocols organized by role are as shown in the following table. Also shown is the protocol group (section 5.3.2) of which each protocol is a part. A single protocol group can contain protocols that span multiple roles, since achieving the task of the group (for example, providing SOAP-based access to the directory service) can require fulfilling multiple roles.

General-Purpose Directory Tree Access

Protocol name	Role description	Protocol group	Reference
LDAP	Permits directory objects to be read, written, and queried using a block-structured protocol.	Core group	[RFC3377], [RFC1777]
WS-Transfer	Permits directory objects to be read and written using a SOAP-based protocol.	Web Services group	[WXFR]
WS-Enumeration	Permits directory objects to be queried using a SOAP-based protocol.	Web Services group	[WSENUM]

Limited Directory Tree Access

Protocol name	Role description	Protocol group	Reference
SAMR	Permits portions of the directory tree corresponding to accounts to be read and written.	SAM group	[MS-SAMR]

Task-Oriented Directory Tree Access

Protocol name	Role description	Protocol group	Reference
DS RPC	Allows tasks like name translation and group membership expansion to be performed.	Core group	[MS-DRSR]
ADCAP	Allows tasks for performing account maintenance and retrieving information about the topology of the Active Directory System.	Web Services group	[MS-ADCAP]
LSAT	Translates security identifier (SIDs) to human-readable names, and vice versa	LSA group	[MS-LSAT]

Specialized Access

Protocol name	Role description	Protocol group	Reference
DSSP	Retrieves information about the status of a computer in the Active Directory System.	Core group	[MS-DSSP]
LSAD	Retrieves and sets security policy information.	LSA group	[MS-LSAD]

5.3.2 Member Protocol Groups

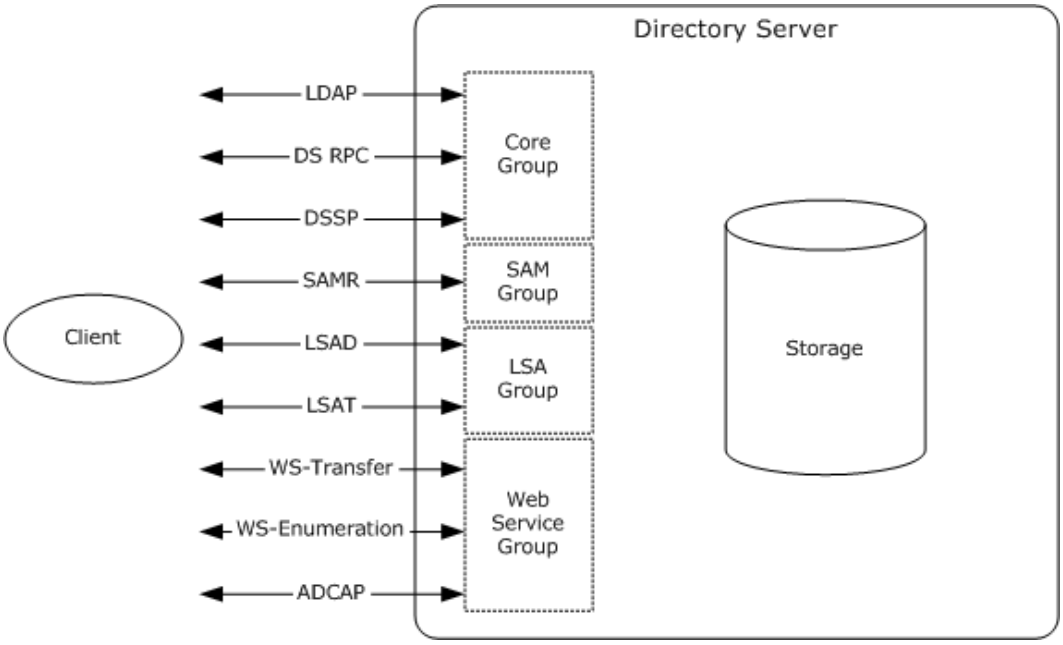

Figure 20: Active Directory protocol groupings

As illustrated in the preceding figure, the member protocols that make up the Active Directory System can be divided into four functional groups. Each group accomplishes an interrelated set of tasks, and protocols in the same group are typically used by a client in conjunction with other protocols in the same group. The groups are as follows:

- The core group contains protocols that are supported by all directory servers in the Active Directory System, whether running in AD DS or AD LDS mode. This group includes the LDAP protocol, which is the primary protocol used to read and write objects in the directory tree. It also includes the DS RPC protocol used for performing certain tasks (such as name translation) against objects in the directory tree. The DSSP protocol does not perform operations against the directory tree, but is included in this group because it is also present on all directory servers in the Active Directory System.
- The SAM group includes a single protocol, SAMR, which is used to perform account maintenance. It operates on the same directory tree as the core group of protocols, but only provides access to a subset of that tree (see section 5.1.2.1). It is only supported when operating in AD DS mode.
- The LSA group contains the LSAD and LSAT protocols. Both protocols are serviced off of the same RPC interface and endpoint (see [MS-LSAD] section 1.8, Vendor-Extensible Fields, and [MS-LSAT] section 1.8, Vendor-Extensible Fields) and are only supported when operating in AD DS mode.
- The Web Services group consists of the WS-Transfer, WS-Enumeration, and ADCAP protocols along with the IMDA, WSDS, and WSPELD protocol extensions. This protocol group is only supported on some versions of the Active Directory System (see section 4.5). Much like the core group, these protocols permit clients to read and write directory objects in the directory tree, as well as to perform selected tasks against the tree. Unlike the core group, the protocols in this group are based on SOAP rather than RPC or block-structured transports.

These groups are merely a conceptual assemblage of protocols used to better illustrate the operation of the Active Directory System in this document. They are not actual components of the Active Directory System and do not have state. Rather, all state is contained in the abstract data models defined by the individual protocols that comprise the groups. The protocols operate against the abstract data models as defined in the protocol documents and in section 5.1.

More information about each protocol group is in the subsections that follow.

5.3.2.1 Core Group

All directory services in the Active Directory System support the core group of protocols. This section describes the protocols that comprise the core group.

The Active Directory System uses LDAP to provide access to the directory tree to clients. LDAP version 3 [RFC3377] is supported, as is the older LDAP version 2 [RFC1777]. There are some features of the LDAP protocol that the Active Directory System does not support. Additionally, there are features of the LDAP protocol that the Active Directory System supports in a manner different than that described in the LDAP RFCs or where the system had to interpret ambiguous portions of the RFCs. These are described in [MS-ADTS] section 3.1.1.3.1, LDAP Conformance. LDAP is the primary communications protocol between clients and the directory servers, providing general-purpose access to create, update, read, and delete directory objects. It provides the ability to search for a directory object that matches a set of criteria supplied by the client. Other systems, such as the Group Policy System [MS-GPSO] and Message Queuing System [MS-MQSO] use LDAP for accessing the directory service.

The core group of protocols exposed to clients also encompasses DS RPC (defined in [MS-DRSR]), excluding methods of the DS RPC interfaces used exclusively for server-to-server communications. This protocol provides clients with the ability to perform tasks such as name translation and determining the group membership of an object.

The DSSP protocol (defined in [MS-DSSP]) SHOULD be supported by all client and server computers. All computers SHOULD listen for requests made via this protocol and respond as documented in [MS-DSSP]. When the computer on which it is running is a domain controller (that is, the system is operating as AD DS), this protocol permits a client to query the AD DS domain controller to determine if it is a **primary domain controller** or a **read-only domain controller**. For computers that are in the process of transitioning to become an AD DS domain controller ("promotion") or are currently AD DS domain controllers but transitioning to no longer be an AD DS domain controller ("demotion"), this protocol also permits the client to query the status of that transition.

When operating as AD LDS, the directory server supports the DSSP protocol in the same manner as any other computer by permitting the client to determine whether the computer on which the directory service is running is joined to a domain and, if so, what domain it is joined to.

The following table summarizes the core group of Active Directory System protocols.

Core Group of Protocols

Protocol name	Reference
LDAP (version 3)	[RFC3377]
LDAP (version 2)	[RFC1777]
DS RPC	[MS-DRSR]

Protocol name	Reference
DSSP	[MS-DSSP]
Active Directory Technical Specification (Although not a protocol itself, this influences and controls the behavior of the protocols in the Active Directory System.)	[MS-ADTS]

5.3.2.2 SAM Group

The SAM group consists of a single protocol, SAMR, as described in [MS-SAMR]. It is supported by the Active Directory System in AD DS mode.

This protocol supports communications between the client and an AD DS domain controller to perform account maintenance, including the creation, modification, retrieval, and deletion of users and groups. Since this protocol operates on the same directory tree as the core protocols (using a mapping between the abstract data models of [MS-ADTS] and [MS-SAMR] as described in section 5.1), changes made using this protocol are visible to clients using the core protocols, such as LDAP, and vice versa.

The following table summarizes the SAM group of Active Directory System protocols.

SAM Group of Protocols

Protocol name	Reference
SAMR	[MS-SAMR]

5.3.2.3 LSA Group

The LSA group of protocols is supported when the Active Directory System is operating in AD DS mode.

The Active Directory System provides support for the following protocols that together form the LSA protocol group:

- LSAD, as described in [MS-LSAD], allows clients to retrieve security policy information.
- LSAT, as described in [MS-LSAT], permits clients to translate the security identifiers (SIDs) that identity security principals, such as users, to human-readable names, and vice versa. This is useful for scenarios such as human management of resource access, by translating the SIDs that are used in access control entries to names understandable by administrators.

The following table summarizes the LSA group of Active Directory System protocols.

LSA Group of Protocols

Protocol name	Reference
LSAD	[MS-LSAD]
LSAT	[MS-LSAT]

5.3.2.4 Web Services Group

The Web Services group of protocols is supported on some versions of the Active Directory System, as described in section 4.5.

The Web Services protocols (except for ADCAP) operate on the XML data model described in [MS-ADDM]. That document also describes the mapping between that XML data model and the directory objects that form the basis of the data model used in this Protocol Family System Document and in [MS-ADTS]. The following Web Services protocols and protocol extensions are supported by the Active Directory System:

- WS-Transfer, as described in [WXFR], provides the ability to read (via a Get operation) or delete (via a Delete operation) an entire directory object. The Put and Create operations are only supported when used in conjunction with the IMDA protocol extensions.
- IMDA, as described in [MS-WSTIM], is a set of extensions to WS-Transfer. It extends the Get and Put operations of WS-Transfer to permit directory objects (represented as XML data) to be read from and written to at the level of individual attributes, instead of requiring the client to always read and write the entire directory object. This protocol extension also contains extensions to the WS-Transfer Create operation permitting the client to specify only selected attributes of the directory object to be created, with the directory service filling in remaining attributes as necessary. The IMDA protocol is defined in terms of identity objects rather than directory objects. However, as used in the Active Directory System, an IMDA identity object is a directory object. This mapping between identity objects and directory objects is given in section 5.1.3.
- WS-Enumeration, as described in [WSENUM], is used to retrieve directory objects (represented as XML data) that match a client-specified criteria.
- WSDS, as described in [MS-WSDS], is a set of extensions to the WS-Enumeration protocol that describes, among other things, the query language used by the clients to specify the criteria of which directory objects should be retrieved.
- WSPELD, as described in [MS-WSPELD], is a protocol extension that permits the use of LDAP version 3 extended controls with WS-Transfer and WS-Enumeration operations.
- ADCAP, as described in [MS-ADCAP], is a protocol for managing Active Directory account information and topologies. Unlike the other Web Services protocols, it does not make use of the XML data model defined in [MS-ADDM] but rather uses its own XML representations defined in [MS-ADCAP].

The following table summarizes the Web Services group of Active Directory System protocols and protocol extensions.

Web Services Group of Protocols

Protocol (or protocol extension) name	Reference
WS-Transfer	[WXFR]
IMDA	[MS-WSTIM]
WS-Enumeration	[WSENUM]
WSDS	[MS-WSDS]
WSPELD	[MS-

Protocol (or protocol extension) name	Reference
	WSPELD]
ADCAP	[MS-ADCAP]
Active Directory Web Services: Data Model and Common Elements (Although not a protocol itself, this defines an XML data model shared by the other Web Service protocols and protocol extensions, as well as common protocol elements referenced by the other documents.)	[MS-ADDM]

Note that despite the name (WS-Transfer: Lightweight Directory Access Protocol (LDAP) v3 Control Extension), the protocol extension described in [MS-WSPELD] applies to both WS-Transfer and WS-Enumeration.

5.4 System Internal Architecture

The Active Directory system is comprised of a single logical component that implements interfaces for multiple protocols. The specific set of interfaces implemented depends on whether Active Directory is operating in AD DS mode or AD LDS mode, and (in the case of AD LDS) whether the directory service is hosted on a server joined to a domain. The following diagrams summarize the interfaces and associated communication paths for the logical component of the Active Directory system.

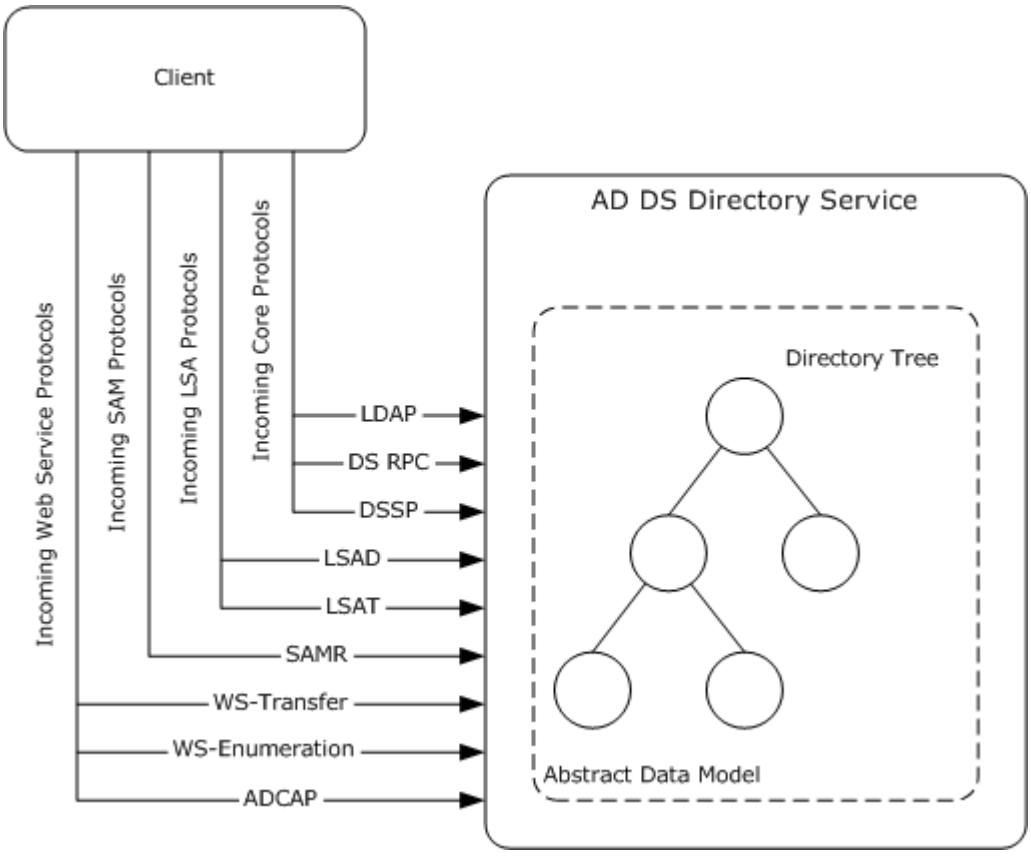

Figure 21: Incoming Interfaces supported by the Active Directory System in AD DS mode

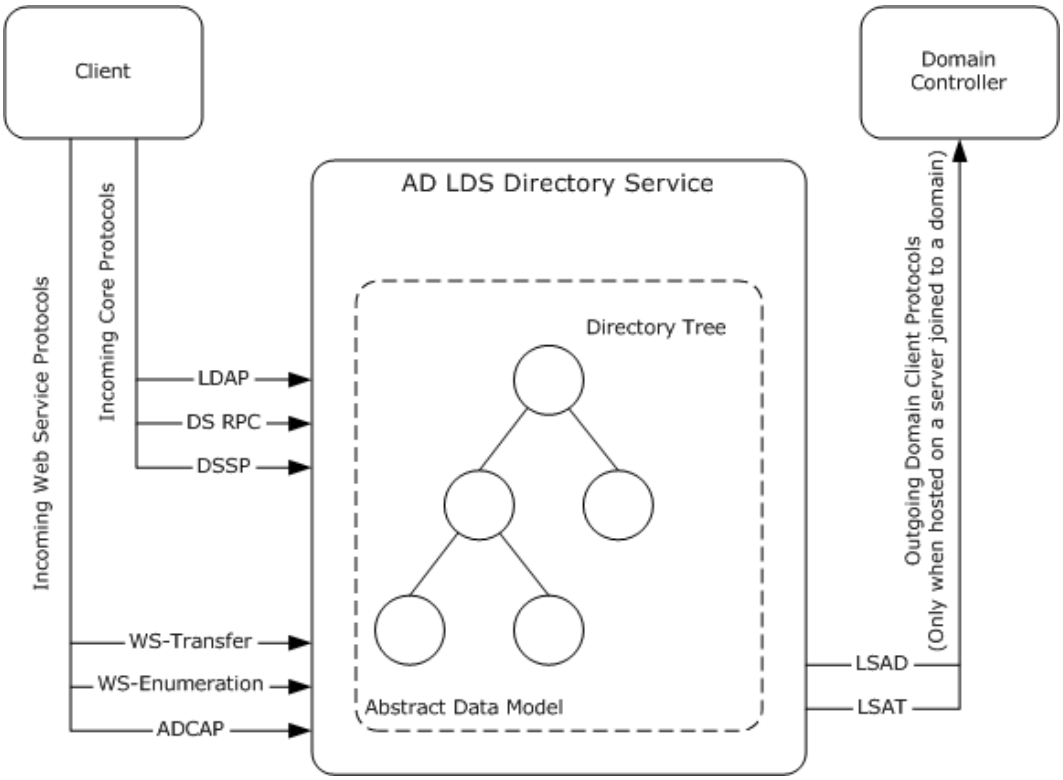

Figure 22: Incoming and Outgoing Interfaces supported by the Active Directory System in AD LDS mode

The following is a comprehensive list of the interfaces used by the Active Directory System. The list is grouped by whether the interface is an incoming interface (the system acts as a server for the protocol) or an outgoing interface (the system acts as a client of the protocol). Only AD LDS has outgoing protocol interfaces. The interfaces are also grouped by the mode of operation (AD DS or AD LDS) of the system. Interfaces that are used only by clients that communicate with the system using Web Services protocols are called out separately.

Incoming Protocols Used When the System is in AD DS or AD LDS Mode

Protocol name	Protocol description	Protocol short name and document
Lightweight Directory Access Protocol	A block protocol for creating, updating, reading, searching for, and deleting directory objects. The system's conformance to and extensions of this protocol is defined in [MS-ADTS]. There are two versions of this protocol, the older version 2 and the preferred version 3. Clients can use either version to communicate with the system.	LDAP Version 3: [RFC3377] Version 2: [RFC1777]
Directory Services Setup Remote Protocol	Exposes an RPC interface that a client can use to obtain information about the **role** of the directory server in the domain. When operating as AD LDS, it is supported in the same manner as any member server or stand-alone server on which the Active Directory System is not running.	DSSP [MS-DSSP]

Protocol name	Protocol description	Protocol short name and document
Directory Replication Service (DRS) Remote Protocol	Exposes an RPC interface that provides various services to the clients, including name format translation and expanding the group membership of an object. Which methods are supported depends on whether the system is operation as AD DS or AD LDS (see section 4.5).	DS RPC [MS-DRSR]

Additional Incoming Protocols Used for Web Services When the System is in AD DS or AD LDS Mode

Protocol name	Protocol description	Protocol short name and document
Active Directory Web Services: Custom Action Protocol	Exposes **Web Service endpoints** for managing account and topology information. When operating as AD LDS, not all the methods are supported (see section 4.5 for a list).	ADCAP [MS-ADCAP]
Web Services Transfer (WS-Transfer)	Exposes **Web Service endpoints** for creating, reading, updating, and deleting directory objects. The XML data model over which it operates is defined in [MS-ADDM]. The system MUST support the protocol extensions defined in [MS-WSTIM] and [MS-WSPELD].	WS-Transfer [WXFR]
Web Services Enumeration (WS-Enumeration)	Exposes **Web Service endpoints** for searching for directory objects. The XML data model over which it operates is defined in [MS-ADDM]. The system MUST support the protocol extensions defined in [MS-WSDS] and [MS-WSPELD].	WS-Enumeration [WSENUM]

Incoming Protocols Used Only When the System is in AD DS Mode

Protocol name	Protocol description	Protocol short name and document
Local Security Authority (Domain Policy) Remote Protocol	Exposes an RPC interface to provide clients a means to retrieve security policy information.	LSAD [MS-LSAD]
Local Security Authority (Translation Methods) Remote Protocol	Exposes an RPC interface to provide clients a means to translate SIDs of security principals to human-readable names and vice versa.	LSAT [MS-LSAT]
Security Account Manager (SAM) Remote Protocol	Exposes an RPC interface to provide clients a means of performing account maintenance, including creating, updating, reading, and deleting account information.	SAMR [MS-SAMR]

Incoming Protocols Used Only When the System is in AD LDS Mode

There are no incoming protocols exclusive to AD LDS mode.

Outgoing Protocols Used Only When the System is in AD LDS Mode and Hosted on a Server Joined to a Domain

Protocol name	Protocol description	Protocol short name and document
Local Security Authority (Translation Methods) Remote Protocol	Used by the directory service to look up the account and domain name corresponding to a SID when an LDAP bind against a bind proxy (see [MS-ADTS] section 5.1.1.5, Supported Types of Security Principals) is performed by a client.	LSAT [MS-LSAT]
Security Account Manager (SAM) Remote Protocol	When a password set or change operation is performed against an AD LDS security principal (see [MS-ADTS] section 5.1.1.5, Supported Types of Security Principals), used by the directory service to validate the new password against the domain password policy (see [MS-ADTS] sections 3.1.1.5.2.2, Constraints, and 3.1.1.5.3.2, Constraints). When a LDAP bind is performed against an AD LDS security principal, used to determine if the security principal's password has expired according to the domain password policy. Also used by the directory service when proxying a password change (see [MS-ADTS] section 3.1.1.3.1.5, Password Modify Operations) made against a bind proxy.	SAMR [MS-SAMR]

Additionally, for the purposes of verifying LDAP binds performed against security principals defined by the Active Directory domain to which the server hosting the directory service is joined (or a domain trusted by the joined domain), and for verifying LDAP binds (see [MS-ADTS] section 5.1.1, Authentication) performed against bind proxies (which correspond to a security principal in the joined domain or a trusted domain), the directory server acts as a domain client (see [MS-DISO] section 4.5.3, Domain Controller and Domain Client Functional Relationships).

5.4.1 Communications within the System

The components and individual protocols that make up the directory service communicate and share data among themselves via the shared abstract data model fields as documented in section 5.1, subject to the atomicity and coherency requirements specified in that section.

5.4.2 Communications with External Systems

The Active Directory System communicates with a number of external systems. These systems are:

- Management Tools: Management tools are used to administer, configure, and monitor the system; for example, by adjusting LDAP policies and configurable settings ([MS-ADTS] sections 3.1.1.3.4.6, LDAP Policies, and 3.1.1.3.4.7, LDAP Configurable Settings). Management tools communicate with the Active Directory System using the protocols listed in section 5.4.3.
- Applications: Applications make use of the directory for the storage and retrieval of information contained in the directory tree. Directory-enabled applications include those written by Microsoft, those written by third-party software vendors, and those custom-written for a specific deployment of the Active Directory System. Applications can communicate with the Active Directory System using any of the protocols listed in section 5.4.3. As an industry-standard protocol, LDAP is particularly widely used by applications for communicating with the Active Directory System. Applications running on a domain-joined client can also make use of the authentication and authorization services provided by the domain environment as described in [MS-DISO] and [MS-AUTHSO].

- Clients (not domain-joined): Clients that are not domain-joined can host applications or other systems that communicate with the Active Directory System using the protocols listed in section 5.4.3.
- Clients (domain-joined): Domain-joined clients can, in addition to the communication possibilities of non-domain-joined clients, interact with the Active Directory System as described in [MS-DISO].
- Kerberos Key Distribution Center: The KDC interacts with the Active Directory System as described in [MS-DISO].
- Windows Authentication Services: The Active Directory System, when operating as AD LDS, uses the outgoing protocols listed in section 5.4.4 to interact with Windows Authentication Services. When operating as AD DS, the interactions between the Active Directory System and Windows Authentication Services do not involve network traffic and take place as described in [MS-AUTHSO].

As has already been described in section 4.3.2, multiple systems, such as the Certification Authority System, Group Policy System, Message Queuing System, Print Services System, and Network Policy and Access Services System communicate with the Active Directory System for the purposes of storing and retrieving information. These systems interact with the Active Directory System in the same manner as any other application, using the protocols listed in section 5.4.3.

5.4.3 Incoming Interfaces

Abstracts of the protocols listed in this section are given in section 5.4.

The Active Directory System SHOULD implement server-side interfaces for the following protocols:

- LDAP (as described in [MS-ADTS] section 3.1.1.3, LDAP)
- DS RPC
- DSSP
- LSAD (only when operating as AD DS)
- LSAT (only when operating as AD DS)
- SAMR (only when operating as AD DS)

The Active Directory System SHOULD support the following protocols and protocol extensions on a SOAP version 1.2 binding [SOAP1.2-1/2003], on the **Web Service endpoints** specified in [MS-ADDM] section 2.1, Endpoints, and using the "net.tcp" transport [MC-NMF]. The security mechanisms on those endpoints SHOULD be as specified in [MS-ADDM] section 2.1, Endpoints. **WS-Addressing** [WSA] SHOULD be bound to SOAP as specified in [WSASB].

- WS-Transfer
- WS-Enumeration
- ADCAP
- IMDA
- WSDS
- WSPELD

Appendix A (section 8) contains the **Web Services Description Language (WSDL)** document specifying the bindings and service for the Web Service protocol interfaces exposed by the Active Directory System.

5.4.4 Outgoing Interfaces

When operating as AD LDS and hosted on a server that is joined to a domain, the Active Directory System SHOULD act as a client of the following protocols, for the purposes specified in section 5.4.

- LSAT
- SAMR

Also, when operating as AD LDS and hosted on a domain-joined server, the Active Directory System should act as a domain client as defined in [MS-DISO], for the purposes specified in section 5.4.

In this scenario, there are two independent Active Directory systems. The first system is the AD LDS instance that is running on a server joined to the domain. This system acts as a client of the domain, which contains the second Active Directory system (running as AD DS and acting as the AD DS domain controller of the domain).

When operating as AD LDS and hosted on a server that is not domain-joined, there are no outgoing interfaces since there is no domain against which to perform authentication or to which password sets and changes should be proxied.

When operating as AD DS, there are no outgoing interfaces. Interactions with Windows Authentication Services are as described in [MS-AUTHSO] and do not involve network protocol, since the Active Directory System operating as AD DS and the server side of the Windows Authentication Services are colocated on the same server.

5.5 Failure Scenarios

There are several potential failure scenarios for the Active Directory System. "Failure", in this context, does not refer to an error returned by a member protocol due to an invalid or not permitted request (for example, a request to modify a directory object when the requestor does not have the necessary permissions to do so). Such errors are part of the normal processing behavior of the system. Rather, "failure" in this context refers to conditions that can prevent the system from successfully servicing requests made by a client using the member protocols.

These failure scenarios are the following:

- Transient unavailability of durable storage (without loss or corruption of data)
- Permanent unavailability of durable storage
- Corruption of data on the durable storage
- Unavailability of networking
- Unavailability of DNS

Additionally, individual member protocols can have their own failure scenarios. Such scenarios are documented in the protocols' Technical Documents and are not repeated here.

5.5.1 Transient Unavailability of Durable Storage

As specified in section 4.2, the Active Directory System requires access to durable storage. The system MUST be able to read from and write to this storage. This storage is used to store all persisted state, including the directory tree, in the Active Directory System.

It is possible that while operating, a directory server may temporarily lose access to its durable storage. The state stored on the durable storage is intact and uncorrupt, but cannot be accessed by the directory server (for example, the disk could have become unplugged or the disk controller could have failed). In such a case, the directory server MUST NOT permit any request to succeed that requires altering the persisted state of the directory. The directory server MAY permit a request to succeed that requires retrieving state if that request can be completely and accurately answered using only the state that the directory server has available to it with the durable storage unavailable. For example, as a performance optimization, an implementation can choose to cache some state in memory, provided that doing so does not violate any transactional guarantees of the member protocols. An implementation could (but is not required to) use such cached state to respond to any requests that require retrieving state, provided the cached state is sufficient to completely and accurately respond to the request.

When rejecting a request while in this scenario, the member protocol is permitted to use any suitable error code that indicates that the directory server is unable to process the request. The system does not constrain the protocol's choice of error code.

A implementation of the system is permitted to, but not required to, monitor the durable storage system to determine if the storage becomes available again and to automatically resume normal servicing of requests. Alternatively, an implementation of the system can require administrative action to recover from such a scenario (for example, requiring a restart of the directory server).

Since the state stored on the durable storage was not altered while the storage was offline, and since the directory server rejected any requests that would have required a modification of the state during that period, after the durable system becomes available and the system resumes normal servicing of requests, the abstract data models of the system's protocols SHOULD be in the same state as they were immediately prior to the storage becoming unavailable.

5.5.2 Permanent Unavailability of Durable Storage

The preceding failure scenario dealt with the case of a transient unavailability of the durable storage. However, it is also possible that the durable storage on which the system's state is stored becomes permanently unavailable; for example, due to a non-repairable hardware failure in the disk media. In this case, all of the state stored in the storage system is lost.

As in the case of a transient unavailability of the storage system, the directory server MUST NOT permit any request to succeed that requires altering the persisted state of the directory. The directory server MAY permit a request to succeed that requires retrieving state if that request can be completely and accurately answered using only the state that the directory server has available to it with the durable storage unavailable.

When rejecting a request while in this scenario, the member protocol is permitted to use any suitable error code that indicates that the directory server is unable to process the request. The system does not constrain the protocol's choice of error code.

As the system will generally not have any means to determine on its own whether the storage system is temporarily or permanently unavailable, the key difference between this scenario and the previous scenario is that recovery in this scenario will typically require administrative intervention. Upon making any necessary repairs or replacements of the storage system to return it to service,

the administrator SHOULD restore the state of the directory server from the most recent backup copy. The means of backing up and restoring such state is implementation-specific.

After the backup is restored, the state of the directory server is as it was at the time the backup was taken. Further, any changes to the state that were replicated to one or more replica directory servers in the directory service subsequent to the time the backup was taken, can be regained after the restore through replication.

5.5.3 Data Corruption

While a durable storage system should do everything possible to protect the integrity of the data stored on it, data corruption nonetheless remains a possibility even in the best maintained storage system. Such corruption could occur while the storage system is online, or it could occur during a transient outage of the storage system.

In such a case, the directory server MUST detect such data corruption. An implementation is permitted to use any means of detection that it has at its disposal. The directory server MUST NOT permit any protocol requests that require access to the corrupted data to succeed. When rejecting a request while in this scenario, the member protocol is permitted to use any suitable error code that indicates that the directory server is unable to process the request. The system does not constrain the protocol's choice of error code.

After data corruption has been detected, recovery proceeds similar to the case of the durable storage being permanently unavailable. Essentially, data that cannot be trusted is treated the same as no data at all.

5.5.4 Unavailability of Networking

A functional networking system is vital to the ability of clients to communicate with the directory server. If the networking system becomes unavailable, clients will not be able to send any new requests to the directory server, nor will they be able to receive any responses to requests that they may have outstanding. Networking could become unavailable due to a hardware failure (such as the failure of a network switch or router) or a software problem (such as misconfiguration of a routing table).

When in this state, the directory server will not be able to respond to any requests that clients attempt to send it. As such, clients SHOULD always enforce a timeout on any requests that they send to the server so they do not hang indefinitely waiting for a response.

Once the network becomes available again and communications are restored between the client(s) and the directory server, the directory server SHOULD resume processing requests in accord with the behavior specified in the Technical Documents. The loss of networking does not in itself cause any change in the state of the abstract data models of the system's protocols.

5.5.5 Unavailability of DNS

DNS can be used by clients of the Active Directory System in order to locate directory servers using the algorithms described in [MS-ADTS] section 6.3, Publishing and Locating a Domain Controller. As such, if DNS ceases to become available (for example, by failure of the DNS servers), any clients that use those location algorithms will be unable to find a directory server to which to send their requests. This will not affect any connections that clients already have to the directory server. Additionally, clients are permitted to use other means of locating a directory server (for example, by prompting the user to enter the IP address of a directory server) or to store and reuse the address of a previously located directory server. Such clients will also not be immediately affected by the loss of DNS.

Once DNS is restored to normal operating behavior, clients dependent on the location algorithms of [MS-ADTS] section 6.3, Publishing and Locating a Domain Controller, will once again be able to locate directory servers.

6 System Details

This section contains the details that complete the descriptions in earlier sections of the document. These details are needed to understand and implement this system.

6.1 Architectural Details

This section contains a set of scenarios illustrating common uses of the Active Directory System. The scenarios are:

- Provision a user account using the LDAP protocol.
- Provision a user account using the SAMR protocol.
- Change a user account's password.
- Determine the group membership of a user.
- Delete a user account.
- Obtain a list of user accounts using the Web Services protocols.
- Obtain a list of user accounts using the LDAP protocol.
- Manage groups and their memberships.
- Delete a group.
- Extend the schema to support an application by adding a new class.
- Extend the schema to support an application by adding a new attribute.
- Extend the schema to support an application by adding an attribute to a class.
- Partition directory data with organizational units.
- Store application data in the directory.
- Manage access control on directory objects.
- Raise the domain functional level.

A key aspect of these scenarios is that multiple protocols, such as LDAP and SAMR, can be used to effect the same state change, such as provisioning a user. Once that state change is made, any protocol that provides access to that state (not just the protocol that originally created the state) can be used to further modify that state. For example, the user created using the SAMR protocol can have his or her password changed using the LDAP protocol, or be queried for using the Web Services protocols.

6.1.1 Provision a User Account Using the LDAP Protocol

In this scenario, an Administrator provisions a user account using the LDAP protocol. To perform this task, an Administrator runs a Client Application from a client computer, targeting a directory server in the Active Directory System. The Client Application creates a user account and sets its user properties using the LDAP protocol, and sets the user account's password using the Kerberos protocol.

This scenario applies only to AD DS.

This scenario uses the LDAP and Kerberos protocols.

The scenario covers the use cases in section 3.3.4.2.1 and section 3.3.4.2.2.

6.1.1.1 Initial System State

The Active Directory System must meet all preconditions specified in sections 3.3.4.2.1.

6.1.1.2 Server Activity Diagram

The server activity diagram for this scenario is illustrated in the following figure. In this scenario the Client Application connects to a directory server in the Active Directory System and communicates with it using the LDAP and Kerberos protocol. The activity diagram shows the actions the directory server takes as it transitions from an initial state to the state of a successful account creation.

The server activities are as follows:

1. Upon receiving an LDAP bind request ([RFC2251] section 4.2, Bind Operation) from the Client Application, the directory server interacts with the Windows Authentication Services which authenticates the Administrator using the supplied credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task).

2. The directory server receives the LDAP add request, containing data about the new user object to be created. The directory server validates that the Administrator has the necessary access to perform the operation. If the Administrator does not have the necessary access to perform the operation, the directory server rejects the LDAP add request.

3. The directory server verifies the constraints for add operations are satisfied as outlined in [MS-ADTS] sections 3.1.1.5.1, General, and 3.1.1.5.2, Add Operation. If the constraints are not satisfied the add request will be terminated with an appropriate error.

4. Additional constraints specific to account creation are validated as outlined in [MS-SAMR] section 3.1.1.6, Attribute Constraints for Originating Updates. If the constraints are not satisfied, the add request will be terminated with an appropriate error.

5. The user object is created in the directory and populated with the data supplied in the request. Additional attributes on the object are populated based on the server's processing rules as outlined in [MS-ADTS] section 3.1.1.5.2.4, Processing Specifics, and [MS-SAMR] section 3.1.1.8, Attribute Triggers for Originating Updates.

6. The directory server receives password information from the KDC. The password is set to the supplied value.

7. The directory server receives LDAP modify requests containing new values for the **userAccountControl** and **pwdLastSet** attributes. The attributes are updated with the supplied values.

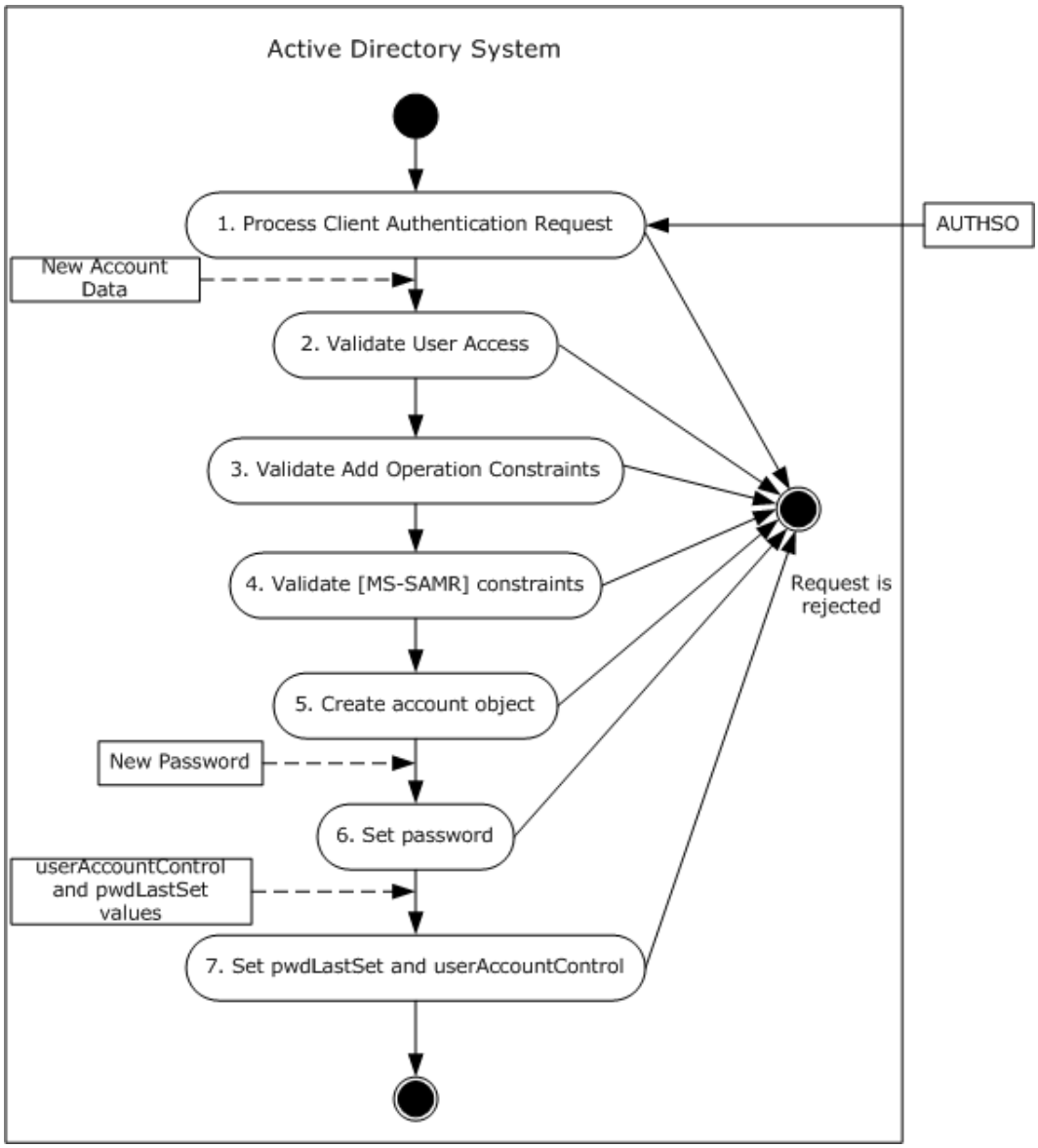

Figure 23: Server activity diagram for provisioning a user account using the LDAP protocol

6.1.1.3 Sequence of Events

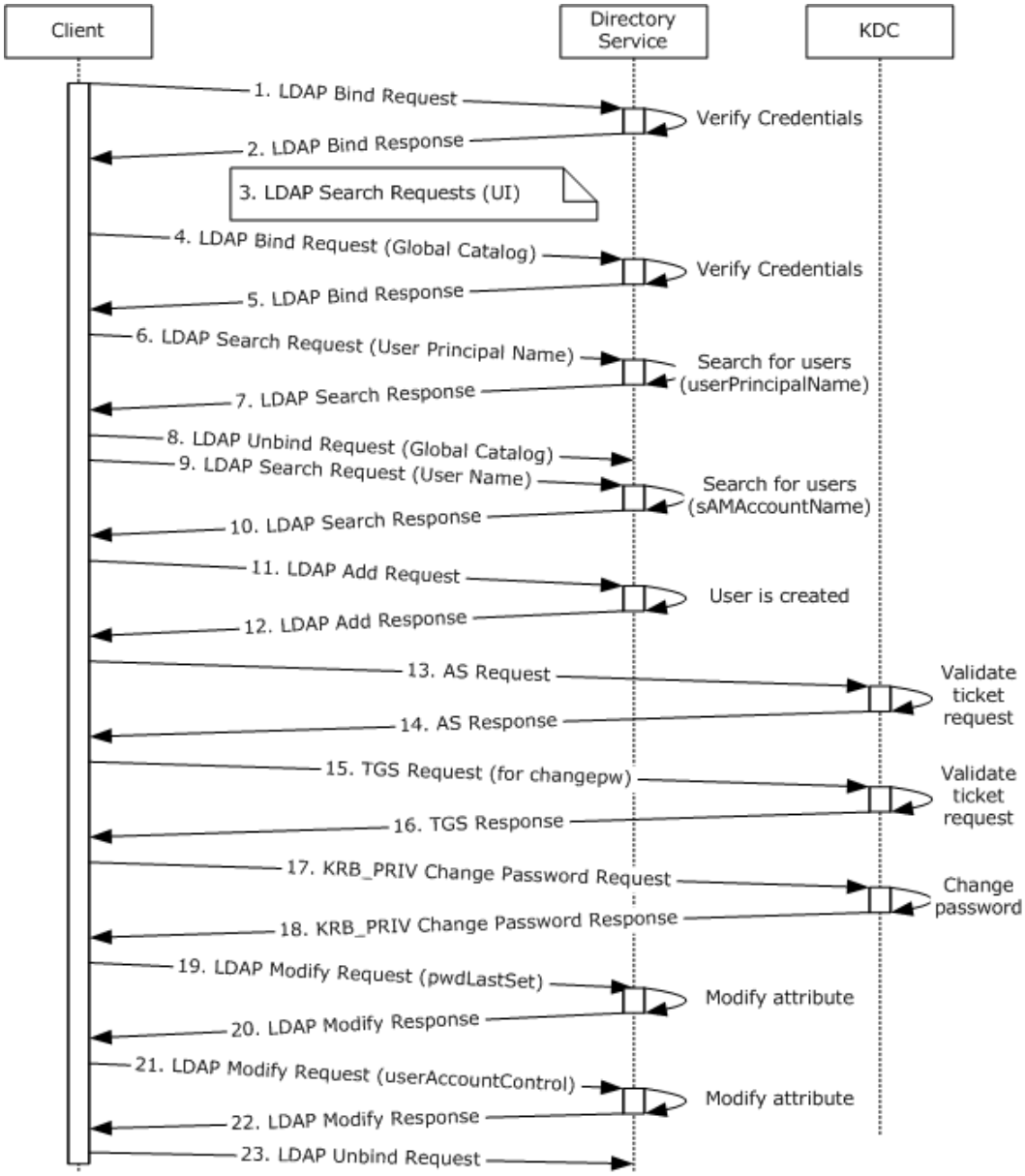

Figure 24: Message flow for provisioning a user account using the LDAP protocol

Unless otherwise noted, all responses that include a return code contain a return code indicating that the operation was successfully performed.

1. The program (see section 6.1.1) establishes an LDAP connection to the directory server. An LDAP bind request ([RFC2251] section 4.2, Bind Operation) is sent to the directory server with the credentials of an administrator.

2. The directory server uses one of the methods defined elsewhere (see [MS-AUTHSO] section 4, Interactive Domain Logon Task) to verify the credentials. Depending on the negotiated

authentication method, this may involve additional client and server interactions not directly relevant here. After that verification, the directory server sends an LDAP bind response ([RFC2251] section 4.2.3, Bind Response) to the client.

3. At this point the client sends LDAP search requests ([RFC2251] section 4.5.1, Search Request) to populate data in the tool's user interface. This step is necessary only for user-interface display purposes specific to the example shown in the figure captioned, "Message flow for provisioning a user account using the LDAP protocol".

4. An LDAP connection to a **global catalog** is established. An LDAP bind request ([RFC2251] section 4.2, Bind Operation) to the global catalog is sent with the administrator's credentials.

5. The global catalog verifies the credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task) and sends an LDAP bind response ([RFC2251] section 4.2.3, Bind Response).

6. An LDAP search request ([RFC2251] section 4.5.1, Search Request) is sent to the global catalog, querying the entire forest, starting at the root of the forest, looking for security principals that have the same user principal name as the requested user principal name of the new account, to verify that the requested user principal name of the new account is not currently in use.

7. The global catalog sends an LDAP search response ([RFC2251] section 4.5.2, Search Result) listing any accounts that have the user principal name specified.

8. An LDAP unbind request ([RFC2251] section 4.3, Unbind Operation) is sent to the global catalog. The LDAP connection to the global catalog is closed.

9. The program sends an LDAP search request ([RFC2251] section 4.5.1, Search Request) to the directory, querying the entire domain, starting at the root of the domain, looking for security principals that have the same name (stored in the sAMAccountName attribute) as the one requested for the new account, to ensure that the name specified by the administrator is not currently in use.

10. The server sends an LDAP search response ([RFC2251] section 4.5.2, Search Result) listing any accounts that have the user name specified.

For the purposes of this scenario, assume that the LDAP search responses for the user principal name and the account name do not contain any accounts, that is, there were no matches. The program has now verified that the user principal name and user name chosen are not currently in use and continues with the add operation.

11. An LDAP add request ([RFC2251] section 4.7, Add Operation) is sent to the server. The LDAP add operation contains the distinguishedName, sAMAccountName, userPrincipalName, displayName, and givenName of the new user, and specifies that the object class of the object to be created is user.

12. The server processes the add request ([RFC2251] section 4.7, Add Operation) and performs validation as described in [MS-ADTS] sections 3.1.1.5.1, General, and 3.1.1.5.2, Add Operation. It then sends an LDAP add response indicating success.

Now that the user has been created, the program starts setting additional attributes provided by the administrator. It begins by setting the password for the new account. This is done via Kerberos messages sent to the Kerberos Key Distribution Center (KDC).

13. The client begins by sending a Kerberos **AS** request ([RFC4120] section 3.1, The Authentication Service Exchange) to the Authentication Service (AS) requesting a **Ticket-Granting Ticket (TGT)** that it will use for later authentication.

14. The service validates the credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task) in the AS request and sends an AS response ([RFC4120] section 3.1, The Authentication Service Exchange) with a TGT.

15. The client sends a Kerberos **TGS** ([RFC4120] section 3.3, The Ticket-Granting Service (TGS) Exchange) request to the Ticket-Granting Service (TGS) using the TGT and requesting a **service ticket** for the kadmin/changepw service ([RFC3244]), which provides functionality to change an account's password via Kerberos.

16. The service validates the credentials in the TGS request and sends a TGS response ([RFC4120] section 3.3, The Ticket-Granting Service (TGS) Exchange) with the service ticket.

17. Using the service ticket, the client sends a **KRB_PRIV** change password request ([RFC4120] section 3.5, The KRB_PRIV Exchange, and [RFC3244] section 2, The Protocol) to the Kerberos password-changing service with the new password for the account.

18. The password-changing service processes the request and sends a KRB_PRIV response ([RFC4120] section 3.5, The KRB_PRIV Exchange, and [RFC3244] section 2, The Protocol) indicating success. As part of this change-password operation, the Active Directory database is modified according to the sequence of actions described in [MS-SAMR] section 3.1.1.8.5, clearTextPassword.

At this point the Kerberos operations are completed. The client now continues to set remaining attributes via LDAP.

19. If the administrator indicated that the user must change his or her password at next logon, the client sends an LDAP modify request ([RFC2251] section 4.6, Modify Operation) setting the pwdLastSet attribute to 0. This setting tells the server that the user's password has expired and must be changed the next time the new user attempts to log on.

20. The server processes the request and sends a response ([RFC2251] section 4.6, Modify Operation).

The client computes the new desired value for the userAccountControl attribute. At a minimum this includes the ADS_UF_NORMAL_ACCOUNT bit ([MS-ADTS] section 2.2.16, userAccountControl Bits) and may contain additional bits depending on administrator-provided values.

21. The client sends an LDAP modify request ([RFC2251] section 4.6, Modify Operation) with the new value for userAccountControl.

22. The server processes the request and sends a response ([RFC2251] section 4.6, Modify Operation).

The new user account (represented as a directory object of class user) has now been created. The user object has been populated with all specified attributes.

23. The client sends an LDAP unbind request ([RFC2251] section 4.3, Unbind Operation) to the server. The LDAP connection to the directory server is closed.

6.1.1.4 Final System State

The new user object has been provisioned in the directory with the attributes specified. No other state in the directory has changed.

6.1.2 Provision a User Account Using the SAMR Protocol

A common administrative task is to provision an account for a new user. As shown in the previous section, this can be done using LDAP. Another way of accomplishing this is to use the SAMR protocol

to communicate with the Active Directory System. Regardless of which protocol is chosen, the end state is the same: a new user object is created in the directory tree.

To perform this task, an administrator runs a Client Application using the SAMR protocol from a client computer targeting a directory server in the Active Directory System.

This scenario applies only to AD DS.

This scenario differs from the previous scenario (section 6.1.1) in that it uses the SAMR protocol rather than LDAP to create the user and only provides a minimal set of attributes for the newly created account.

The scenario covers the use cases in section 3.3.4.2.1, Create a New Account - Client Application.

6.1.2.1 Initial System State

The Active Directory System must meet all preconditions specified in sections 3.3.4.2.1 and 3.3.4.1.1.

6.1.2.2 Server Activity Diagram

The server activity diagram for this scenario is illustrated in the following figure. In this scenario the Client Application connects to a directory server in the Active Directory System and communicates with it using the SAMR protocol. The activity diagram shows the actions the directory server takes as it transitions from an initial state to the state of a successful account creation.

The server activities are as follows:

1. Upon receiving an incoming connection from the Client Application, the directory server interacts with the Windows Authentication Services which authenticates the Administrator using the supplied credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task).

2. The server receives a request from the client application for a server handle to be used in later requests. The directory server validates that the Administrator has the necessary access to acquire the server handle. If the validation fails, the directory server rejects the request. Otherwise, the server processes the request and returns a server handle to the client.

3. The server receives a request from the client application for a domain handle to be used in later requests. The server validates that the server handle associated with the request has the required access rights ([MS-SAMR] section 3.1.2.1, Standard Handle-Based Access Checks) and the directory server validates that the Administrator has the necessary access to acquire the domain handle. If the validation fails, the directory server rejects the request. Otherwise, the server processes the request and returns a domain handle to the client.

4. The server receives a request from the client application containing data about the new account to be created. Constraints specific to account creation are validated as outlined in [MS-SAMR] section 3.1.1.6, Attribute Constraints for Originating Updates. If the constraints are not satisfied the add request will be terminated with an appropriate error.

5. The account object is created in the directory and populated with the data supplied in the request. Additional attributes on the object are populated based on the server's processing rules as outlined in [MS-ADTS] sections 3.1.1.5.1, General, and 3.1.1.5.2, Add Operation, and [MS-SAMR] section 3.1.1.8, Attribute Triggers for Originating Updates.

6. The client application sends the server requests to query information about the userAccountControl attribute on the newly created account. The server sends a response with the

requested information. If the request is rejected by the server the operation will be terminated with an appropriate error.

7. The server receives a request with new values for userAccountControl and the password of the account. The server processes this request and sets the attributes to the specified values. If the request is rejected by the server the operation will be terminated with an appropriate error.

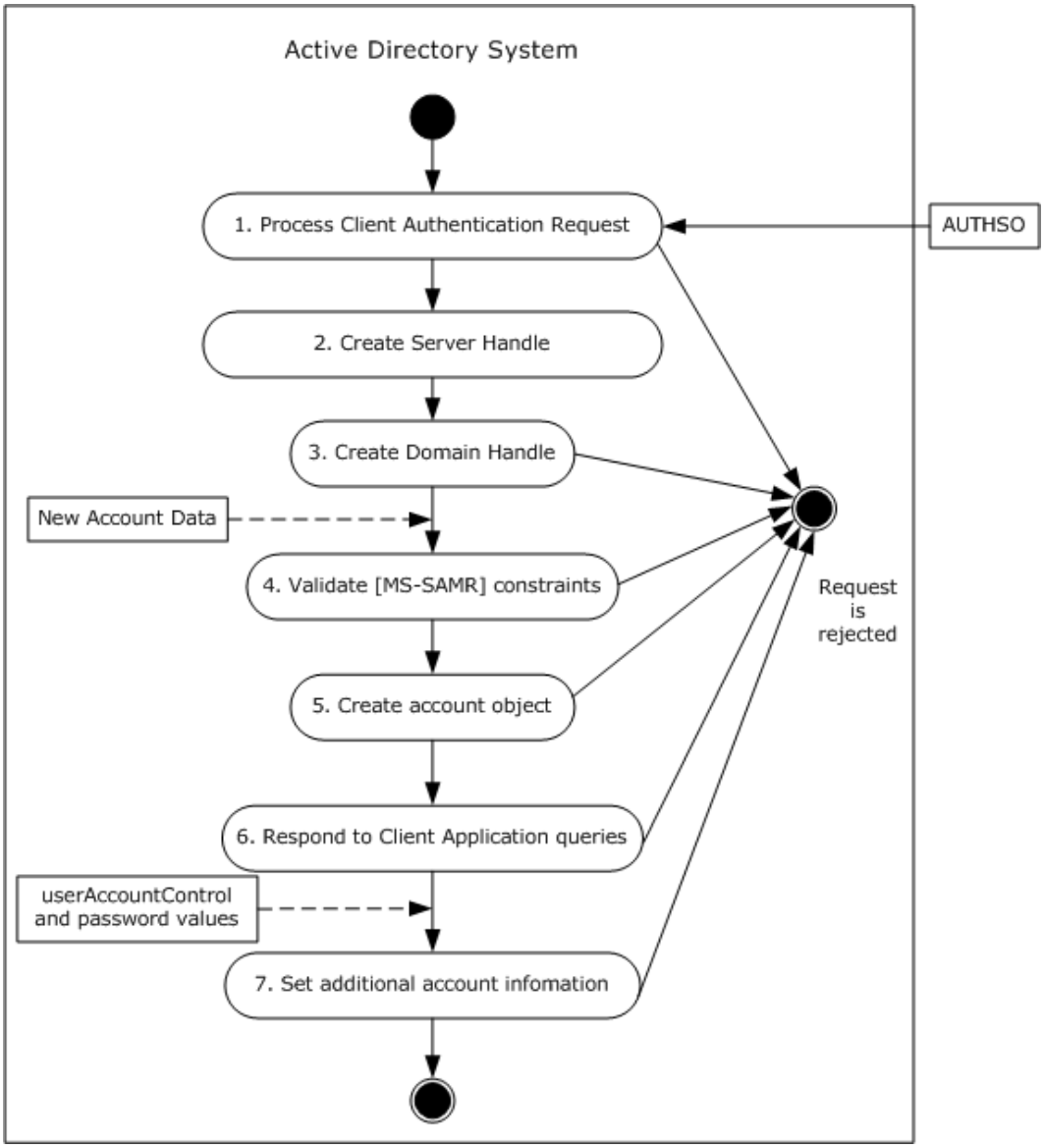

Figure 25: Server activity diagram for provisioning a user account using the SAMR protocol

6.1.2.3 Sequence of Events

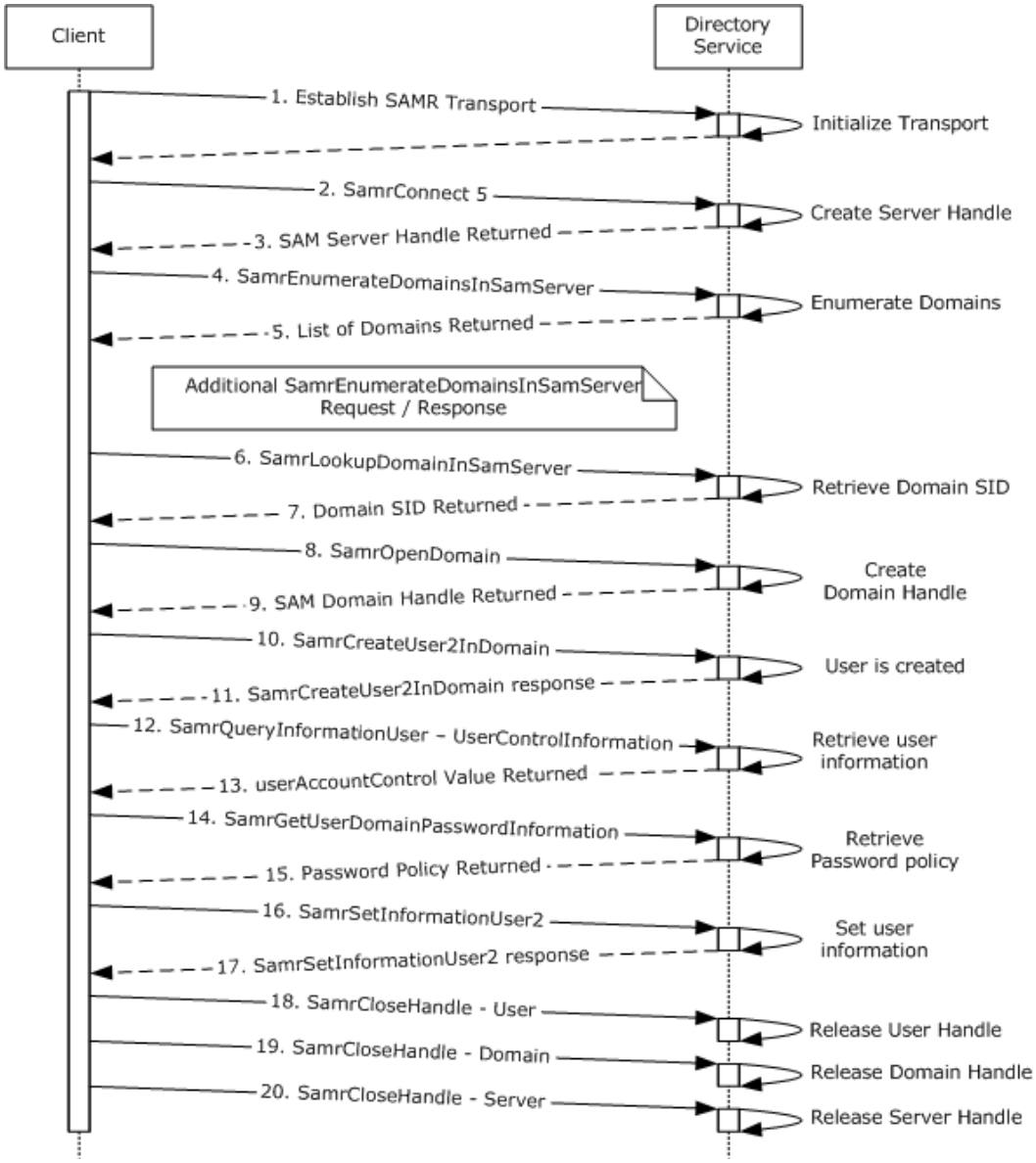

Figure 26: Message flow for provisioning a user account using the SAMR protocol

Unless otherwise noted, all responses that include a return code contain a return code indicating that the operation was successfully performed.

1. The **client** binds to the SAMR **endpoint** on the server using a supported transport, as specified in [MS-SAMR] section 2.1, Transport.

2. The next step is to open a SAMR handle to the directory server. The client application sends a SamrConnect5 request ([MS-SAMR] section 3.1.5.1.1, SamrConnect5 (Opnum 64)) with the *DesiredAccess* parameter set to MAXIMUM_ALLOWED ([MS-SAMR] section 2.2.1.1, Common ACCESS_MASK Values) to the server requesting a server handle.

3. The server processes the request ([MS-SAMR] section 3.1.5.1.1, SamrConnect5 (Opnum 64)) and sends a response with the server handle to be used by later calls.

Before continuing by opening a SAMR handle to the domain, the client application must first know the security identifier (SID) of the domain. This is determined using steps 4-7.

4. The client application sends a SamrEnumerateDomainsInSamServer request ([MS-SAMR] section 3.1.5.2.1, SamrEnumerateDomainsInSamServer (Opnum 6)) using the server handle it previously obtained in step 3.

5. The server processes the request ([MS-SAMR] section 3.1.5.2.1, SamrEnumerateDomainsInSamServer (Opnum 6)) and returns a list of all domains hosted by the server.

The client application repeats sending SamrEnumerateDomainsInSamServer if the directory service returns STATUS_MORE_ENTRIES ([MS-SAMR] section 3.1.5.2.1, SamrEnumerateDomainsInSamServer (Opnum 6)) to indicate that there is additional data to retrieve.

6. The client application sends a SamrLookupDomainInSamServer request ([MS-SAMR] section 3.1.5.11.1, SamrLookupDomainInSamServer (Opnum 5)) using the server handle from step 3. In this request it specifies the name that was returned in step 5, which corresponds to the domain object.

7. The server processes the request ([MS-SAMR] section 3.1.5.11.1, SamrLookupDomainInSamServer (Opnum 5)) and returns the domain SID.

The client application has now obtained the domain SID and can use it to open a SAMR handle to the domain.

8. The client application sends a SamrOpenDomain request ([MS-SAMR] section 3.1.5.1.5, SamrOpenDomain (Opnum 7)) using the server handle it previously obtained in step 3 and the domain SID it obtained in step 7, and the *DesiredAccess* parameter set to MAXIMUM_ALLOWED ([MS-SAMR] section 2.2.1.1, Common ACCESS_MASK Values).

9. The server processes this request ([MS-SAMR] section 3.1.5.1.5, SamrOpenDomain (Opnum 7)) and returns a response with a domain handle.

Now that the client application has the domain handle, it can use this handle to create users in the domain.

10. The client application sends a SamrCreateUser2InDomain request ([MS-SAMR] section 3.1.5.4.4, SamrCreateUser2InDomain (Opnum 50)) with the *DesiredAccess* parameter set to MAXIMUM_ALLOWED ([MS-SAMR] section 2.2.1.1, Common ACCESS_MASK Values) to create the new user. The request includes the domain handle received earlier, the user name of the new user to create, and the AccountType parameter set to USER_NORMAL_ACCOUNT ([MS-SAMR] section 2.2.1.12, USER_ACCOUNT Codes). The request also specifies the account type (in this case USER_NORMAL_ACCOUNT) and the desired access on the returned user handle (in this case USER_WRITE_ACCOUNT and USER_FORCE_PASSWORD_CHANGE so as to be able to update the account in subsequent steps).

11. The server processes the request ([MS-SAMR] section 3.1.5.4.4, SamrCreateUser2InDomain (Opnum 50)) and creates the new user. It creates a user directory object with sAMAccountName set equal to the user name specified by the client application in the request. The server then returns a response with a user handle for the newly-created user, the **relative identifier (RID)** of the new user, and the access that was granted on the user handle (in accordance with the behavior specified in [MS-SAMR] section 3.1.5.4.4, SamrCreateUser2InDomain (Opnum 50)).

12. The client application sends a SamrQueryInformationUser request ([MS-SAMR] section 3.1.5.5.6, SamrQueryInformationUser (Opnum 36)), using the user handle it obtained in the previous step and requesting that UserControlInformation be returned, in order to retrieve the value of the userAccountControl attribute of the newly-created user.

13. The server processes the request ([MS-SAMR] section 3.1.5.5.6, SamrQueryInformationUser (Opnum 36)) and returns the value of the userAccountControl attribute (in accordance with the behavior specified in [MS-SAMR] sections 3.1.5.4.4, SamrCreateUser2InDomain (Opnum 50), and 3.1.5.14.11, User Field to Attribute Name Mapping) of the newly-created user.

14. To obtain password policy information, the client application sends a SamrGetUserDomainPasswordInformation request ([MS-SAMR] section 3.1.5.13.3, SamrGetUserDomainPasswordInformation (Opnum 44)) using the user handle it previously obtained for the newly-created user in step 11.

15. The server processes the request ([MS-SAMR] section 3.1.5.13.3, SamrGetUserDomainPasswordInformation (Opnum 44)) and returns information about the password policy that applies to the newly-created user.

16. The client application sets the password of the newly-created user by sending a SamrSetInformationUser2 request ([MS-SAMR] section 3.1.5.6.4, SamrSetInformationUser2 (Opnum 58)), specifying a UserInformationClass of UserInternal4InformationNew. As part of this request, it also updates the value of the user's userAccountControl attribute (whose current value was previously retrieved in steps 12 and 13) with any additional bits ([MS-SAMR] section 2.2.7.1, Common User Fields) based on administrator-provided values.

17. The server processes the request ([MS-SAMR] section 3.1.5.6.4, SamrSetInformationUser2 (Opnum 58)) and returns a response indicating that the user was successfully updated.

18-20. The client application must perform cleanup by closing all the handles it has opened during the session. This is done by calling SamrCloseHandle ([MS-SAMR] section 3.1.5.13.1, SamrCloseHandle (Opnum 1)) with SamHandle set to the handle the client is attempting to close. The client application closes the handles in the reverse order they were received (namely, the user handle, domain handle, and server handle).

As the final step, the transports created for communication are closed.

6.1.2.4 Final System State

The new user object has been created in the directory. No other state in the directory has changed.

6.1.3 Change a User Account's Password

In this scenario, a User changes the password on their account by using the SAMR protocol. To perform this task, a User runs a Client Application from a client computer, targeting a directory server in the Active Directory System. The Client Application changes the account's password using the SAMR protocol.

This scenario applies only to AD DS.

This scenario uses the SAMR protocol.

The scenario covers the use cases in section 3.3.4.2.3, Change and existing account's password – client application.

6.1.3.1 Initial System State

The Active Directory System must meet all preconditions specified in section 3.3.4.2.3.

6.1.3.2 Server Activity Diagram

The server activity diagram for this scenario is illustrated in the following figure. In this scenario the Client Application connects to a directory server in the Active Directory System and communicates with it using the SAMR protocol. The activity diagram shows the activities the directory server performs as it transitions from an initial state to the state of a successful account password change.

The server activities are as follows:

1. Upon receiving an incoming connection from the Client Application, the directory server interacts with the Windows Authentication Services which authenticates the Administrator using the supplied credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task).

2. The server receives a request from the client application for a server handle to be used in later requests. The directory server validates that the user has the necessary access to acquire the server handle. If the validation fails, the directory server rejects the request; Otherwise, the server processes the request and returns a server handle to the client.

3. The server receives a request from the client application for a domain handle to be used in later requests. The server validates that the server handle associated with the request has the required access rights ([MS-SAMR] section 3.1.2.1, Standard Handle-Based Access Checks) and the directory server validates that the user has the necessary access to acquire the domain handle. If the validation fails, the directory server rejects the request. Otherwise, the server processes the request and returns a domain handle to the client.

4. The server receives a request from the client application for a user handle to be used in later requests. The server validates that the domain handle associated with the request has the required access rights ([MS-SAMR] section 3.1.2.1, Standard Handle-Based Access Checks) and the directory server validates that the user has the necessary access to acquire the user handle. If the validation fails, the directory server rejects the request. Otherwise, the server processes the request and returns a user handle.

5. The server receives a change password request from the Client Application. This request contains the old and new password hashes for the account. The server validates the inputs and processes the request as outlined in [MS-SAMR] section 3.1.5.10.1, SamrChangePasswordUser (Opnum 38). If the request is rejected by the server, the operation will be terminated with an appropriate error.

6. The server updates the account object with the new password. The server sends a response to the Client Application indicating the change password request was successfully processed.

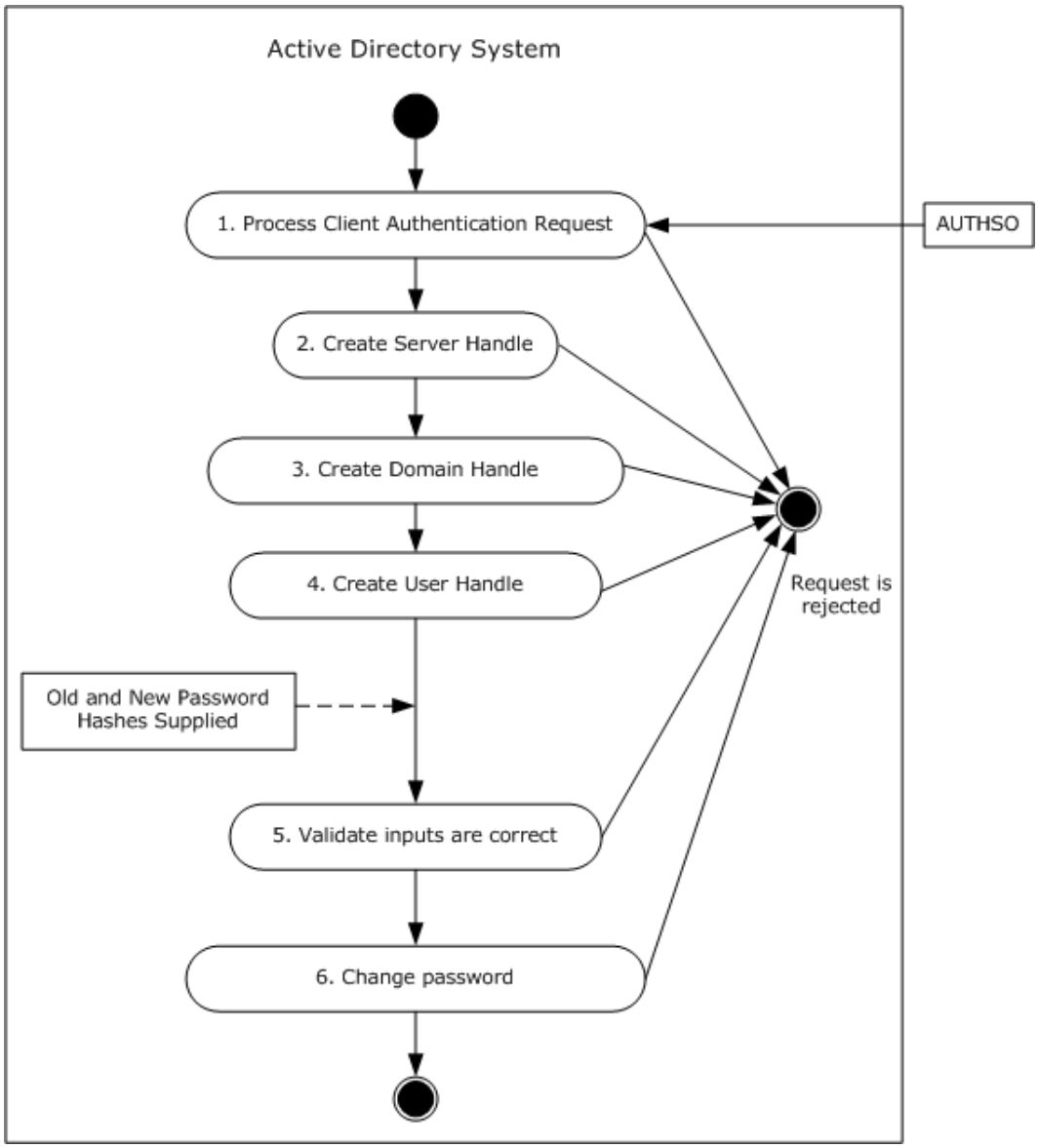

Figure 27: Server activity diagram for changing a user account's password

6.1.3.3 Sequence of Events

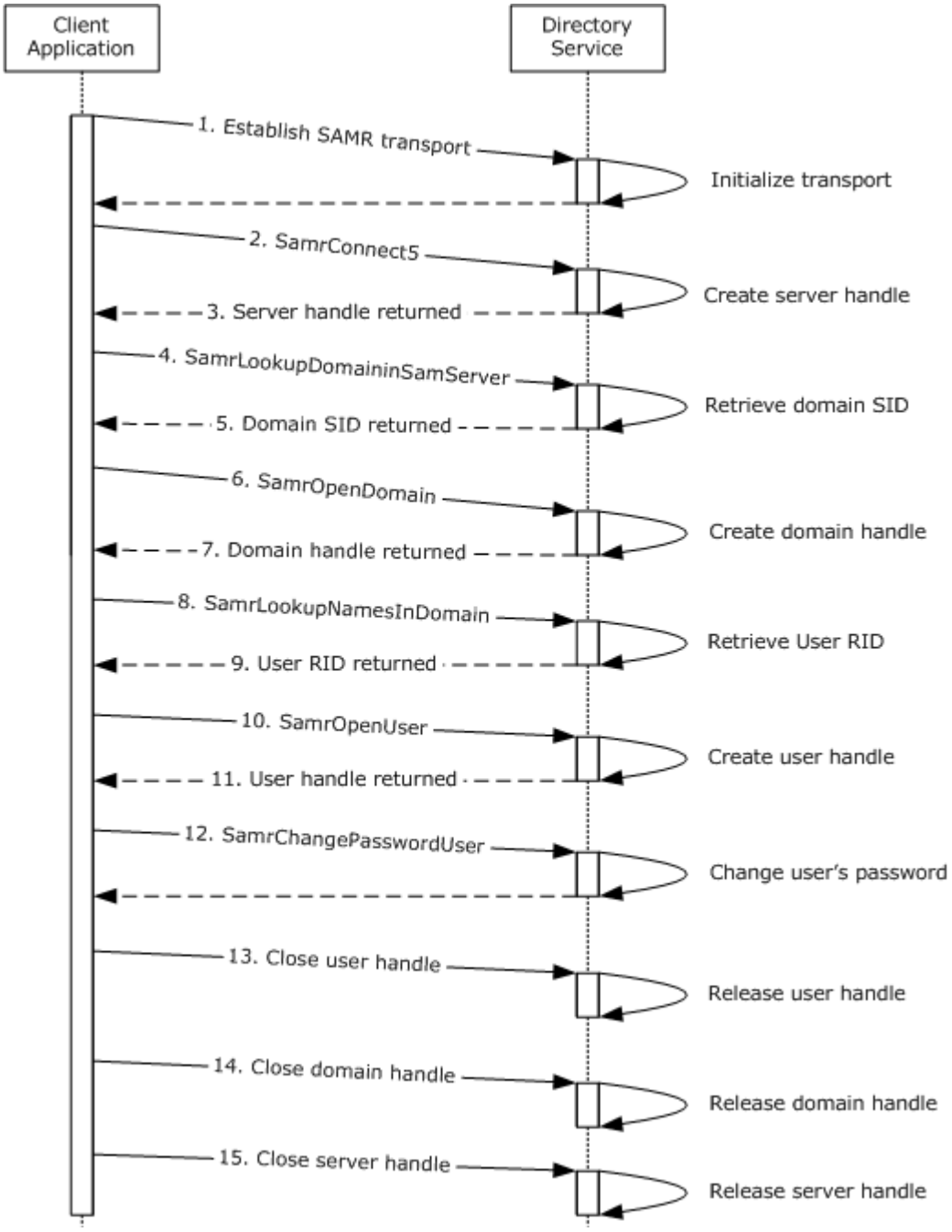

Figure 28: Message flow for changing a user account's password

Unless otherwise noted, all responses that include a return code contain a return code indicating that the operation was successfully performed.

1. The client binds to the SAMR endpoint on the server using a supported transport, as specified in [MS-SAMR] section 2.1, Transport.

2. The next step is to open a SAMR handle to the directory server. The client application sends a SamrConnect5 request ([MS-SAMR] section 3.1.5.1.1, SamrConnect5 (Opnum 64)) with the *DesiredAccess* parameter set to MAXIMUM_ALLOWED ([MS-SAMR] section 2.2.1.1, Common ACCESS_MASK Values) to the server requesting a server handle.

3. The server processes the request ([MS-SAMR] section 3.1.5.1.1, SamrConnect5 (Opnum 64)) and sends a response with the server handle to be used by later calls.

   Before continuing by opening a SAMR handle to the domain, the Client Application must first know the security identifier (SID) of the domain. This is determined using steps 4-7.

4. The client application sends a SamrLookupDomainInSamServer request ([MS-SAMR] section 3.1.5.11.1, SamrLookupDomainInSamServer (Opnum 5)) using the server handle from step 2. In this request it specifies the domain name for the account.

5. The server processes the request ([MS-SAMR] section 3.1.5.11.1, SamrLookupDomainInSamServer (Opnum 5)) and returns the domain SID.

   The client application has now obtained the domain SID and can use it to open a SAMR handle to the domain.

6. The client application sends a SamrOpenDomain request ([MS-SAMR] section 3.1.5.1.5, SamrOpenDomain (Opnum 7)) using the server handle it previously obtained in step 2, the domain SID it obtained in step 5, and the *DesiredAccess* parameter set to MAXIMUM_ALLOWED ([MS-SAMR] section 2.2.1.1, Common ACCESS_MASK Values).

7. The server processes this request ([MS-SAMR] section 3.1.5.1.5, SamrOpenDomain (Opnum 7)) and returns a response with a domain handle.

   The client application must now get the relative identifier (RID) of the user so that it can open a user handle.

8. The client application sends a SamrLookupNamesInDomain request ([MS-SAMR] section 3.1.5.11.2, SamrLookupNamesInDomain (Opnum 17)). The request includes the domain handle and sAMAccountName attribute.

9. The server processes the request ([MS-SAMR] section 3.1.5.11.2, SamrLookupNamesInDomain (Opnum 17)) and returns the RID of the user account.

   The client application now has the user account RID and can use it to open a handle to the user.

10. The client application sends a SamrOpenUser request ([MS-SAMR] section 3.1.5.1.9, SamrOpenUser (Opnum 34)). The request includes the domain handle, the RID of the user, and the *DesiredAccess* parameter set to MAXIMUM_ALLOWED ([MS-SAMR] section 2.2.1.1, Common ACCESS_MASK Values).

11. The server processes the request ([MS-SAMR] section 3.1.5.1.9, SamrOpenUser (Opnum 34)) and returns a response with a user handle.

12. Now that the client application has a handle to the user it calls SamrChangePasswordUser to change the user's password. The request includes the user handle and hashes of the password as specified in the processing rules below. The server processes the list and changes the password of the user. Refer to [MS-SAMR] section 3.1.5.10.1, SamrChangePasswordUser (Opnum 38), for the algorithms to compute the hashes.

1. Compute the OldLmEncryptedWithNewLm and NewLmEncryptedWithOldLm hashes.
2. Compute the OldNtEncryptedWithNewNt and NewNtEncryptedWithOldNt hashes.
3. Call SamrChangePasswordUser with parameters set as specified in the following table:

Parameter	Value
UserHandle	User handle from step 11.
LmPresent	If any errors are encountered in step 12.1, set this value to FALSE. Otherwise set this value to TRUE.
OldLmEncryptedWithNewLm	If any errors are encountered in step 12.1, set this value to NULL. Otherwise set this value to the computed value of OldLmEncryptedWithNewLm from step 12.1.
NewLmEncryptedWithOldLm	If any errors are encountered in step 12.1, set this value to NULL. Otherwise set this value to the computed value of NewLmEncryptedWithOldLm from step 12.1.
NtPresent	TRUE
OldNtEncryptedWithNewNt	Value of OldNtEncryptedWithNewNt from step 12.2.
NewNtEncryptedWithOldNt	Value of NewNtEncryptedWithOldNt from step 12.2.
NtCrossEncryptionPresent	FALSE
NewNtEncryptedWithNewLm	NULL
LmCrossEncryptionPresent	FALSE
NewLmEncryptedWithNewNt	NULL

4. If the error status returned by the server is STATUS_NT_CROSS_ENCRYPTION_REQUIRED, compute NewNtEncryptedWithNewLm and call SamrChangePasswordUser with NtCrossEncryptionPresent as TRUE and NewNtEncryptedWithNewLm as the value computed in this step. All other values stay the same from the table in step 12.3.
5. If the error status returned by the server is STATUS_LM_CROSS_ENCRYPTION_REQUIRED, compute NewLmEncryptedWithNewNt and call SamrChangePasswordUser with LmCrossEncryptionPresent as TRUE and NewLmEncryptedWithNewNt as the value computed in this step. All other values stay the same from the table in step 12.3.

13. The client application must perform cleanup by closing all the handles it has opened during the session. This is done by calling SamrCloseHandle ([MS-SAMR] section 3.1.5.13.1, SamrCloseHandle (Opnum 1)) with SamHandle set to the handle the client application is attempting to close. The client application closes the handles in the reverse order they were received (namely, the user handle, domain handle, and server handle).

6.1.3.4 Final System State

The user account's password in the directory has been changed to the new value. No other state in the directory has changed.

6.1.4 Determine the Group Membership of a User

In this scenario, an Administrator determines the group membership of a user by querying the directory using the SAMR protocol. To perform this task, an Administrator runs a Client Application from a client computer, targeting a directory server in the Active Directory System. The Client Application queries the user's group membership using the SAMR protocol.

This scenario applies only to AD DS.

This scenario uses the SAMR protocol.

The scenario covers the use cases in section 3.3.4.2.4.

6.1.4.1 Initial System State

The Active Directory System must meet all preconditions specified in section 3.3.4.2.4.

6.1.4.2 Server Activity Diagram

The server activity diagram for this scenario is illustrated in the following figure. In this scenario, the client application connects to a directory server in the Active Directory System and communicates with it using the SAMR protocol. The activity diagram shows the activities the server performs as it transitions from an initial state to the state of a successful retrieval of group membership.

The server activities are as follows:

1. Upon receiving an incoming connection from the client application, the directory server interacts with the Windows Authentication Services, which authenticates the Administrator using the supplied credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task).

2. The server receives a request from the client application for a server handle to be used in later requests. The directory server validates that the user has the necessary access to acquire the server handle. If the validation fails, the directory server rejects the request. Otherwise, the server processes the request and returns a server handle to the client.

3. The server receives a request from the client application for a domain handle to be used in later requests. The server validates that the server handle associated with the request has the required access rights ([MS-SAMR] section 3.1.2.1, Standard Handle-Based Access Checks) and the directory server validates that the user has the necessary access to acquire the domain handle. If the validation fails the directory server rejects the request. Otherwise, the server processes the request and returns a domain handle to the client.

4. The server receives a request from the client application for a user handle to be used in later requests. The server validates that the domain handle associated with the request has the required access rights ([MS-SAMR] section 3.1.2.1, Standard Handle-Based Access Checks) and the directory server validates that the user has the necessary access to acquire the user handle. If the validation fails, the directory server rejects the request. Otherwise, the server processes the request and returns a user handle.

5. The server receives a request from the client to retrieve group membership of a user. This request contains a user handle for the account. The server retrieves the account's group membership and returns it to the client application. If the request is rejected by the server, the operation will be terminated with an appropriate error.

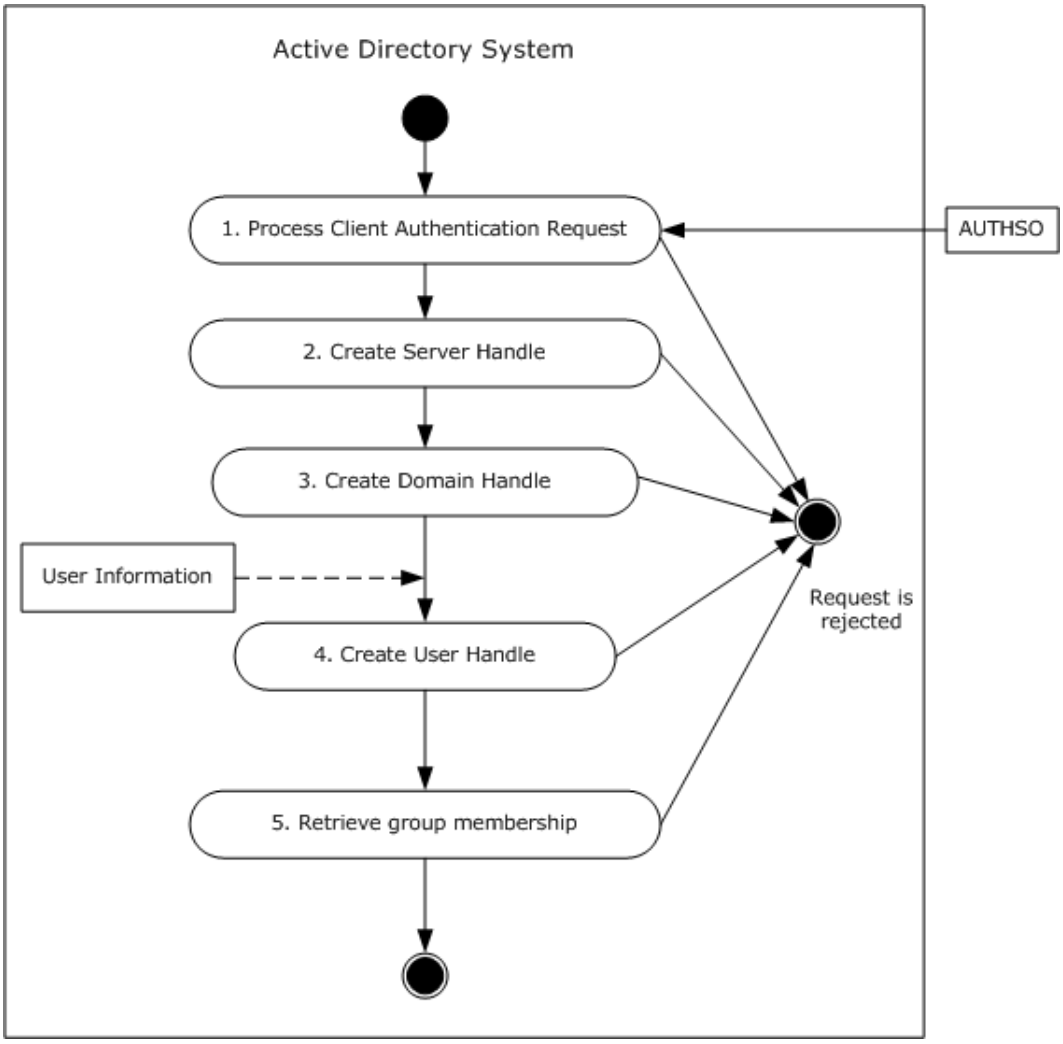

Figure 29: Server activity diagram for determining the group membership of a user

6.1.4.3 Sequence of Events

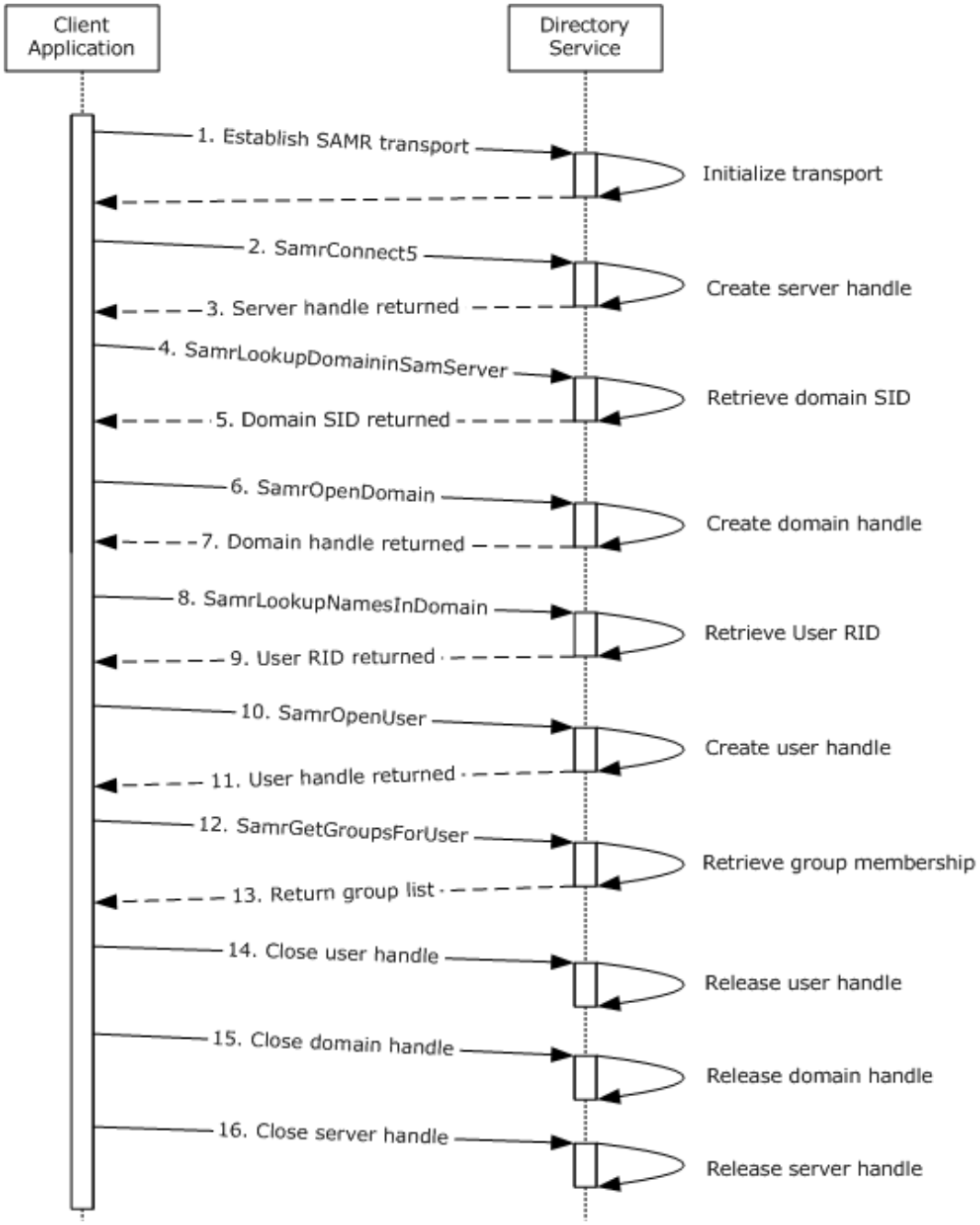

Figure 30: Message flow for determining the group membership of a user

Unless otherwise noted, all responses that include a return code contain a return code indicating that the operation was successfully performed.

1. The client binds to the SAMR endpoint on the server using a supported transport, as specified in [MS-SAMR] section 2.1, Transport.

2. The next step is to open a SAMR handle to the directory server. The client application sends a SamrConnect5 request ([MS-SAMR] section 3.1.5.1.1, SamrConnect5 (Opnum 64)) with the *DesiredAccess* parameter set to MAXIMUM_ALLOWED ([MS-SAMR] section 2.2.1.1, Common ACCESS_MASK Values) to the server requesting a server handle.

3. The server processes the request ([MS-SAMR] section 3.1.5.1.1, SamrConnect5 (Opnum 64)) and sends a response with the server handle to be used by later calls.

   Before continuing by opening a SAMR handle to the domain, the client must first know the security identifier (SID) of the domain. This is determined using steps 4-7.

4. The client application sends a SamrLookupDomainInSamServer request ([MS-SAMR] section 3.1.5.11.1, SamrLookupDomainInSamServer (Opnum 5)) using the server handle from step 2. In this request it specifies the domain name for the account.

5. The server processes the request ([MS-SAMR] section 3.1.5.11.1, SamrLookupDomainInSamServer (Opnum 5)) and returns the domain SID.

   The client application has now obtained the domain SID and can use it to open a SAMR handle to the domain.

6. The client application sends a SamrOpenDomain request ([MS-SAMR] section 3.1.5.1.5, SamrOpenDomain (Opnum 7)) using the server handle it previously obtained in step 2, the domain SID it obtained in step 5, and the *DesiredAccess* parameter set to MAXIMUM_ALLOWED ([MS-SAMR] section 2.2.1.1, Common ACCESS_MASK Values).

7. The server processes this request ([MS-SAMR] section 3.1.5.1.5, SamrOpenDomain (Opnum 7)) and returns a response with a domain handle.

   The client application must now get the relative identifier (RID) of the user so that it can open a user handle.

8. The client application sends a SamrLookupNamesInDomain request ([MS-SAMR] section 3.1.5.11.2, SamrLookupNamesInDomain (Opnum 17)). The request includes the user name and domain handle.

9. The server processes the request ([MS-SAMR] section 3.1.5.11.2, SamrLookupNamesInDomain (Opnum 17)) and returns the RID of the user account.

   The client application now has the user account RID and can use it to open a handle to the user.

10. The client application sends a SamrOpenUser request ([MS-SAMR] section 3.1.5.1.9, SamrOpenUser (Opnum 34)) with the *DesiredAccess* parameter set to MAXIMUM_ALLOWED ([MS-SAMR] section 2.2.1.1, Common ACCESS_MASK Values). The request includes the domain handle and the RID of the user.

11. The server processes the request ([MS-SAMR] section 3.1.5.1.9, SamrOpenUser (Opnum 34)) and returns a response with a user handle.

12. Now that the client application has a handle to the user it calls SamrGetGroupsForUser ([MS-SAMR] section 3.1.5.9.1, SamrGetGroupsForUser (Opnum 39)) to retrieve the group membership for the user.

13. The server processes the request ([MS-SAMR] section 3.1.5.9.1, SamrGetGroupsForUser (Opnum 39)) and returns a response with the list of groups that the user is a member of.

14.- 16. The client application must perform cleanup by closing all the handles it has opened during the session. This is done by calling SamrCloseHandle ([MS-SAMR] section 3.1.5.13.1, SamrCloseHandle (Opnum 1)) with SamHandle set to the handle the client application is attempting to close. The client application closes the handles in the reverse order they were received (namely, the user handle, domain handle, and server handle).

6.1.4.4 Final System State

The user's group membership has been returned to the client application. No state in the directory has changed.

6.1.5 Delete a User Account

In this scenario, an Administrator deletes a user account. This includes directory objects of class user as well as those of classes derived from user. One way this can be accomplished is by using the LDAP protocol. To perform this task, an Administrator runs a Client Application from a client computer and targets a directory server in the Active Directory System. The Client Application deletes the user by using the LDAP protocol.

This scenario applies only to AD DS.

This scenario uses the LDAP protocol.

The scenario covers the use cases in section 3.3.4.2.5.

6.1.5.1 Initial System State

The Active Directory System must meet all preconditions specified in section 3.3.4.2.5.

6.1.5.2 Server Activity Diagram

The server activity diagram for this scenario is illustrated in the following figure. In this scenario the client application connects to a directory server in the Active Directory System and communicates with it using the LDAP protocol. The activity diagram shows the activities the server performs as it transitions from an initial state to the state of a successful deletion of the user.

The server activities are as follows:

1. Upon receiving an LDAP bind request ([RFC2251] section 4.2, Bind Operation) from the Client Application, the Directory Server interacts with the Windows Authentication Services which authenticates the Administrator using the supplied credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task).

2. The server receives a request from the client application to search for an account’s distinguished name. The request contains the account name of the account. The server locates the DN of the account object and returns this information to the client application. If the request is rejected by the server the operation will be terminated with an appropriate error.

3. The server receives a request to delete the account with the specified DN. The directory server validates that the user has the necessary access to perform the operation. If the Administrator does not have the necessary access to perform the operation, the Directory Server rejects the LDAP delete request.

4. The server deletes the account.

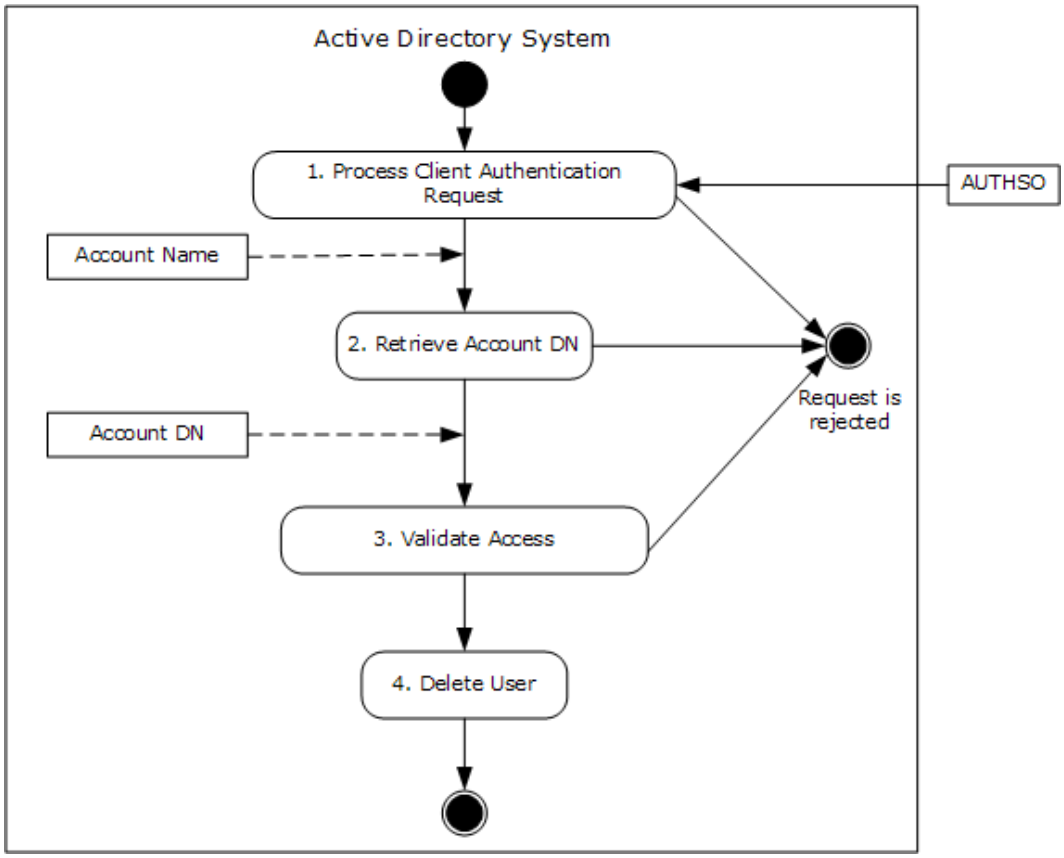

Figure 31: Server activity diagram for deleting a user account

6.1.5.3 Sequence of Events

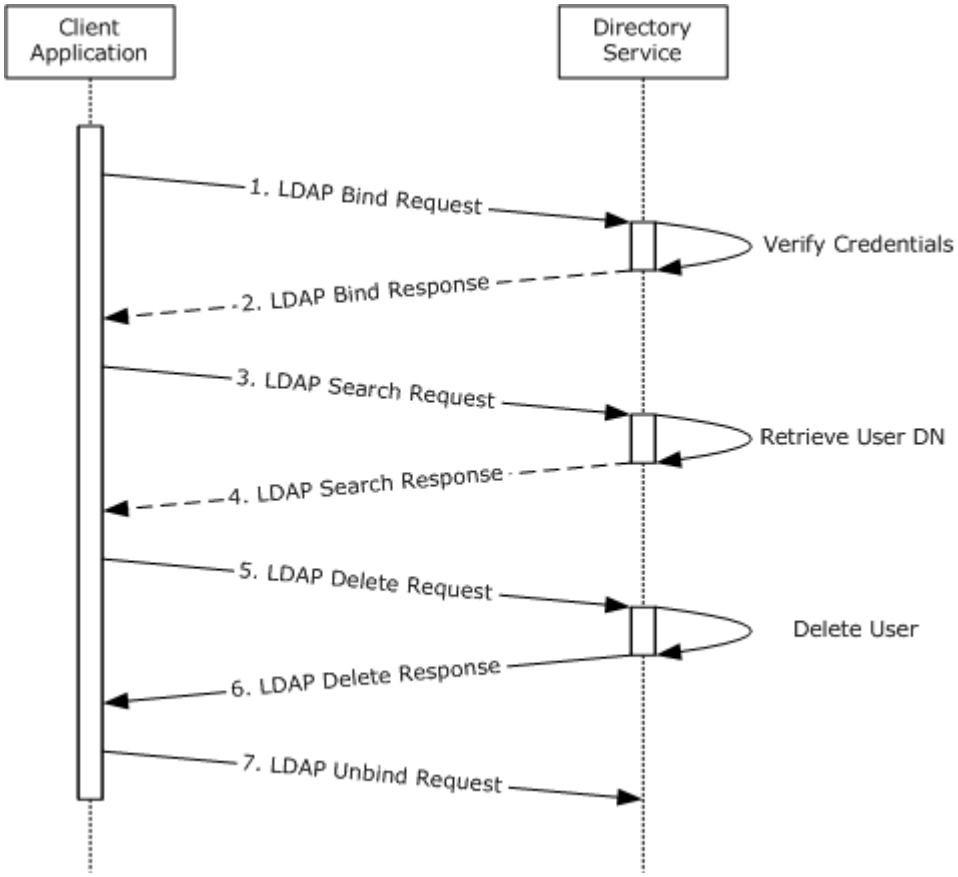

Figure 32: Message flow for deleting a user account

Unless otherwise noted, all responses that include a return code contain a return code indicating that the operation was successfully performed.

1. The Client Application launches and an LDAP bind request (see [RFC2251] section 4.2, Bind Operation) is sent to the directory server with credentials.

2. The directory server verifies the credentials (see [MS-AUTHSO] section 4, Interactive Domain Logon Task) and sends an LDAP bind response (see [RFC2251] section 4.2.3, Bind Response) to the client application.

3. The User interacts with the Client Application and provides details of the search criteria to be performed on the directory tree. The Client Application sends an LDAP search request (see [RFC2251] section 4.5.1, Search Request) to the server, querying the entire domain, starting at the root of the domain, looking for the user (see [MS-ADSC] section 2.263, Class User) and requesting the user's **distinguishedName** attribute.

4. The server sends back an LDAP search response (see [RFC2251] section 4.5.2, Search Result) containing the **distinguishedName** attribute for the user.

5. The client application sends an LDAP delete request (see [RFC2251] section 4.8, Delete Operation) to the server which contains the distinguishedName attribute of the user to be deleted.

6. The server processes the delete request (see [RFC2251] section 4.8, Delete Operation), verifies the processing rules and constraints, and then deletes the user object (see [MS-ADTS] section 3.1.1.5.5, Delete Operation). It then sends an LDAP delete response (see [RFC2251] section 4.8, Delete Operation) indicating success.

7. The Client Application sends an LDAP unbind request (see [RFC2251] section 4.3, Unbind Operation) to the server. The LDAP connection to the directory server is closed.

6.1.5.4 Final System State

The specified user object is successfully converted into a Tombstone or deleted object, depending whether the recycle bin optional feature is enabled or not, as described in [MS-ADTS] sections 3.1.1.5.5.1.1 and 3.1.1.5.5.1.2. No other state in the directory has changed.

6.1.6 Obtain a List of User Accounts Using the Web Services Protocols

One way to obtain a list of users in the Active Directory System is to query the directory using the Web Services protocols, specifically, WS-Enumeration. A client application can create a query with a supplied filter to locate accounts based on specific criteria, similar to an LDAP search operation.

To perform this task the client application sends a query to the directory service using the Web Services protocols. This scenario uses the WS-Enumeration protocol to communicate with the directory service.

This scenario covers the use case in section 3.3.4.1.2.

6.1.6.1 Initial System State

The Active Directory System must meet all preconditions specified in section 3.3.4.1.2. The system must support the Web Services protocols (see section 4.5).

6.1.6.2 Server Activity Diagram

The server activity diagram for this scenario is illustrated in the following figure. In this scenario, the Client Application connects to a directory server in the Active Directory System and communicates with it using the WS-Enumeration protocol. The activity diagram shows the activities the directory server performs as it transitions from an initial state to the state of successfully returning the results of the specified query.

1. Upon receiving an incoming connection from the Client Application, the Directory Server interacts with the Windows Authentication Services, which authenticates the Administrator using the supplied credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task).

2. The server receives an enumeration request from the client application. The request contains the query used to create the enumeration. The directory server validates that the user has the necessary access to perform the operation. If the client application does not have the necessary access to perform the operation, the Directory Server rejects the request.

3. The server processes the request and creates an enumeration context. It returns an identifier for this context to the client for the client to use in later requests. If the server cannot create an enumeration context, then the operation will be terminated with an appropriate error.

4. The server receives a WS-Enumeration Pull request from the client application. The server retrieves results of the query and returns a portion of the enumerated data to the client as specified by the pull request. The client application continues to send pull requests to the server until it has retrieved all data from the enumeration. If the request is rejected by the server, the operation will be terminated with an appropriate error.

5. The server receives a WS-Enumeration Release request from the client application. The server terminates the enumeration and releases any allocated resources used to service the previous requests. A response is sent to the client application indicating that the enumeration has been successfully released. If the request is rejected by the server, the operation will be terminated with an appropriate error.

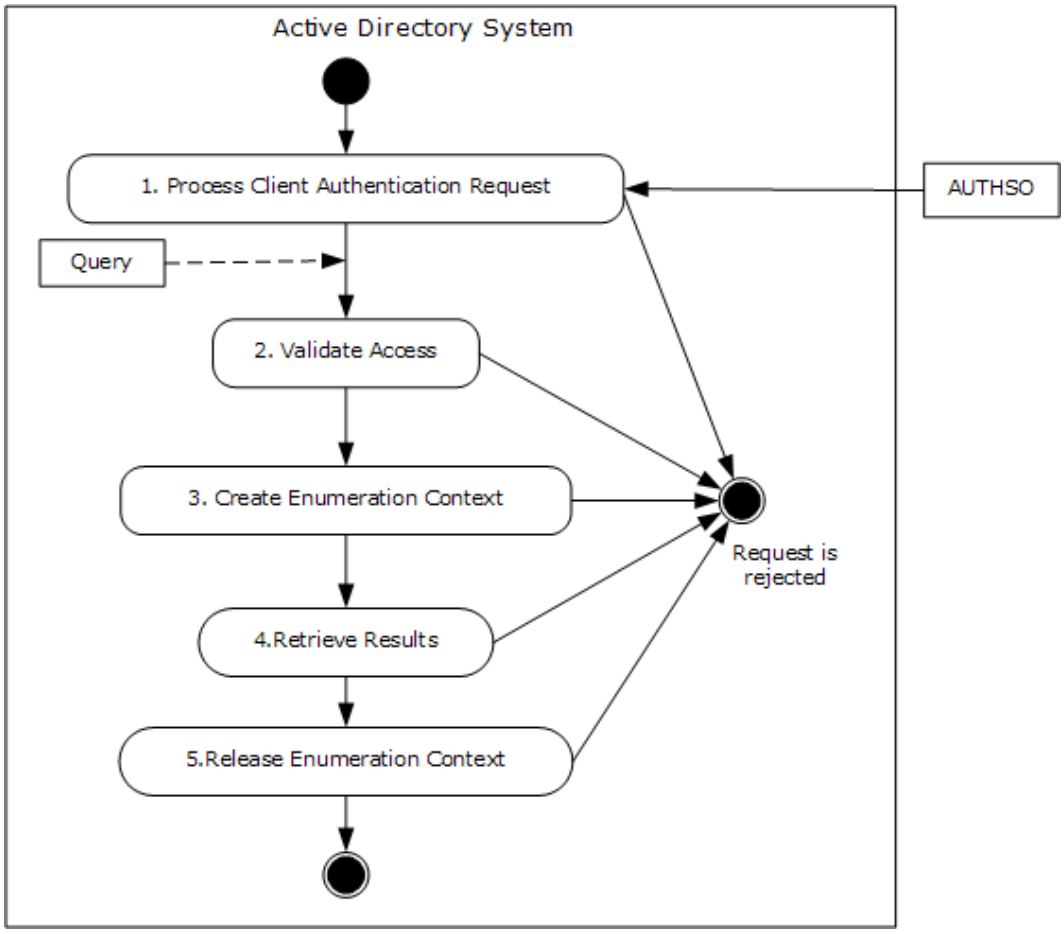

Figure 33: Server activity diagram for obtaining a list of user accounts using the Web Services protocols

6.1.6.3 Sequence of Events

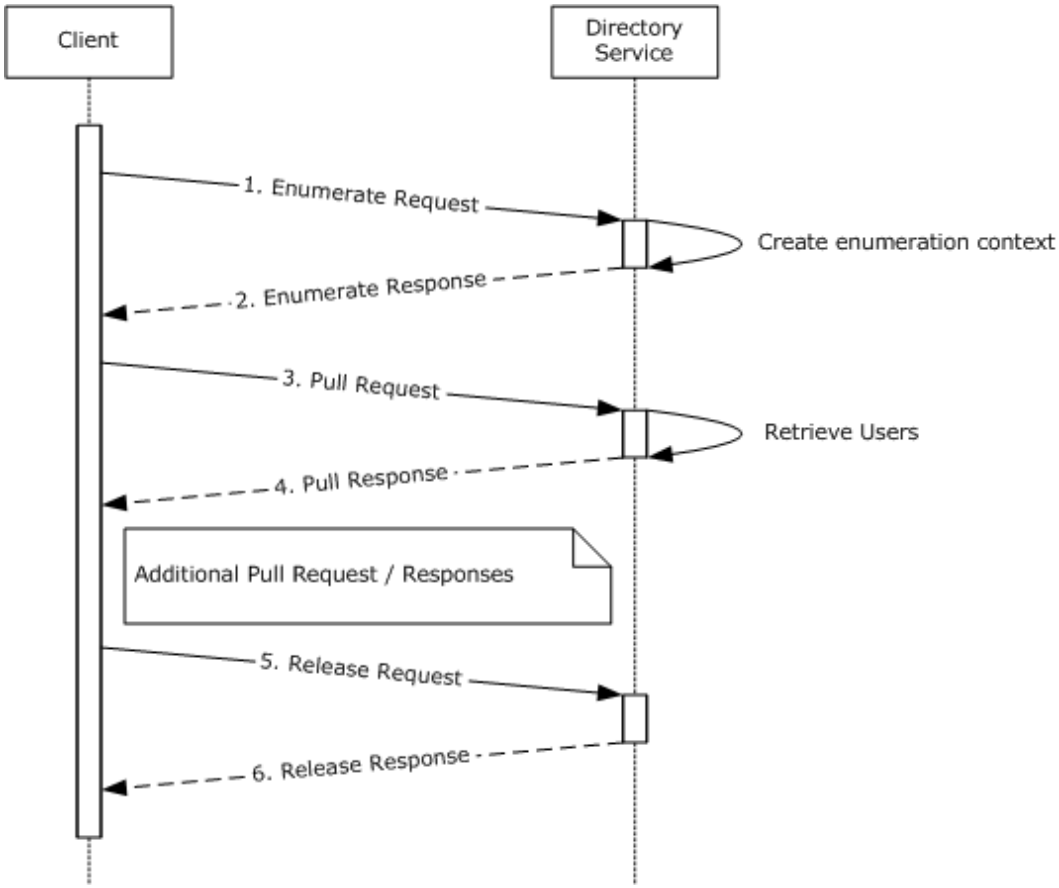

Figure 34: Communication flow for obtaining a list of user accounts using the Web Services protocols

The client application establishes a connection to the directory service using the net.tcp transport [MC-NMF]. In this scenario all communications sent and received via this transport use the SOAP 1.2 and WS-Addressing protocols. Unless otherwise noted, all responses that include a return code contain a return code indicating that the operation was successfully performed.

1. The client application sends a SOAP message containing a WS-Enumeration Enumerate request ([WSENUM] section 3.1, Enumerate). The filter section of the Enumerate request contains the query the client wishes to use to identify which accounts should be returned ([MS-WSDS] section 3.1.4.1.1.1, adlq:LdapQuery). The filter also includes (objectClass=user) to indicate that only user directory objects should be returned.

The request also contains a Selection element that indicates which attributes of the queried user objects are to be returned to the client ([MS-WSDS] section 3.1.4.1.1.2, ad:Selection).

2. The directory server processes the request and generates an **enumeration context** ([WSENUM] section 3, Enumeration Messages) that can be used by the client application in further requests. It then returns a SOAP message containing a WS-Enumeration Enumeration response with the **enumeration context** and the expiration time of the request.

3. Now that the client application has set up the enumeration it can begin requesting data. It sends a SOAP message containing a Pull request ([WSENUM] section 3.2, Pull). The request contains the previously-returned **enumeration context**, along with optional values such as how many items (directory objects) the server is to return to the client application in the Pull response.

4. The server retrieves the matching user objects ([MS-ADTS] section 3.1.1.3.1.3, Search Operations) from the directory. It returns a SOAP message containing a Pull response to the client application ([WSENUM] section 3.2, Pull). This response contains the objects that match the client application's query, including the attributes of those objects requested by the client in the Enumerate request that established the **enumeration context**.

The client application repeats sending Pull requests and processing Pull responses as needed until the directory service indicates that there is no additional data to retrieve ([WSENUM] section 3.2, Pull).

5. Having retrieved all the data requested the client application terminates the enumeration by sending a SOAP message containing a Release request ([WSENUM] section 3.5, Release). This tells the server the client application is finished with the enumeration content and that the server can release any server side resources it has allocated to process the enumeration.

6. The server performs any necessary processing of the Release request and returns a Release response to the client.

6.1.6.4 Final System State

The requested information for the user object(s) are returned to the Client Application. No state in the directory has changed.

6.1.7 Obtain a List of User Accounts Using the LDAP Protocol

Obtaining a list of user accounts in the Active Directory System can be achieved by querying the directory using the LDAP protocol. A client can create a query with a supplied filter to locate accounts based on specific criteria. To perform this task, a user runs a Client Application from a client computer that sends a query targeting a directory server in the Active Directory System.

This scenario covers the use case in section 3.3.4.1.2.

6.1.7.1 Initial System State

The Active Directory System must meet all preconditions specified in section 3.3.4.1.2.

6.1.7.2 Server Activity Diagram

The server activity diagram for this scenario is illustrated in the following figure. In this scenario, the client application connects to a directory server in the Active Directory System and communicates with it using the LDAP protocol. The activity diagram shows the actions the directory server takes as it transitions from an initial state to the state of successful retrieval of user account(s) information.

The server activities are as follows:

1. Upon receiving an LDAP bind request ([RFC2251] section 4.2, Bind Operation) from the Client Application, the Directory Server interacts with the Windows Authentication Services, which authenticates the Administrator using the supplied credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task).

2. The Directory Server receives the LDAP search request ([RFC2251] section 4.5.1, Search Request) containing the search criteria for the user(s) information to retrieve. The Directory Server verifies the User's access-control rights to see the user object(s) and requested attributes as outlined in [MS-ADTS] sections 3.1.1.4.3, Access Checks, and 3.1.1.4.4, Extended Access Checks. If the security constraints are not satisfied, the search request will be terminated with an appropriate error.

3. The Directory Server receives a request to retrieve the users specified in the request. The Directory Server processes the request and returns the user information to the Client Application. If the user does not have the access-control rights to see a subset of the user objects, these objects are not returned to the search result set.

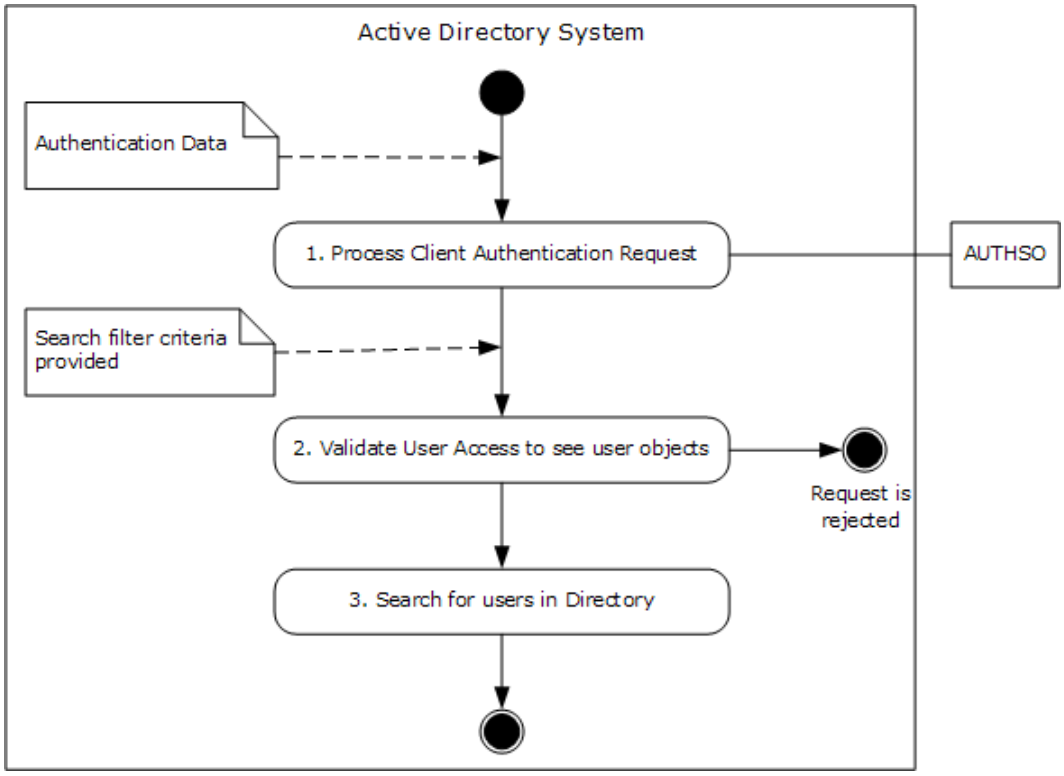

Figure 35: Server activity diagram for obtaining a list of user accounts using the LDAP protocol

6.1.7.3 Sequence of Events

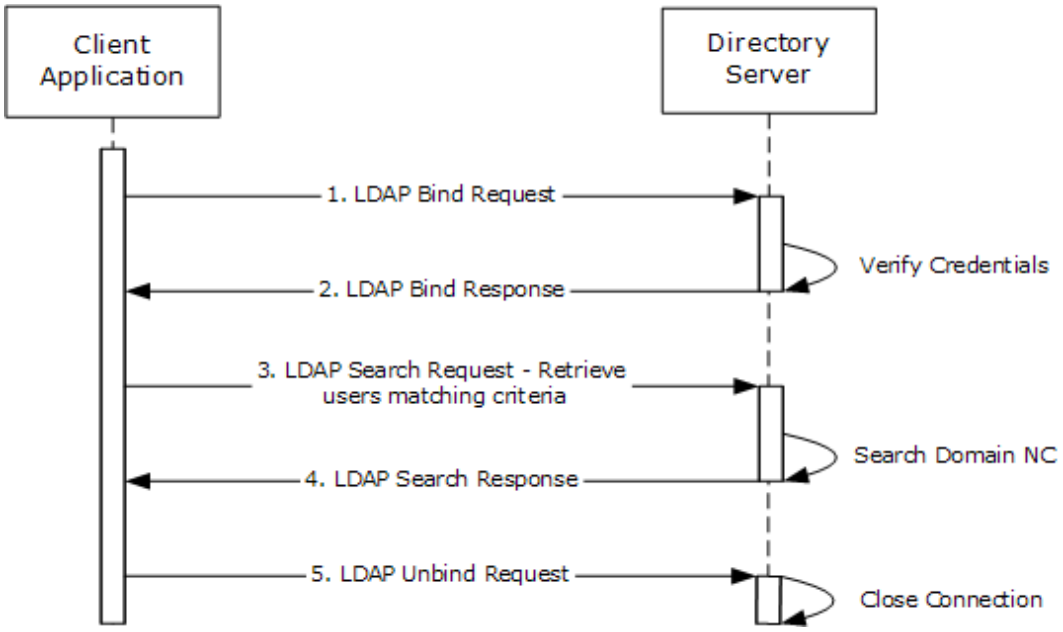

Figure 36: Message flow for obtaining a list of user accounts using the LDAP protocol

Unless otherwise noted, all responses that include a return code contain a return code indicating that the operation was successfully performed.

1. The Client Application launches and a LDAP bind request ([RFC2251] section 4.2, Bind Operation) is sent to the directory server with credentials.

2. The directory server verifies the credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task) and sends a LDAP bind response ([RFC2251] section 4.2.3, Bind Response) to the Client Application.

3. The User interacts with the Client Application and provides details of the search criteria to be performed on the directory tree. The Client Application sends a LDAP search request ([RFC2251] section 4.5.1, Search Request) to the directory, querying the entire domain, starting at the root of the domain, looking for users ([MS-ADSC] section 2.263, Class User) and requesting all attributes.

4. The server sends back a LDAP search response ([RFC2251] section 4.5.2, Search Result) containing the list of users underneath the domain **NC**. The Client Application organizes this information and displays to the user. Search filters and results are additionally validated by the server's processing rules and constraints specified in [MS-ADTS] sections 3.1.1.3.1.3, Search Operations, and 3.1.1.3.4.6, LDAP Policies.

5. The Client Application sends an LDAP unbind request ([RFC2251] section 4.3, Unbind Operation) to the server. The LDAP connection to the directory server is closed.

6.1.7.4 Final System State

The requested information for the user object(s) are returned to the Client Application. No state in the directory has changed.

6.1.8 Manage Groups and Their Memberships

This section discusses the process of creating groups, adding members to that group, and querying its membership. This process is illustrated by the state transitions of the directory client and the message flow between client and server.

This scenario covers the use cases in sections 3.3.4.2.6, 3.3.4.2.7, and 3.3.4.2.8.

6.1.8.1 Initial System State

The Active Directory System must meet all preconditions specified in section 3.3.4.1.3.

6.1.8.2 Server Activity Diagram

The server activity diagram for this scenario is illustrated in the following figure. In this scenario, the Client Application connects to a directory server in the Active Directory System and communicates with it using the LDAP and RPC protocols. The activity diagram shows the activities the server performs as it transitions from an initial state to the state of a successfully querying group membership.

The server activities are as follows:

1. Upon receiving an LDAP bind request ([RFC2251] section 4.2, Bind Operation) from the Client Application, the directory server interacts with the Windows Authentication Services, which authenticates the Administrator using the supplied credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task).

2. The server receives a group creation request from the client application. This request contains the name, type, and scope for the new group. The directory server validates that the user has the necessary access to perform the operation. If the user does not have the necessary access to perform the operation, the directory server rejects the request.

3. The server validates the group creation request as outlined in [MS-SAMR] sections 3.1.1.6, Attribute Constraints for Originating Updates, and 3.1.1.8.4, sAMAccountName. If the validation fails, the operation will be terminated with an appropriate error.

4. The server creates the group object in the directory.

5. The server receives a request to translate an account name to distinguished name. The server processes the request and returns the translated name to the client application. If the request is rejected by the server, the operation will be terminated with an appropriate error.

6. The server receives a request to add a user to the group. The server processes the request and adds the user to the group. If the request is rejected by the server, the operation will be terminated with an appropriate error.

7. The server receives a request to retrieve the group membership for a user. The server processes the request and returns the group membership information to the client application.

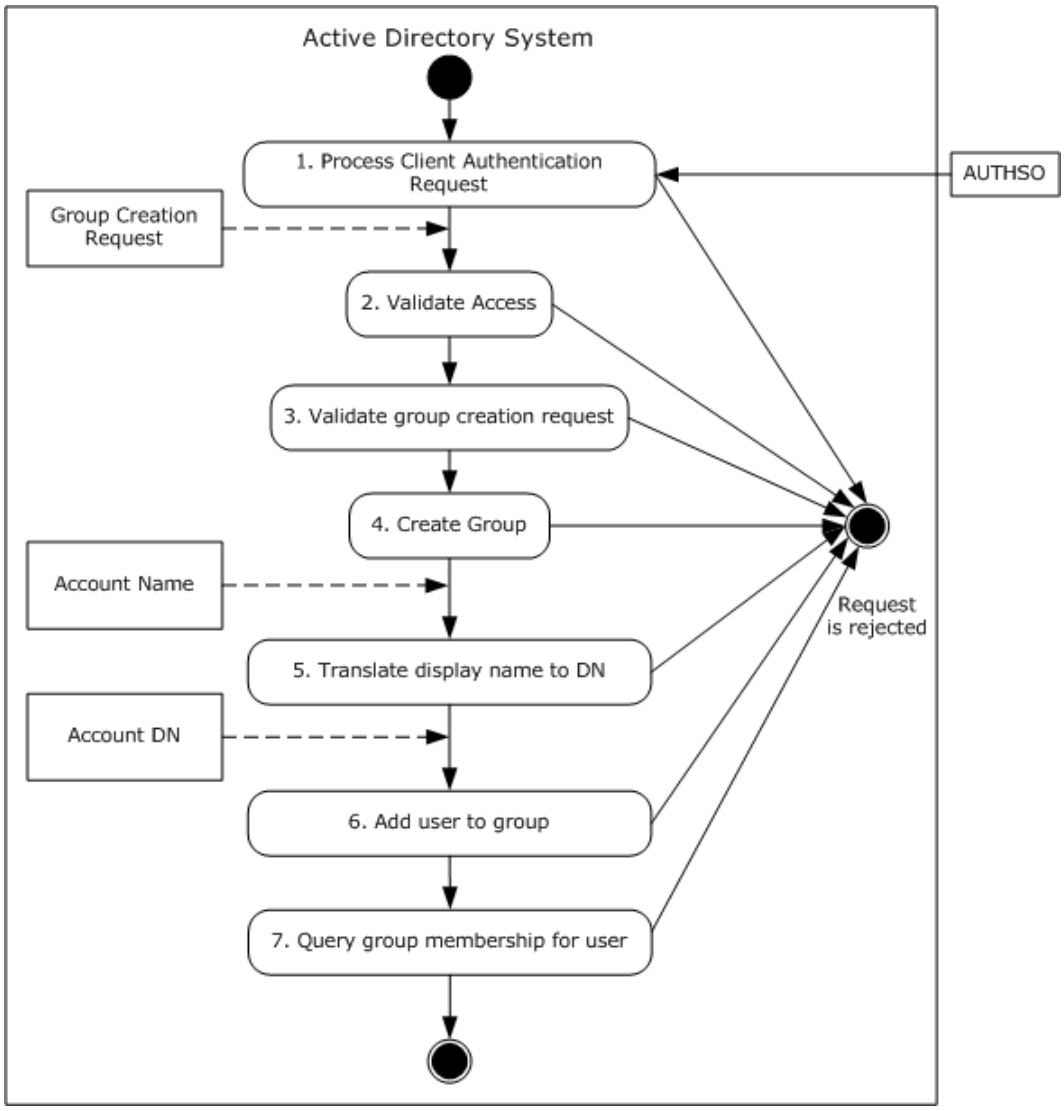

Figure 37: Server activity diagram for managing groups and their memberships

6.1.8.3 Sequence of Events

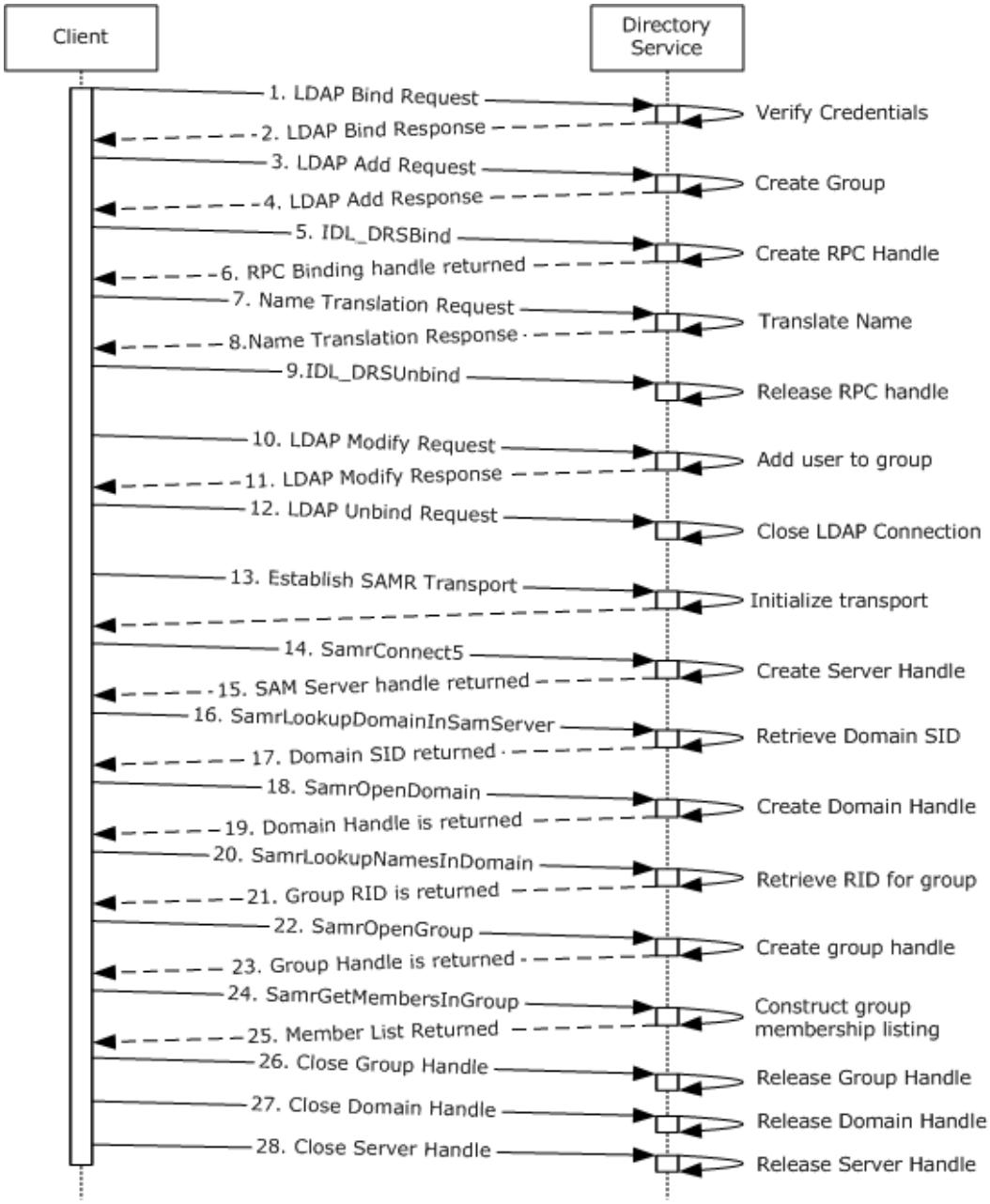

Figure 38: Communication flow for managing groups and their memberships

Unless otherwise noted, all responses that include a return code contain a return code that indicates that the operation was successfully performed.

1. The client application establishes an LDAP connection to the directory server. An LDAP bind request ([RFC2251] section 4.2, Bind Operation) is sent to the directory server with the credentials of an administrator.

2. The directory server verifies the credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task) and sends an LDAP bind response ([RFC2251] section 4.2.3, Bind Response) to the client application.

3. An LDAP Add Request ([RFC2251] section 4.7, Add Operation) is sent to the server. The LDAP add operation contains the distinguishedName, samAccountName, objectClass, and groupType for the new group.

4. The server processes the add request ([RFC2251] section 4.7, Add Operation) and performs validation, as described in [MS-ADTS] sections 3.1.1.5.1, General, and 3.1.1.5.2, Add Operation. The server then sends an LDAP add response that indicates success.

5. The client requests an RPC binding handle to establish a connection with the directory server by using the Directory Replication Service (DRS) Remote protocol, as defined in [MS-DRSR] section 4.1.3, IDL_DRSBind (Opnum 0).

6. The server processes the bind request and sends a response with an RPC Binding handle.

7. The client sends a request for name translation to the server using the RPC binding handle, as defined in [MS-DRSR] section 4.1.4, IDL_DRSCrackNames (Opnum 12). The request specifies the name to translate (the user's NT4 account name in this example), its format (NT4 account name), and the desired format for the response (distinguished name (DN)).

8. The server processes the request and returns the translated name (the user's DN), as defined in [MS-DRSR] section 4.1.4.3, Server Behavior of the IDL_DRSCrackNames Method.

9. The client requests to release the RPC binding handle it received in step 5, as defined in [MS-DRSR] section 4.1.25, IDL_DRSUnbind (Opnum 1). The server processes that request as specified in [MS-DRSR] section 4.1.25.1, Server Behavior of the IDL_DRSUnbind Method.

10. An LDAP modify request is sent to the server to add the user to the group, as specified in [RFC2251] section 4.6, Modify Operation. The distinguished name of the user is added to the multi-valued linked attribute "member" of the group object (see [MS-ADA2] section 2.43, Attribute member).

11. The server processes the request and sends a response ([RFC2251] section 4.6, Modify Operation).

12. The client sends an LDAP unbind request ([RFC2251] section 4.3, Unbind Operation) to the server. The LDAP connection to the directory server is closed.

    At this point the group has been created with the attributes desired by the administrator, and a user has been added as a member to that group with the only knowledge of that user's NT4 account name. The remainder of this scenario is about displaying the resultant group's member list.

13. The client binds to the SAMR endpoint on the server using a supported transport, as specified in [MS-SAMR] section 2.1, Transport.

14. The next step is to open a SAMR handle to the directory server. The client application sends a SamrConnect5 request ([MS-SAMR] section 3.1.5.1.1, SamrConnect5 (Opnum 64)) with the *DesiredAccess* parameter set to MAXIMUM_ALLOWED ([MS-SAMR] section 2.2.1.1, Common ACCESS_MASK Values) to the server to request a server handle.

15. The server processes the request and sends a response with the server handle to be used by later calls.

Before continuing by opening a SAMR handle to the domain, the client must first know the security identifier (SID) of the domain. The SID is determined using steps 16-17.

16. The client application sends a SamrLookupDomainInSamServer request ([MS-SAMR] section 3.1.5.11.1, SamrLookupDomainInSamServer (Opnum 5)) using the server handle from step 14. In this request, the client specifies the domain name for the account.

17. The server processes the request and returns the domain SID.

    The client application has now obtained the domain SID and can use it to open a SAMR handle to the domain.

18. The client application sends a SamrOpenDomain request ([MS-SAMR] section 3.1.5.1.5, SamrOpenDomain (Opnum 7)) by using the server handle that it previously obtained in step 14, the domain SID it obtained in step 17, and the *DesiredAccess* parameter set to MAXIMUM_ALLOWED ([MS-SAMR] section 2.2.1.1, Common ACCESS_MASK Values).

19. The server processes this request and returns a response with a domain handle.

    The client application must now get the relative identifier (RID) of the group so that it can open a group handle.

20. The client application sends a SamrLookupNamesInDomain request ([MS-SAMR] section 3.1.5.11.2, SamrLookupNamesInDomain (Opnum 17)). The request includes the group name and domain handle.

21. The server processes the request and returns the RID of the group.

    The client application now has the group RID and can use it to open a handle to the group.

22. The client application sends a SamrOpenGroup request ([MS-SAMR] section 3.1.5.1.7, SamrOpenGroup (Opnum 19)). The request includes the domain handle, the RID of the group, and the *DesiredAccess* parameter set to MAXIMUM_ALLOWED ([MS-SAMR] section 2.2.1.1, Common ACCESS_MASK Values).

23. The server processes the request and returns a response with a group handle.

24. Now that the client application has a handle to the user it calls SamrGetMembersInGroup ([MS-SAMR] section 3.1.5.8.3, SamrGetMembersInGroup (Opnum 25)) to retrieve the group's member list.

25. The server processes the list and returns the member list for the group.

    The client application now knows the member list for the group and can present it to the administrator.

26. -28. The client application must perform cleanup by closing all the handles that it has opened during the session. This is done by calling SamrCloseHandle ([MS-SAMR] section 3.1.5.13.1, SamrCloseHandle (Opnum 1)) with SamHandle set to the handle that the client application is attempting to close. The client application closes the handles in the reverse order that they were received (namely, the group handle, domain handle, and server handle).

6.1.8.4 Final System State

The new group object has been provisioned in the directory and updated with the membership specified. No other state in the directory has changed.

6.1.9 Delete a Group

In this scenario, a user deletes a security group which is transformed into a Tombstone, [MS-ADTS] section 3.1.1.5.5.1.1. One way this can be accomplished is by using the LDAP protocol. To perform this task, a user runs a client **application** from a client computer, targeting a directory server in the Active Directory System and deletes a security group.

This scenario covers the use case in section 3.3.4.1.4.

6.1.9.1 Initial System State

The Active Directory System must meet all preconditions specified in section 3.3.4.1.4.

6.1.9.2 Server Activity Diagram

The server activity diagram for this state scenario is illustrated in the following figure. In this scenario, the client application connects to a directory server in the Active Directory System and communicates with it using the LDAP protocol. The activity diagram shows the activities the server performs as it transitions from an initial state to the state of the security group becoming a Tombstone or deleted-object, depending whether the recycle bin optional feature is enabled or not, as described in [MS-ADTS] sections 3.1.1.5.5.1.1, Tombstone Requirements, and 3.1.1.5.5.1.2, Deleted-Object Requirements.

The server activities are as follows:

1. Upon receiving an LDAP bind request ([RFC2251] section 4.2, Bind Operation) from the client application, the directory server interacts with the Windows Authentication Services, which authenticates the **administrator** using the supplied credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task).

2. The directory server receives a request from the client application to delete ([RFC2251] section 4.8, Delete Operation) a security group. The request contains the name of the group. The directory server verifies the Administrator's access-control rights to see the group object as outlined in [MS-ADTS] sections 3.1.1.4.3, Access Checks, and 3.1.1.4.4, Extended Access Checks. If the security constraints are not satisfied the deletion request will be terminated with an appropriate error. The directory server proceeds to verify the Administrator's access-control rights to delete the group object as outlined in [MS-ADTS] sections 3.1.1.5.5.3, Protected Objects, and 3.1.1.5.5.4, Security Considerations. If the security constraints are not satisfied, the deletion request will be terminated with an appropriate error.

3. The directory server verifies the constraints for delete operations are satisfied as outlined in [MS-ADTS] sections 3.1.1.5.1, General, and 3.1.1.5.5.5, Constraints. If the constraints are not satisfied, the delete request will be terminated with an appropriate error.

4. The security group is deleted in the directory and additional processing is done on the deleted object as outlined in [MS-ADTS] section 3.1.1.5.5.6, Processing Specifics.

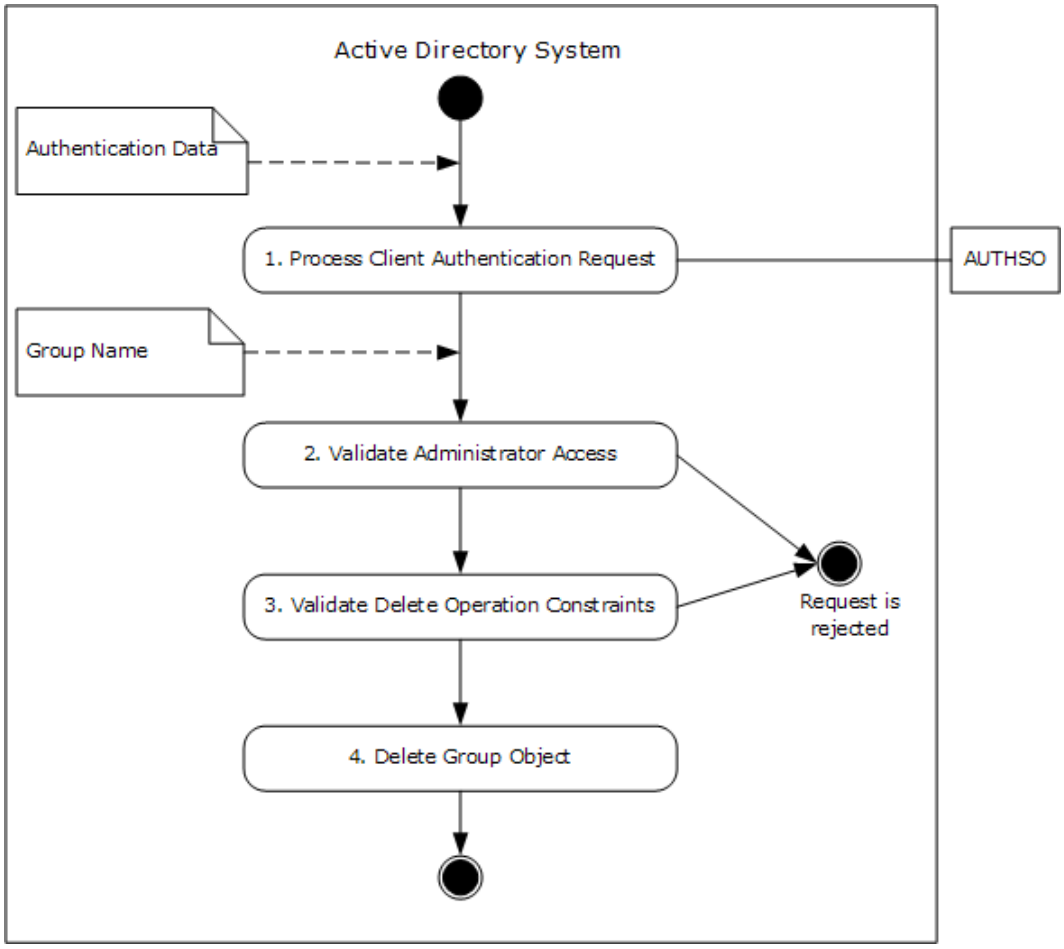

Figure 39: Server activity diagram for deleting a security group

6.1.9.3 Sequence of Events

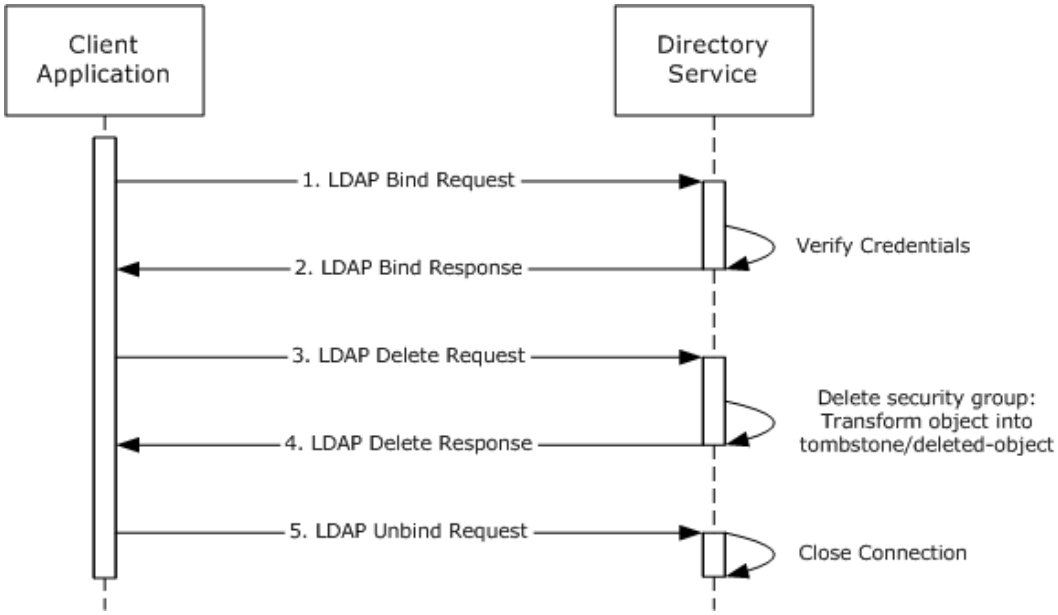

Figure 40: Message flow for deleting a security group

Unless otherwise noted, all responses that include a return code contain a return code indicating that the operation was successfully performed.

1. The client application launches and a LDAP bind request ([RFC2251] section 4.2, Bind Operation) is sent to the directory server with credentials.

2. The directory server verifies the credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task) and sends a LDAP bind response ([RFC2251] section 4.2.3, Bind Response) to the client application.

3. The User interacts with the client application and identifies the security group to be deleted. An LDAP delete request ([RFC2251] section 4.8, Delete Operation) is sent to the server. The LDAP delete operation contains the distinguishedName of the security group to be deleted.

4. The server processes the delete request ([RFC2251] section 4.8, Delete Operation) and verifies the processing rules and constraints and the tombstone transformation operations on the security group as described in [MS-ADTS] sections 3.1.1.5.1, General, and 3.1.1.5.5, Delete Operation. It then sends an LDAP delete response ([RFC2251] section 4.8, Delete Operation) indicating success.

5. The client application sends an LDAP unbind request ([RFC2251] section 4.3, Unbind Operation) to the server. The LDAP connection to the directory server is closed.

6.1.9.4 Final System State

The security group object is successfully converted into a Tombstone or deleted-object, depending whether the recycle bin optional feature is enabled or not, as described in [MS-ADTS] sections 3.1.1.5.5.1.1 and 3.1.1.5.5.1.2. No other state in the directory has changed.

6.1.10 Extend the Schema to Support an Application by Adding a New Class

In this scenario, an Administrator extends the schema by adding a class required by an application. This is accomplished using the LDAP protocol. To perform this task, an Administrator runs a Client Application from a client computer targeting a Directory Server which is the Schema Master FSMO role in the Active Directory System. The Client Application adds a class and sets its properties using the LDAP protocol.

This scenario uses the LDAP protocol.

The scenario covers the use cases in section 3.3.4.3.1.

6.1.10.1 Initial System State

The Active Directory System must meet all preconditions specified in section 3.3.4.3.1.

6.1.10.2 Server Activity Diagram

The various activities performed by the server in response to client requests are illustrated in the following figure. The client communicates with the server by using the LDAP Protocols.

The server performs various activities upon receiving inputs from the client as follows:

1. Upon receiving an LDAP bind request ([RFC2251] section 4.2, Bind Operation) from the Client Application, the directory server interacts with the Windows Authentication Services, which authenticates the Administrator using the supplied credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task).

2. Upon receiving an LDAP add request ([RFC2252] section 4.7, Add Operation) from the Client Application, the directory server validates if the Administrator has the necessary access to perform the operation. If the Administrator does not have the necessary access to perform the operation, the directory server rejects the LDAP add request.

3. The directory server validates that it owns the Schema Master FSMO role ([MS-ADTS] section 3.1.1.2.5, Schema Modifications) to perform the operation. If the directory server does not own the Schema Master FSMO role, the directory server rejects the LDAP add request.

4. The directory server validates the constraints to perform the operation as outlined in [MS-ADTS] section 3.1.1.2.5, Schema Modifications. If it fails to validate any of the constraints, the directory server rejects the LDAP add request.

5. If all the constraints pass, the directory server performs to add the new class to the schema.

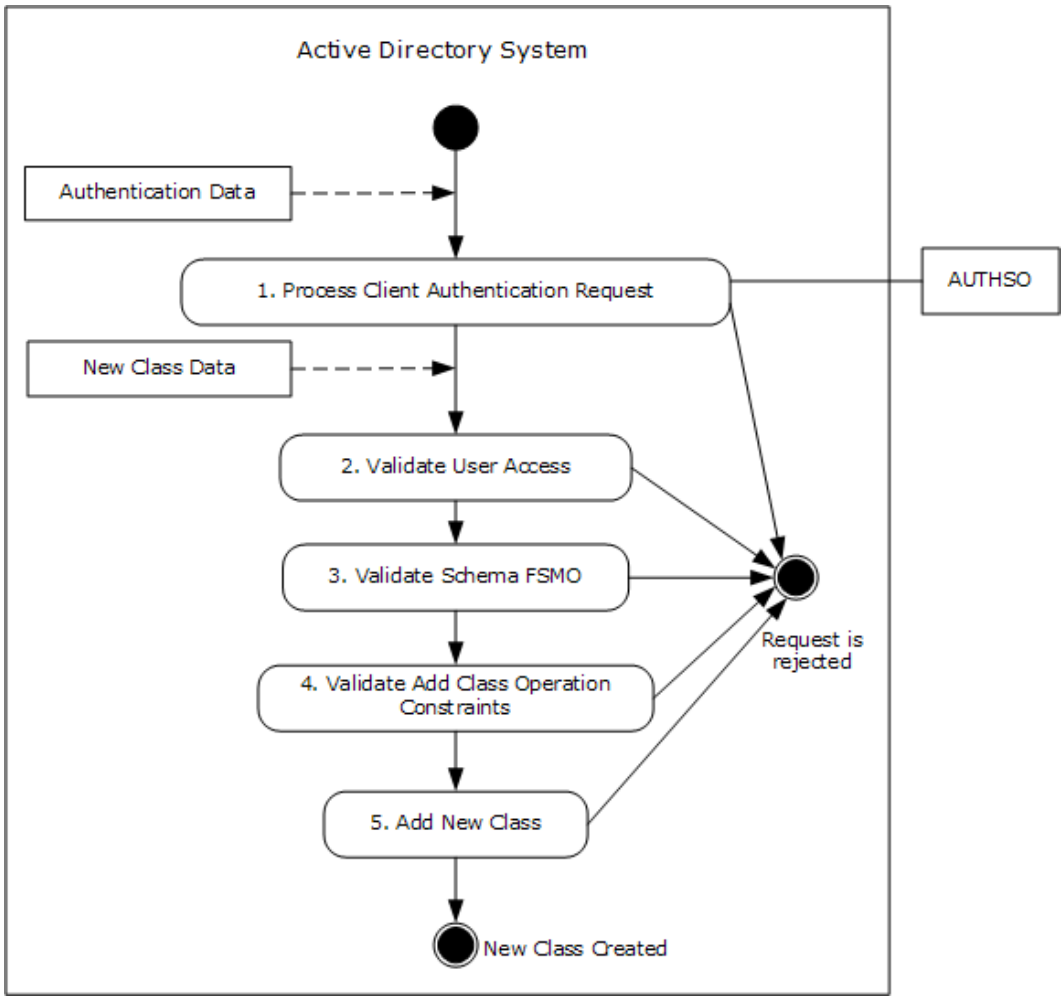

Figure 41: Server activity diagram for extending the schema by adding a class

6.1.10.3 Sequence of Events

The diagram that follows illustrates the messages that are exchanged between a Client and a Directory Server when adding a class to the schema is successful.

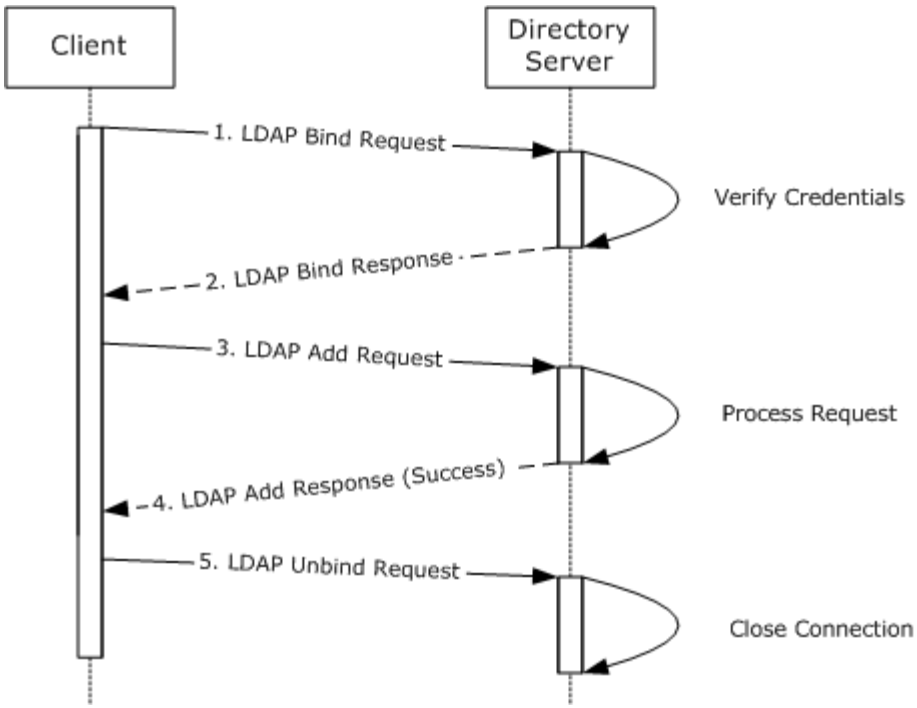

Figure 42: Message flow for extending the schema by adding a class

The sequence of events is described in the following steps.

1. The Client Application establishes an LDAP connection to the Directory Server. An LDAP bind request ([RFC2251] section 4.2, Bind Operation) is sent to the Directory Server with credentials of the Administrator.

2. The Directory Server verifies the credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task) and sends an LDAP bind response ([RFC2251 section 4.2.3, Bind Response) to the Client Application.

3. The Client Application sends an LDAP add request ([RFC2251] section 4.7, Add Operation) to the server. The request contains the values of the mandatory attributes ([MS-ADTS] section 3.1.1.2.4.8, Class classSchema) for the new object of class classSchema.

4. The server verifies all the processing rules and constraints ([MS-ADTS] sections 3.1.1.5.1, General, 3.1.1.5.2, Add Operation, and 3.1.1.2.5, Schema Modifications). Upon success, the class is added to the schema and the server sends an LDAP add response ([RFC2252] section 4.7, Add Operation) indicating that the object creation was successful.

5. The Client Application closes the LDAP connection by sending an LDAP unbind request ([RFC2251] section 4.3, Unbind Operation) to the directory server.

6.1.10.4 Final System State

A new class has been added to the schema.

6.1.11 Extend the Schema to Support an Application by Adding a New Attribute

In this scenario, an Administrator extends the schema by adding an attribute required by an application. This is accomplished using the LDAP protocol. To perform this task, an Administrator runs a Client Application from a client computer targeting a Directory Server which is the Schema Master FSMO role in the Active Directory System. The Client Application adds an attribute and sets its properties using the LDAP protocol.

This scenario uses the LDAP protocol.

The scenario covers the use cases in section 3.3.4.3.2.

6.1.11.1 Initial System State

The Active Directory System must meet all preconditions specified in section 3.3.4.3.2.

6.1.11.2 Server Activity Diagram

The various activities performed by the server in response to client requests are illustrated in the following figure. The client communicates with the server by using the LDAP Protocols.

The server performs various activities upon receiving inputs from the client as follows:

1. Upon receiving an LDAP bind request ([RFC2251] section 4.2, Bind Operation) from the Client Application, the directory server interacts with the Windows Authentication Services, which authenticates the Administrator using the supplied credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task).

2. Upon receiving an LDAP add request ([RFC2252] section 4.7, Add Operation) from the Client Application, the directory server validates if the Administrator has the necessary access to perform the operation. If the Administrator does not have the necessary access to perform the operation, the directory server rejects the LDAP add request.

3. The directory server validates that it owns the Schema Master FSMO role ([MS-ADTS] section 3.1.1.2.5, Schema Modifications) to perform the operation. If the directory server does not own the Schema Master FSMO role, the directory server returns a referral ([MS-ADTS] section 3.1.1.1.11, FSMO Roles) to the directory server which owns the Schema Master FSMO role.

4. The directory server validates the constraints to perform the operation as outlined in [MS-ADTS] section 3.1.1.2.5, Schema Modifications. If it fails to validate any of the constraints, the directory server rejects the LDAP add request.

5. If all the constraints pass, the directory server performs to add the new attribute to the schema.

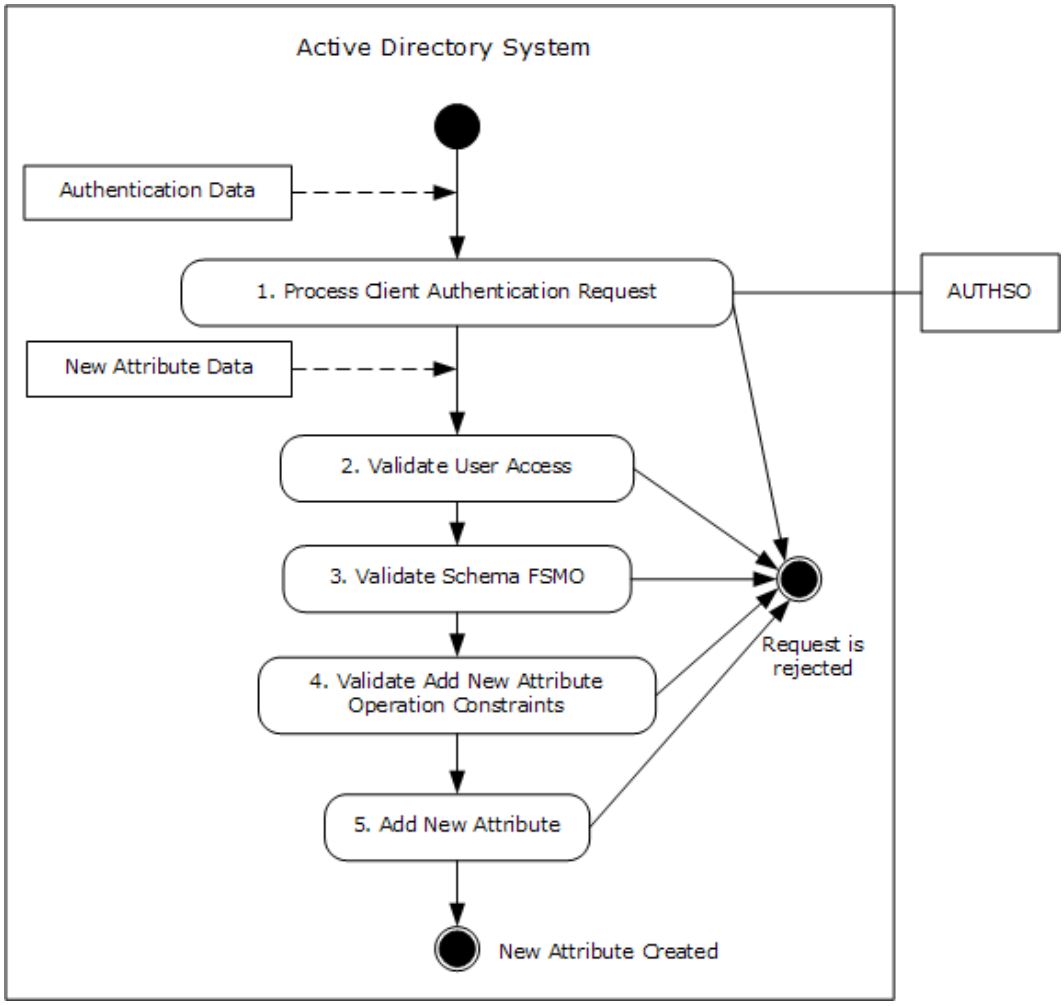

Figure 43: Server activity diagram for extending the schema by adding a new attribute

6.1.11.3 Sequence of Events

The diagram that follows illustrates the messages that are exchanged between a Client Application and a Directory Server when adding an attribute to the schema is successful.

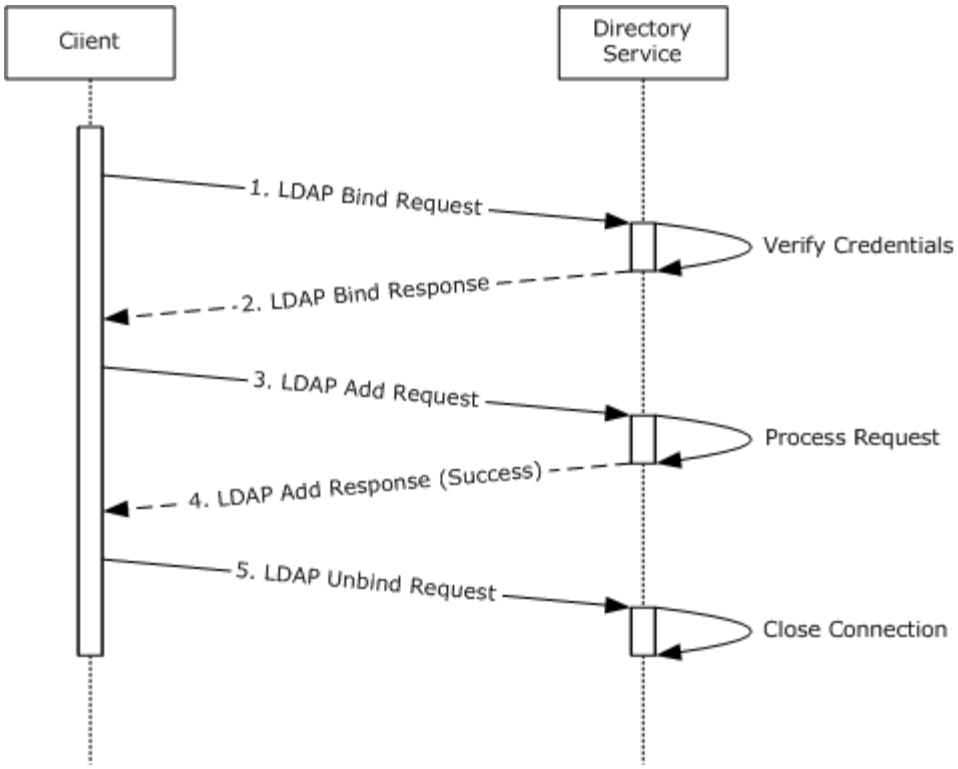

Figure 44: Message flow for extending the schema by adding an attribute

The sequence of events is described in the following steps.

1. The Client Application establishes an LDAP connection to the Directory Server. An LDAP bind request ([RFC2251] section 4.2, Bind Operation) is sent to the Directory Server with credentials of the Administrator.

2. The Directory Server verifies the credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task) and sends an LDAP bind response ([RFC2251] section 4.2.3, Bind Response) to the Client Application.

3. The Client Application sends an LDAP add request ([RFC2251] section 4.7, Add Operation) to the server. The request contains the values of the mandatory attributes ([MS-ADTS] section 3.1.1.2.3, Attributes) for the new object of class attributeSchema.

4. The server verifies all the processing rules and constraints ([MS-ADTS] sections 3.1.1.5.1, General, 3.1.1.5.2, Add Operation, and 3.1.1.2.5, Schema Modifications). Upon success, an instance of an object of class attributeSchema is added to the schema and the server sends an LDAP add response ([RFC2252] section 4.7, Add Operation) indicating that the object creation was successful.

5. The Client Application closes the LDAP connection by sending an LDAP unbind request ([RFC2251] section 4.3, Unbind Operation) to the directory server.

6.1.11.4 Final System State

A new attribute has been added to the schema.

6.1.12 Extend the Schema to Support an Application by Adding an Attribute to a Class

In this scenario, an Administrator extends the schema by adding an attribute already present in the schema to a class that is also already present in the schema. This is accomplished using the LDAP protocol. To perform this task, an Administrator runs a Client Application from a client computer targeting a Directory Server which is the Schema Master FSMO role in the Active Directory System. The Client Application adds an attribute to a class using the LDAP protocol.

This scenario uses the LDAP protocol.

The scenario covers the use cases in section 3.3.4.3.3.

6.1.12.1 Initial System State

The Active Directory System must meet all preconditions specified in sections 3.3.4.2.1 and 3.3.4.1.1.

6.1.12.2 Server Activity Diagram

The various activities performed by the server in response to client requests are illustrated in the following figure. The client communicates with the server by using the LDAP Protocols.

The server performs various activities upon receiving inputs from the client as follows:

1. Upon receiving an LDAP bind request ([RFC2251] section 4.2, Bind Operation) from the Client Application, the directory server interacts with the Windows Authentication Services, which authenticates the Administrator using the supplied credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task).

2. Upon receiving an LDAP modify request ([RFC2252] section 4.6, Modify Operation) from the Client Application, the directory server validates if the Administrator has the necessary access to perform the operation. If the Administrator does not have the necessary access to perform the operation, the directory server rejects the LDAP modify request.

3. The directory server validates that it owns the Schema Master FSMO role ([MS-ADTS] section 3.1.1.2.5, Schema Modifications) to perform the operation. If the directory server does not own the Schema Master FSMO role, the directory server rejects the LDAP add request.

4. The directory server validates the constraints to perform the operation as outlined in [MS-ADTS] section 3.1.1.2.5, Schema Modifications. If it fails to validate any of the constraints, the directory server rejects the LDAP modify request.

5. If all the constraints pass, the directory server performs the schema modification operation.

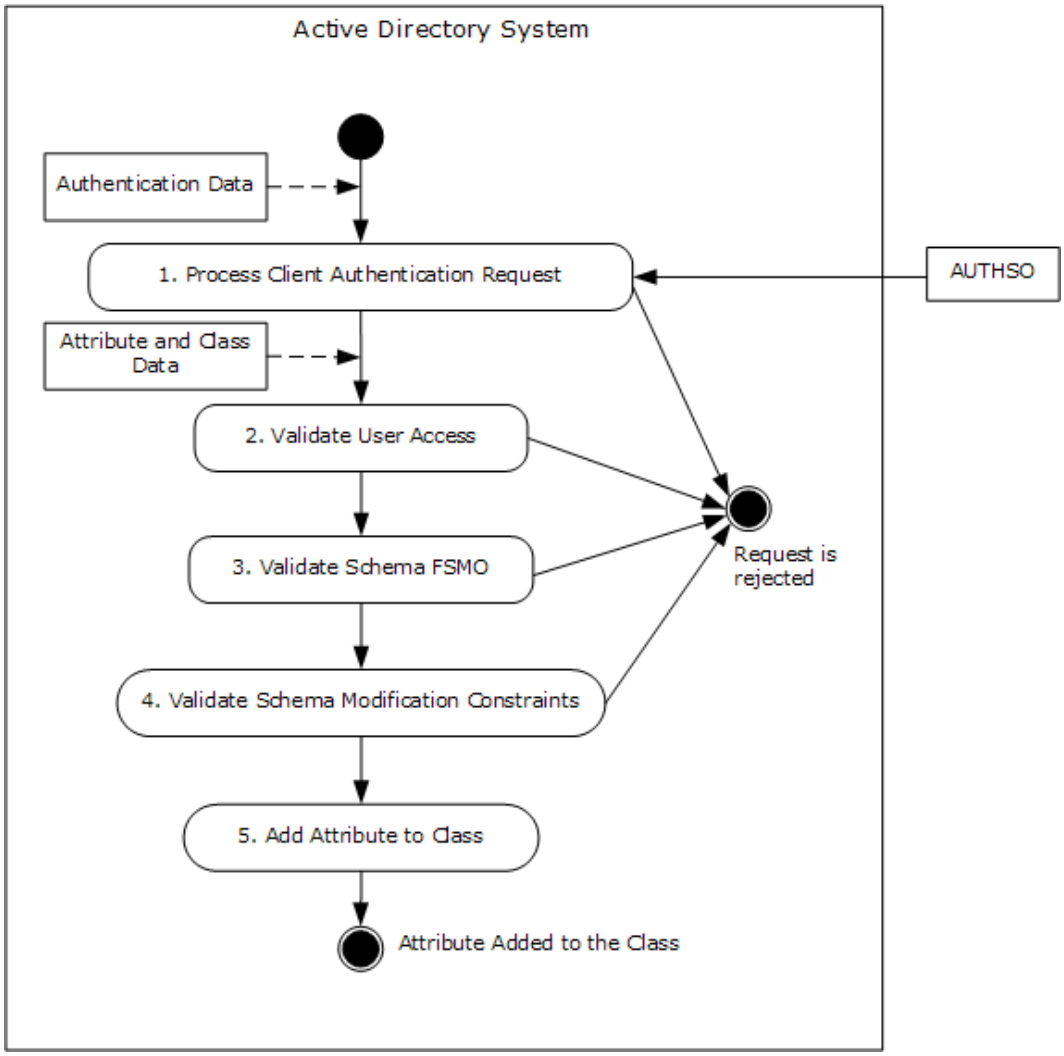

Figure 45: Server activity diagram for extending the schema by adding an attribute to a class

6.1.12.3 Sequence of Events

The diagram that follows illustrates the messages that are exchanged between a Client and a Directory Server when adding an attribute to a class is successful.

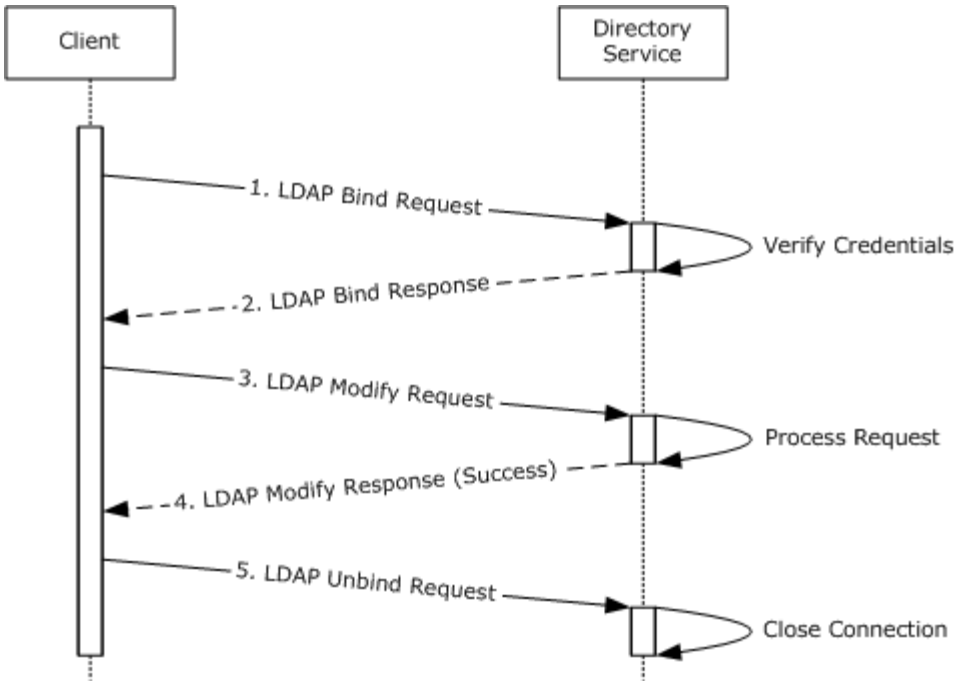

Figure 46: Message flow for extending the schema by adding an attribute to a class

The sequence of events is described in the following steps.

1. The Client Application establishes an LDAP connection to the Directory Server. An LDAP bind request ([RFC2251] section 4.2, Bind Operation) is sent to the Directory Server with credentials of the Administrator.

2. The Directory Server verifies the credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task) and sends an LDAP bind response ([RFC2251] section 4.2.3, Bind Response) to the Client Application.

3. The Client Application sends an LDAP modify request ([RFC2251] section 4.6, Modify Operation) to the server. The request contains the attribute name, the class name, and the attribute name in the class definition to which this attribute is to be added (namely **systemMayContain** as defined in [MS-ADA3] section 2.295, Attribute systemMayContain, or **mayContain** as defined in [MS-ADA2] section 2.18, Attribute mayContain).

4. The server verifies all the processing rules and constraints ([MS-ADTS] sections 3.1.1.5.1, General, 3.1.1.5.3, Modify Operation, and 3.1.1.2.5, Schema Modifications). Upon success, the attribute is added to the class and the server sends an LDAP add response ([RFC2252] section 4.7, Add Operation) indicating that the object modification was successful.

5. The Client Application closes the LDAP connection by sending an LDAP unbind request ([RFC2251] section 4.3, Unbind Operation) to the directory server.

6.1.12.4 Final System State

The schema has been modified to add the specified attribute as a valid attribute on the specified class.

6.1.13 Partition Directory Data with Organizational Units

In this scenario, a user partitions the directory data using organizational units. This can be accomplished using the LDAP protocol. To perform this task, a user runs a client application from a client computer targeting a directory server in the Active Directory System. The client application creates an organizational unit to represent an organization's department and existing directory objects are moved under the new departmental organizational unit.

This scenario covers the use case in section 3.3.4.1.5 and section 3.3.4.1.3.

6.1.13.1 Initial System State

The Active Directory System must meet all preconditions specified in section 3.3.4.1.5 section 3.3.4.1.3 and section 3.3.4.2.

6.1.13.2 Server Activity Diagram

The server activity diagram for this scenario is illustrated in the following figure. In this scenario, the client application connects to a directory server in the Active Directory System and communicates with it using the LDAP protocol. The activity diagram shows the actions the directory server takes as it transitions from an initial state to the state of moving existing directory objects under a newly created organizational unit.

The server activities are as follows:

1. Upon receiving an LDAP bind request ([RFC2251] section 4.2, Bind Operation) from the Client Application, the directory server interacts with the Windows Authentication Services, which authenticates the Administrator using the supplied credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task).

2. The directory server receives the LDAP add request ([RFC2251] section 4.7, Add Operation), containing data about the new organizational unit to be created. The directory server verifies the Administrator’s access-control rights to create the organizational unit as outlined in [MS-ADTS] section 3.1.1.5.2.1, Security Considerations. If the security constraints are not satisfied, the add request will be terminated with an appropriate error.

3. The directory server verifies the constraints for add operations are satisfied as outlined in [MS-ADTS] sections 3.1.1.5.1, General, and 3.1.1.5.2.2, Constraints. If the constraints are not satisfied, the add request will be terminated with an appropriate error.

4. The organizational unit object is created in the directory and populated with the data supplied in the request. Additional attributes on the object are populated based on the server’s processing rules as outlined in [MS-ADTS] section 3.1.1.5.2.4, Processing Specifics.

5. The directory server receives the LDAP modify distinguished name (DN) request(s) ([RFC2251] section 4.9, Modify DN Operation) containing data about the directory object(s) to be moved under the newly created organizational unit. The directory server verifies the Administrator’s access-control rights to modify the DNs of the directory object(s) as outlined in [MS-ADTS] section 3.1.1.5.4.1.1, Security Considerations. If the security constraints are not satisfied, the modify DN request(s) will be terminated with an appropriate error.

6. The directory server verifies the constraints for modify DN operations are satisfied as outlined in [MS-ADTS] sections 3.1.1.5.1, General, and 3.1.1.5.4.1.2, Constraints. If the constraints are not satisfied, the modify DN request will be terminated with an appropriate error.

7. The directory object(s) specified in the modify DN request are moved under the newly created organizational unit.

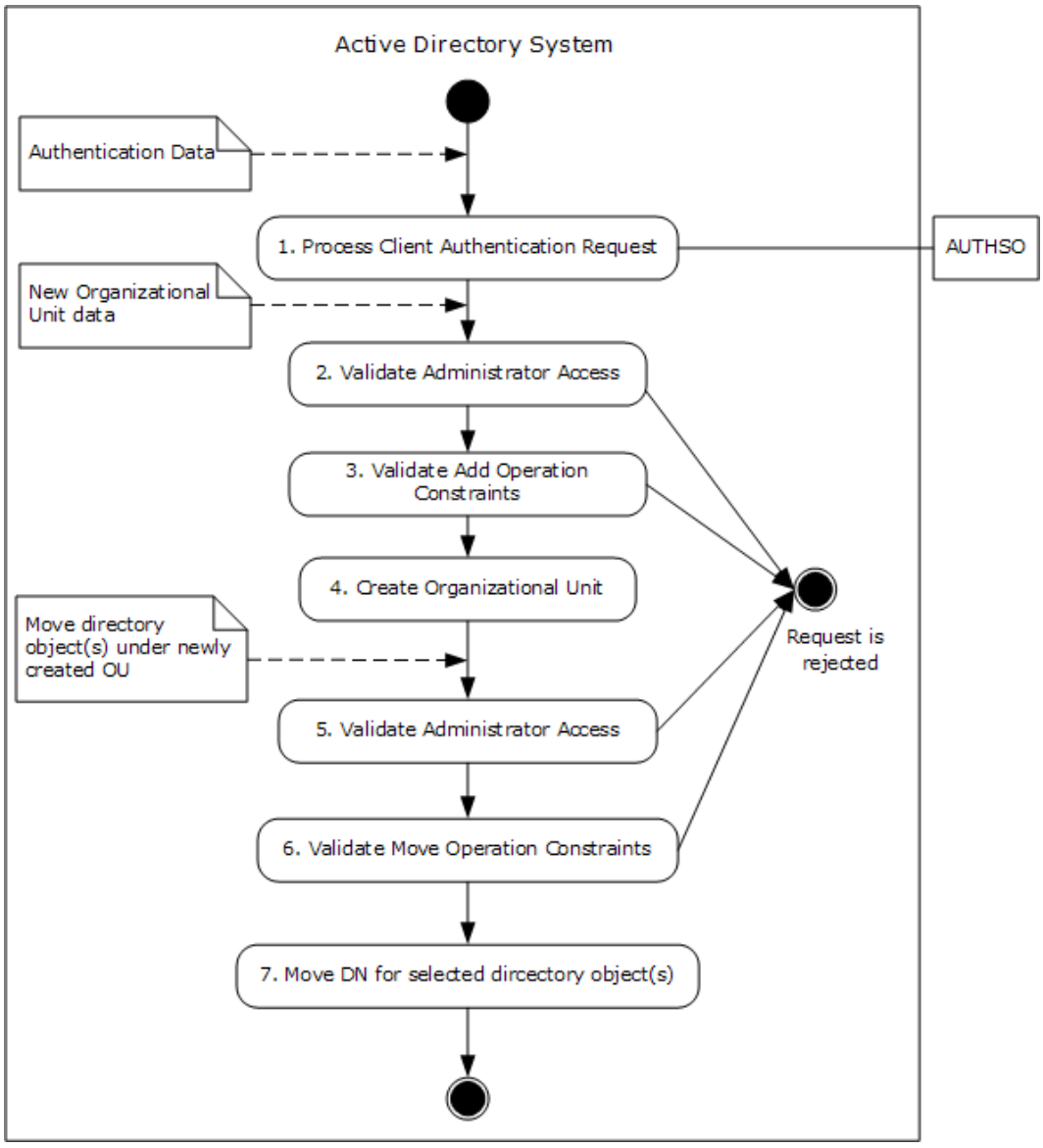

Figure 47: Server activity diagram for partitioning directory data

6.1.13.3 Sequence of Events

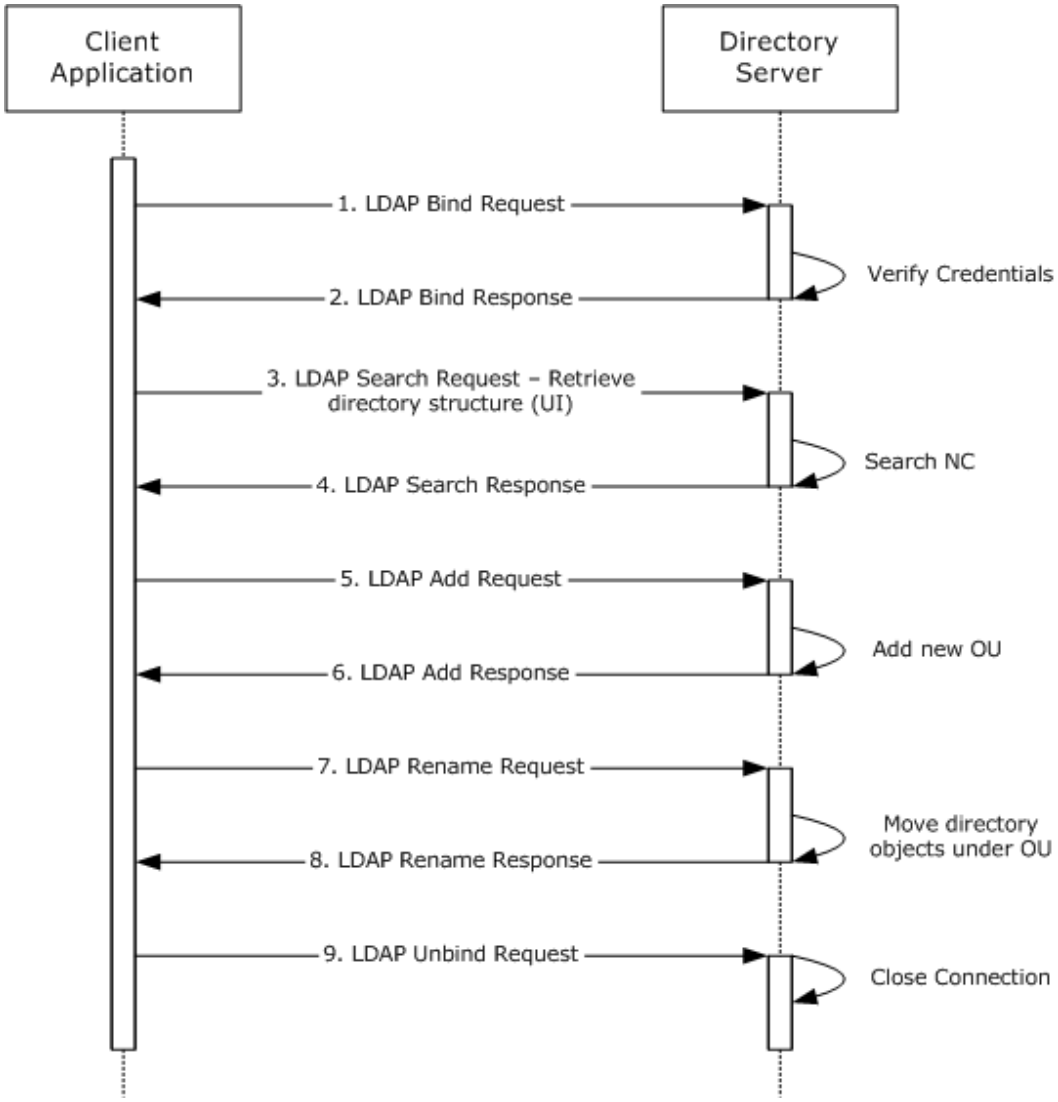

Figure 48: Message flow for partitioning directory data

Unless otherwise noted, all responses that include a return code contain a return code indicating that the operation was successfully performed.

1. The Client Application launches and an LDAP bind request ([RFC2251] section 4.2, Bind Operation) is sent to the directory server with credentials.

2. The directory server verifies the credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task) and sends an LDAP bind response ([RFC2251] section 4.2.3, Bind Response) to the Client Application.

3. The Client Application sends an LDAP search request ([RFC2251] section 4.5.1, Search Request) to the server. The application requests the sub-tree contents of the domain naming context or application NC for AD DS or application NC for AD LDS.

4. The server sends back an LDAP search response ([RFC2251] section 4.5.2, Search Result) containing the list of objects underneath the NC to populate data in the tool's user interface. This step is necessary only for user-interface display purposes specific to the example shown in the figure captioned, "Message flow for partitioning directory data".

5. The User selects a parent directory object where the new organizational unit will be located under and provides the name of the new organizational unit to the Client Application. An LDAP add request ([RFC2251] section 4.7, Add Operation) is sent to the directory server. The LDAP add operation contains the distinguished name (DN), and specifies that the object class of the object to be created is organizationalUnit ([MS-ADSC] section 2.212, Class organizationalUnit).

6. The server processes the add request ([RFC2251] section 4.7, Add Operation) and verifies the processing rules and constraints as described in [MS-ADTS] sections 3.1.1.5.1, General, and 3.1.1.5.2, Add Operation. It then sends an LDAP add response ([RFC2251] section 4.7, Add Operation) indicating success.

7. The User selects directory object(s) to be moved under the OU. The directory object(s) are moved by sending a series of LDAP modify DN request ([RFC2251] section 4.9, Modify DN Operation) to the directory server. The LDAP modify DN operation contains the DN of the object to be moved, the new relative distinguished name (RDN), the DN of the new parent and a **Boolean** flag to indicate whether the old RDN should be retained.

8. The server processes the modify DN request ([RFC2251] section 4.9, Modify DN Operation) for each directory object and verifies the processing rules and constraints as described in [MS-ADTS] sections 3.1.1.5.1, General, and 3.1.1.5.4, Modify DN. For each directory object that was moved, the directory server sends back an LDAP modify DN response ([RFC2251] section 4.9, Modify DN Operation) to the Client Application indicating success.

9. The client sends an LDAP unbind request ([RFC2251] section 4.3, Unbind Operation) to the server. The LDAP connection to the directory server is closed.

6.1.13.4 Final System State

The new organizational unit object has been created in the directory with the attributes specified. Selected directory objects are moved under the new organizational unit. No other state in the directory has changed.

6.1.14 Store Application Data in the Directory

Developers can create directory-enabled applications that store data in the Active Directory System. To store application data, a user runs the client application from a client computer, targeting a directory server in the Active Directory System. The Client Application creates the directory object in the application NC using the LDAP protocol.

This scenario covers the use case in section 3.3.4.1.1.

6.1.14.1 Initial System State

The Active Directory System must meet all preconditions specified in section 3.3.4.1.1.

6.1.14.2 Server Activity Diagram

The server activity diagram for this state model is illustrated in the following figure. In this scenario, the client application connects to a directory server in the Active Directory System and communicates with it using the LDAP protocol. The activity diagram shows the actions the directory

server takes as it transitions from an initial state to the state of a successful application directory object creation.

The server activities are as follows:

1. Upon receiving an LDAP bind request ([RFC2251] section 4.2, Bind Operation) from the Client Application, the directory server interacts with the Windows Authentication Services, which authenticates the Administrator using the supplied credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task).

2. Upon receiving an LDAP search request ([RFC2251] section 4.5.1, Search Request) from the Client Application, for the sub tree contents of the application NC, the server sends back an LDAP search response ([RFC2251] section 4.5.2, Search Result) containing the list of objects underneath the application NC.

3. The directory server receives the LDAP add request ([RFC2251] section 4.7, Add Operation), containing data about the new directory object to be created. The directory server verifies the Administrator's access-control rights to create the directory object as outlined in [MS-ADTS] section 3.1.1.5.2.1, Security Considerations. If the security constraints are not satisfied, the add request will be terminated with an appropriate error.

4. The directory server verifies the constraints for add operations are satisfied as outlined in [MS-ADTS] sections 3.1.1.5.1, General, and 3.1.1.5.2.2, Constraints. If the constraints are not satisfied, the add request will be terminated with an appropriate error.

5. The directory object is created under the application NC in the directory and populated with the data supplied in the request. Additional attributes on the object are populated based on the server's processing rules as outlined in [MS-ADTS] section 3.1.1.5.2.4, Processing Specifics.

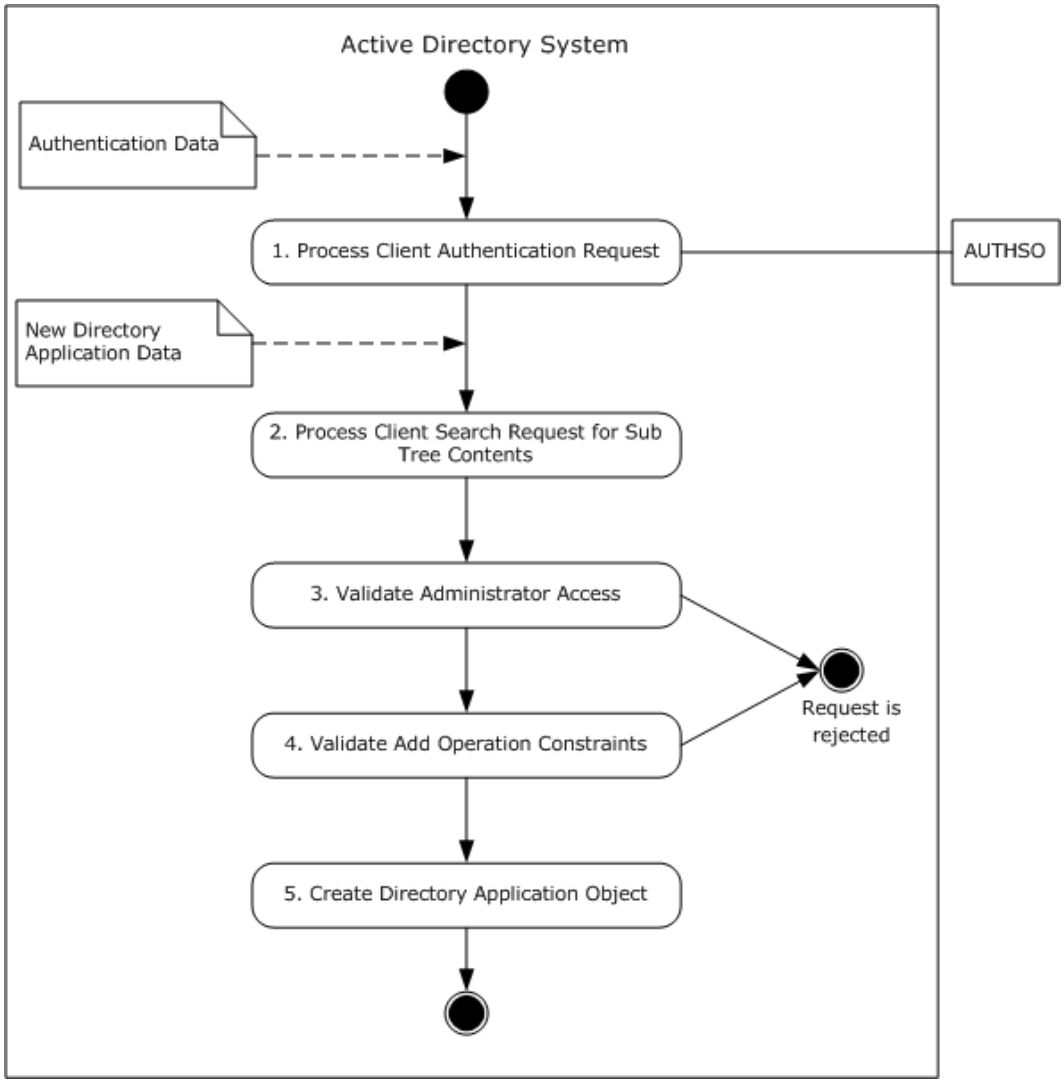

Figure 49: Server activity diagram for storing application data in the directory

6.1.14.3 Sequence of Events

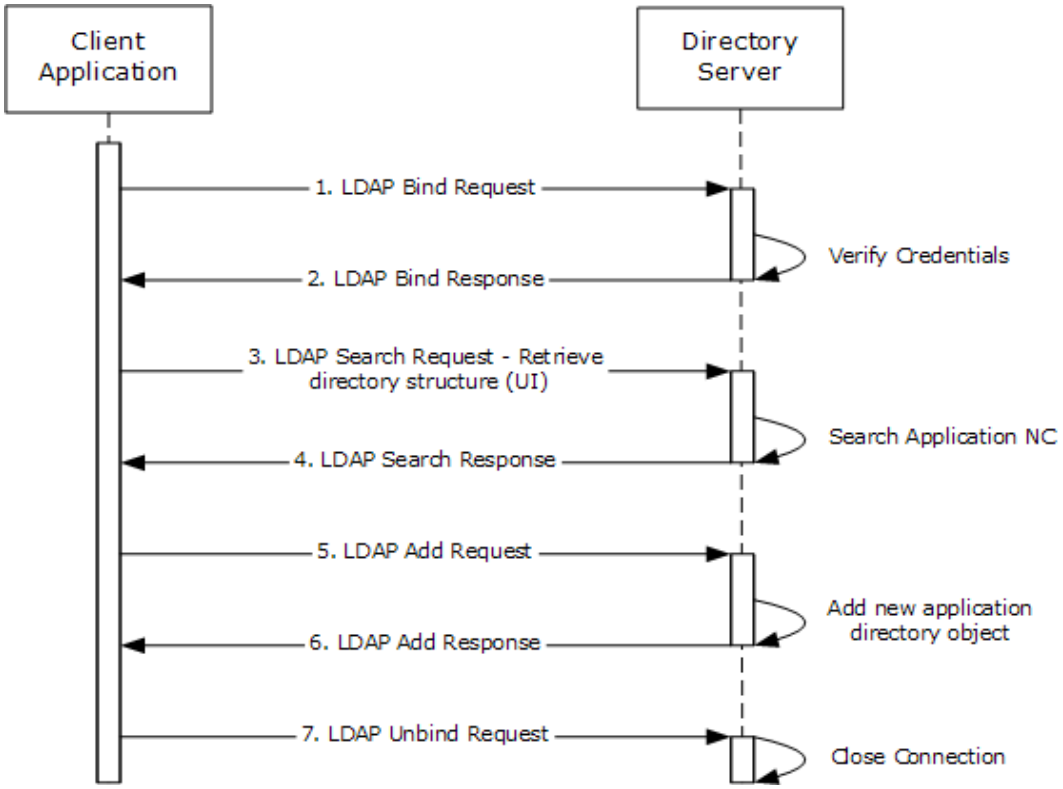

Figure 50: Message flow for storing application data in the directory

Unless otherwise noted, all responses that include a return code contain a return code indicating that the operation was successfully performed.

1. The Client Application launches and a LDAP bind request ([RFC2251] section 4.2, Bind Operation) is sent to the directory server with credentials.

2. The directory server verifies the credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task) and sends a LDAP bind response ([RFC2251] section 4.2.3, Bind Response) to the Client Application.

3. The Client Application sends a LDAP search request ([RFC2251] section 4.5.1, Search Request) to the server. The application requests the sub tree contents of the application NC.

4. The server sends back a LDAP search response ([RFC2251] section 4.5.2, Search Result) containing the list of objects underneath the application NC. The Client Application organizes this information and displays to the user. Steps 3 and 4 are artifacts of the Client Application.

5. The User selects a parent directory object where the new application directory object will be located under and all the necessary data is provided in preparation for the directory object creation. An LDAP add request ([RFC2251] section 4.7, Add Operation) is sent to the directory server. The LDAP add operation contains the distinguished name (DN), object class of the object to be created and any other mandatory parameters specified in [MS-ADTS] section 3.1.1.2.4.5, Content Rules. Security principal objects can only be created in the domain NC for AD DS and in

the application NC for AD LDS, as described in [MS-ADTS] section 5.1.1.5, Supported Types of Security Principals.

6. The server processes the add request ([RFC2251] section 4.7, Add Operation) and verifies the processing rules and constraints as described in [MS-ADTS] section 3.1.1.5.1, General, and section 3.1.1.5.2, Add Operation. It then sends back a LDAP add response ([RFC2251] section 4.7, Add Operation) to the Client Application indicating success.

7. The Client Application sends an LDAP unbind request ([RFC2251] section 4.3, Unbind Operation) to the server. The LDAP connection to the directory server is closed.

6.1.14.4 Final System State

The new directory object for use by the application has been created in the application NC with the attributes specified. No other state in the directory has changed.

6.1.15 Manage Access Control on Directory Objects

In this scenario, an administrator reads and modifies the access control settings of a directory object. This permits the administrator to control who has access to that object, and what type of access they have.

This scenario covers the use case in section 3.3.4.4.1.

6.1.15.1 Initial System State

The Active Directory System must meet all preconditions specified in section 3.3.4.4.1.

6.1.15.2 Server Activity Diagram

The various activities performed by the server in response to client requests are illustrated in the following figure. The client communicates with the server by using both the LDAP and the [MS-LSAT] Protocols.

The server performs various activities upon receiving inputs from the client as follows:

1. Upon receiving an LDAP bind request ([RFC2251] section 4.2, Bind Operation) from the client, the server verifies the client's credentials and accepts or rejects the bind request.

2. Upon receiving an LDAP search ([RFC2251] section 4.5, Search Operation) for the ntSecurityDescriptor attribute ([MS-ADTS] section 5.1.3, Authorization) of the object selected by the administrator, the server performs an access check on that object ([MS-ADTS] section 5.1.3, Authorization). If the access check fails or the operation itself fails, the server returns an error to the client.

3. The server retrieves the value for the ntSecurityDescriptor attribute ([MS-ADTS] section 5.1.3, Authorization) and includes it in the LDAP response.

4. Upon receiving an LsarOpenPolicy request from the client ([MS-LSAT] section 3.1.4.2, LsarOpenPolicy (Opnum 6)), the server performs an access check for the desired access ([MS-LSAD] section 3.1.4.2, Access Rights and Access Checks).

5. The server constructs an LSAPR_HANDLE as specified in [MS-LSAD] section 2.2.2.1, LSAPR_HANDLE. The response to the client indicates whether the operation was successful or not, as well as details about the failure if applicable.

6. Upon receiving an LsarLookupSids request from the client, the server performs an access check ([MS-LSAD] section 3.1.4.2, Access Rights and Access Checks).

7. The server performs the translation as explained in [MS-LSAT] section 3.1.4.11, LsarLookupSids (Opnum 15). The response to the client indicates whether the operation was successful or not, as well as details about the failure if applicable.

8. Upon receiving an LsarLookupNames request from the client, the server performs an access check ([MS-LSAD] section 3.1.4.2, Access Rights and Access Checks).

9. The server performs the translation as explained in [MS-LSAT] section 3.1.4.8, LsarLookupNames (Opnum 14). The response to the client indicates whether the operation was successful or not, as well as details about the failure if applicable.

10. Upon receiving an LsarClose request from the client, the server frees the resources held by the context handle opened in activity 3 ([MS-LSAT] section 3.1.4.3, LsarClose (Opnum 0)). The response to the client indicates whether the operation was successful or not, as well as details about the failure if applicable.

11. Upon receiving an LDAP modify request ([RFC2251] section 4.6, Modify Operation) from the client, the server verifies that the user has the appropriate permissions ([MS-ADTS] section 5.1.3, Authorization).

12. The server modifies the value of the nTSecurityDescriptor according to the request. The response to the client indicates whether the operation was successful or not, as well as details about the failure if applicable.

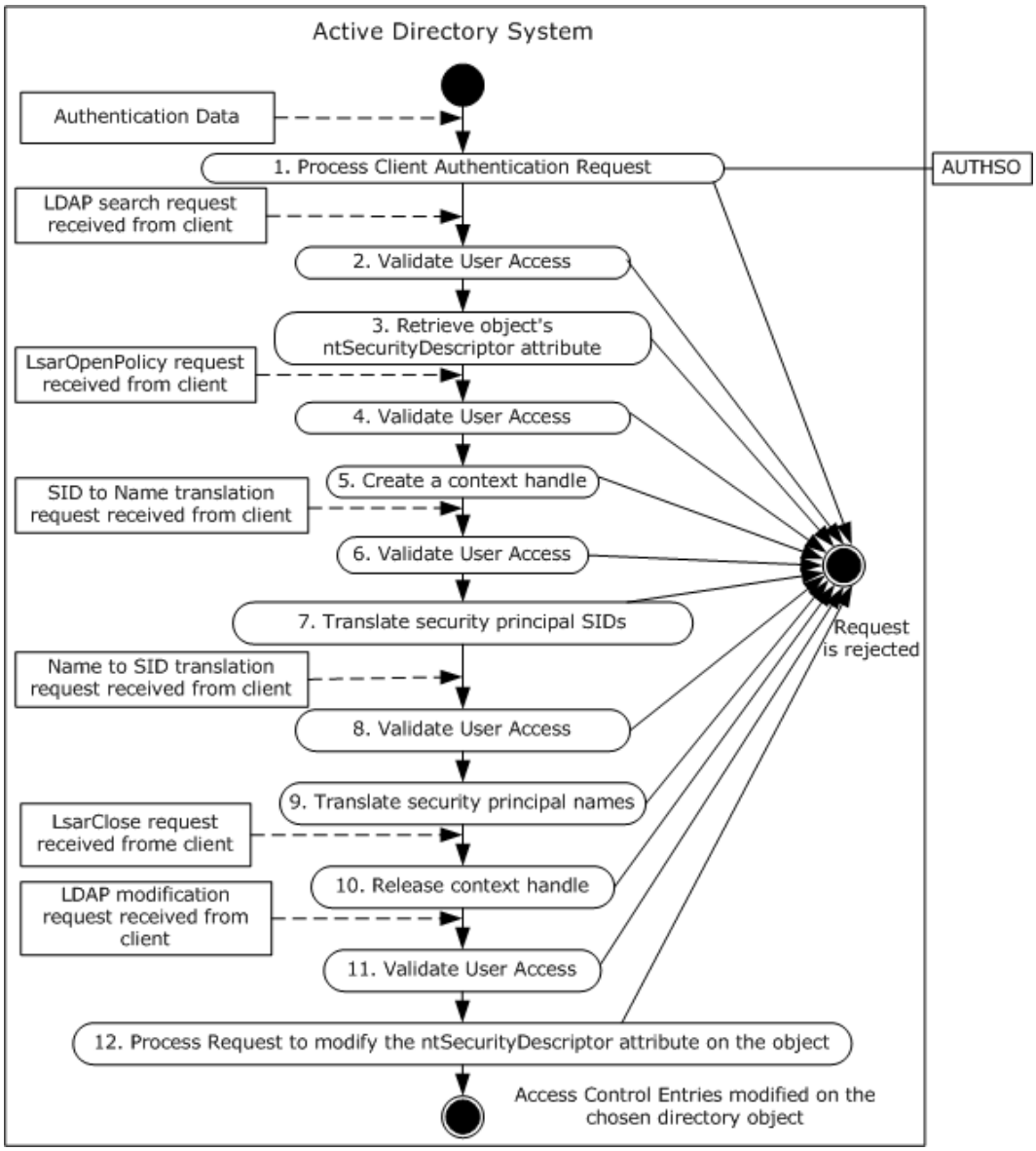

Figure 51: Server Activity Diagram for managing access control on a directory object

6.1.15.3 Sequence of Events

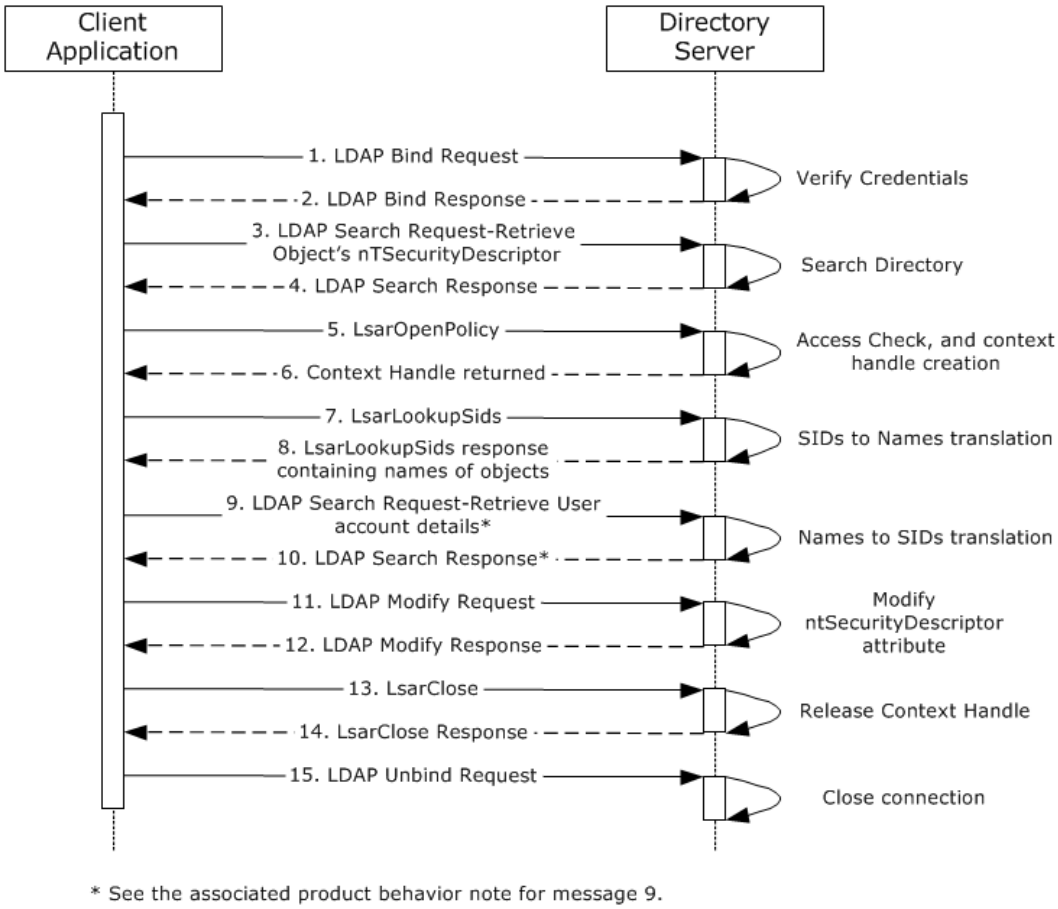

Figure 52: Message flow for managing access control on a directory object

Unless otherwise noted, all responses that include a return code contain a return code indicating that the operation was successfully performed.

1. The client application launches and a LDAP bind request ([RFC2251] section 4.2, Bind Operation) is sent to the directory server with credentials.

2. The directory server verifies the credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task) and sends a LDAP bind response ([RFC2251] section 4.2.3, Bind Response) to the client application.

3. An LDAP search request ([RFC2251] section 4.5.1, Search Request) is sent to the directory, querying the directory for the object specified by the administrator to the client application, to retrieve the nTSecurityDescriptor attribute ([MS-ADA3] section 2.37, Attribute nTSecurityDescriptor, and [MS-ADTS] section 5.1.3, Authorization).

4. The server performs an access check to ensure that the client has the permissions to read the attribute ([MS-ADTS] section 5.1.3, Authorization), then sends back a LDAP search response ([RFC2251] section 4.5.2, Search Result) containing the current value of the nTSecurityDescriptor for the object specified in the request.

5. The client extracts the SIDs of the security principals specified in the ntSecurityDescriptor of the object ([MS-ADTS] section 5.1, LDAP Security, and [MS-DTYP] section 2.4.6, SECURITY_DESCRIPTOR), then sends an LsarOpenPolicy request ([MS-LSAT] section 3.1.4.2, LsarOpenPolicy (Opnum 6)) with the *DesiredAccess* parameter set to POLICY_LOOKUP_NAMES ([MS-LSAD] section 2.2.1.1.2, ACCESS_MASK for Policy Objects) to the server to retrieve a context handle that it can later use to request names/SIDs translations.

6. The server processes the LsarOpenPolicy request by performing an access check for the desired access ([MS-LSAD] section 3.1.4.2, Access Rights and Access Checks) and constructing an LSAPR_HANDLE ([MS-LSAD] section 2.2.2.1, LSAPR_HANDLE) as specified in [MS-LSAT] section 3.1.4.2, LsarOpenPolicy (Opnum 6). It sends the response back to the client.

7. The client sends a LsarLookupSids request ([MS-LSAT] section 3.1.4.11, LsarLookupSids (Opnum 15)) using the server handle from step 2. In this request it specifies the SIDs that it wants to be translated to names for display to the administrator.

8. The server processes the request as specified in [MS-LSAT] section 3.1.4.11, LsarLookupSids (Opnum 15), and sends a response back to the client including the names of the security principals whose SIDs were passed in the request.

9. The client receives input from the administrator as to which users he wants to define Access Control Entries for on the object. It then sends an LsarLookupNames request ([MS-LSAT] section 3.1.4.8, LsarLookupNames (Opnum 14)) including those names in order to retrieve their SIDs.

10. The server translates the SIDs included in the request according to the processing defined in [MS-LSAT] section 3.1.4.8, LsarLookupNames (Opnum 14), and sends a response back to the client including the SIDs of the users whose names were passed in the request.

11. The client prepares the updated value for the nTSecurityDescriptor attribute ([MS-ADTS] section 5.1, LDAP Security) and sends an LDAP modify request ([RFC2251] section 4.6, Modify Operation) to commit that update.

12. The server verified that the client has permissions to update the attribute, then processes the modify request and performs validation as described in [MS-ADTS] section 3.1.1.5.1, General, and section 3.1.1.5.3, Modify Operation. It then sends an LDAP modify response indicating success.

13. The client finally sends an LsarClose Request ([MS-LSAT] section 3.1.4.3, LsarClose (Opnum 0)) to the server to release the context handle that was opened in step 6.

14. The server releases the handle and sends a response to the client indicating success.

15. The client application sends an LDAP unbind request ([RFC2251] section 4.3, Unbind Operation) to the server. The LDAP connection to the directory server is closed.

6.1.15.4 Final System State

The access control entries (ACEs) on the object specified by the administrator have been updated to secure that object as specified by the administrator.

6.1.16 Raise the Domain Functional Level

In this scenario, an administrator modifies the ms-DS-Behavior-Version attribute ([MS-ADA2] section 2.246, Attribute msDS-Behavior-Version, and [MS-ADTS] section 3.1.1.5.3.1.1.5, msDS-Behavior-Version) using the LDAP protocol, to an incremental value. To perform this task, an administrator runs a client application from a client computer, targeting a directory server in the

Active Directory System and raises the domain functional level ([MS-ADTS] section 6.1.4.3, msDS-Behavior-Version: Domain NC Functional Level).

This scenario applies only to AD DS.

This scenario covers the use case in section 3.3.4.1.3.

6.1.16.1 Initial System State

The Active Directory System must meet all preconditions specified in section 3.3.4.1.3.

6.1.16.2 Server Activity Diagram

The server activity diagram for this scenario is illustrated in the following figure. In this scenario, the client application connects to a directory server in the Active Directory System and communicates with it using the LDAP protocol. The activity diagram shows the activities the server performs as it transitions from an initial state to the state of the domain Functional level being raised incrementally.

The server activities are as follows:

1. Upon receiving an LDAP bind request ([RFC2251] section 4.2, Bind Operation) from the client application, the directory server interacts with the Windows Authentication Services, which authenticates the administrator using the supplied credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task).

2. Upon receiving an LDAP search request ([RFC2251] section 4.5.1, Search Request) from the client application, the directory server queries the base domain, looking for an **msDS-Behavior-Version** attribute ([MS-ADA2] section 2.246, Attribute msDS-Behavior-Version, and [MS-ADTS] section 3.1.1.5.3.1.1.5, msDS-Behavior-Version) and sends back an LDAP search response ([RFC2251] section 4.5.2, Search Result) containing the current domain functional level value (2, DS_BEHAVIOR_WIN2003 [MS-ADTS] section 6.1.4.2, msDS-Behavior-Version: DC Functional Level).

3. The directory server receives a modify ([RFC2251] section 4.6, Modify Operation) attribute request from the client application. The directory server verifies the administrator's access-control rights to see the domain crossRef object and its attributes as outlined in [MS-ADTS] sections 3.1.1.4.3, Access Checks, and 3.1.1.4.4, Extended Access Checks. If the security constraints are not satisfied, the modify request will be terminated with an appropriate error. The directory server proceeds to verify the administrator's access-control rights to modify the domain crossRef object as outlined in [MS-ADTS] sections 3.1.1.5.5.3, Protected Objects, and 3.1.1.5.5.4, Security Considerations. If the security constraints are not satisfied, the modify request will be terminated with an appropriate error.

4. The directory server verifies the constraints for modify operations are satisfied as outlined in [MS-ADTS] sections 3.1.1.5.3.1.2, FSMO Changes, and 3.1.1.5.3.4, BehaviorVersion Updates. If the constraints are not satisfied, the modify request will be terminated with an appropriate error.

5. The directory server performs the modification on the domain crossRef object and the domain functional level is now raised to the new level.

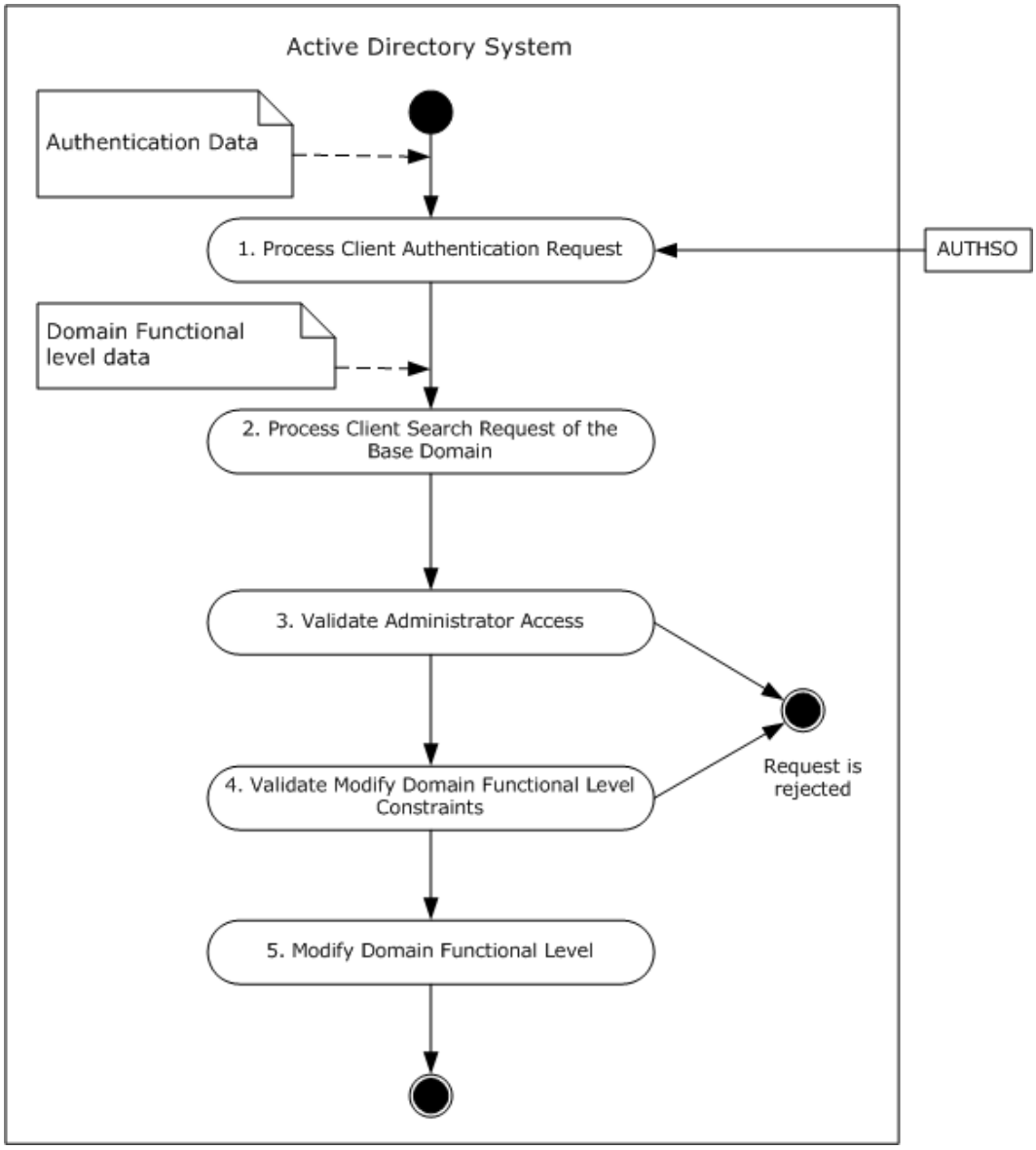

Figure 53: Server activity diagram for raising the domain functional level

6.1.16.3 Sequence of Events

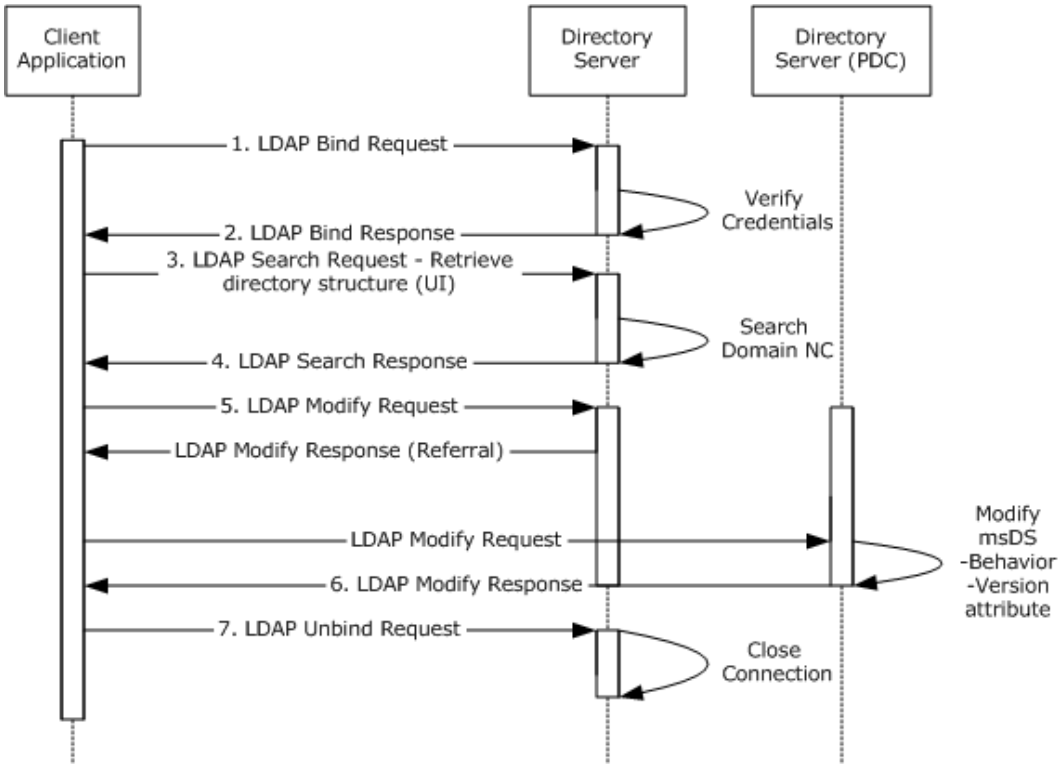

Figure 54: Message flow for raising the domain functional level

Unless otherwise noted, all responses that include a return code contain a return code indicating that the operation was successfully performed.

1. The client application launches and a LDAP bind request ([RFC2251] section 4.2, Bind Operation) is sent to the directory server with credentials.

2. The directory server verifies the credentials ([MS-AUTHSO] section 4, Interactive Domain Logon Task) and sends a LDAP bind response ([RFC2251] section 4.2.3, Bind Response) to the client application.

3. An LDAP search request ([RFC2251] section 4.5.1, Search Request) is sent to the directory, querying the base domain, looking for msDS-Behavior-Version attribute ([MS-ADA2] section 2.246, Attribute msDS-Behavior-Version, and [MS-ADTS] section 3.1.1.5.3.1.1.5, msDS-Behavior-Version).

4. The server sends back a LDAP search response ([RFC2251] section 4.5.2, Search Result) containing the current domain functional level value (2, DS_BEHAVIOR_WIN2003 [MS-ADTS] section 6.1.4.2, msDS-Behavior-Version: DC Functional Level).

5. The User interacts with the Client Application and provides the name of the domain NC and the new domain functional level (3, DS_BEHAVIOR_WIN2008 [MS-ADTS] section 6.1.4.2, msDS-Behavior-Version: DC Functional Level).

6. An LDAP modify request ([RFC2251] section 4.6, Modify Operation) is sent to the server. The LDAP modify operation contains the distinguishedName of the domain along with the msDS-

Behavior-Version attribute ([MS-ADA2] section 2.246, Attribute msDS-Behavior-Version) as a replace operation with the value of 3.

7. If necessary, the client application will reroute the modify request ([RFC2251] section 4.6, Modify Operation) to the PDC FSMO if a referral was returned from the initial modify request in step 6. This is because only the PDC FSMO role holder can perform this modification ([MS-ADTS] section 6.1.4.3, msDS-Behavior-Version: Domain NC Functional Level). The server processes the modify request and verifies the processing rules and constraints as described in [MS-ADTS] section 3.1.1.5.1, General, and section 3.1.1.5.3, Modify Operation. It then sends an LDAP modify response ([RFC2251] section 4.6, Modify Operation) indicating success.

8. The client application sends an LDAP unbind request ([RFC2251] section 4.3, Unbind Operation) to the server. The LDAP connection to the directory server is closed.

6.1.16.4 Final System State

The crossRef object is modified and the domain functional level is raised.

6.2 Communication Details

Some of the protocols in the Active Directory System can operate on more than one transport. However, for several of these protocols, not all transports are considered equivalent. In these cases, the client is either encouraged or required to choose a specific transport when performing an operation using the protocol. This section documents these constraints. For information on the transports supported by the protocols in the Active Directory System, see section 6.3 as well as the description in the Technical Documents for the individual protocols.

Windows uses LDAP as defined in [RFC1777] for LDAP version 2, and [RFC3377] and [RFC2251] for LDAP version 3. Clients authenticated to an Active Directory server using **GSS-(SPNEGO) SASL** authentication mechanism ([MS-ADTS] section 5.1.1.1.2, SASL Authentication), observe LDAP version 3 compliant semantics, with the extensions and deviations documented in [MS-ADTS] section 3.1.1.3.1, LDAP Conformance. Unauthenticated clients and clients authenticated under a different authentication mechanism observe LDAP behavior compliant with the requested LDAP version. Windows clients authenticate to the Active Directory server using GSS-(SPNEGO) SASL authentication mechanism.

While the Active Directory System supports both TCP and UDP transports for LDAP versions 2 and 3, TCP is the preferred transport. LDAP over the UDP transport does not have a mechanism by which clients can authenticate to the directory service and so clients can only perform two specific anonymous operations. These anonymous operations are **rootDSE** search and LDAP abandon. The UDP transport is primarily intended for use by LDAP ping requests used for the AD DS domain controller location mechanism described in [MS-ADTS] section 6.3, Publishing and Locating a Domain Controller. LDAP over TCP is described in sections 6.2.1 through 6.2.6, while LDAP over UDP is described in section 6.2.7.

For SAMR, all opnums are exposed over both the Server Message Block (SMB) and TCP transports except that SamrValidatePassword (opnum 67) is only exposed via the TCP transport. Clients MUST use the TCP transport when performing this operation.

For LSAT, clients MUST choose which transport to use (either SMB or TCP) based on the opnum of the request that they are sending to the server, as documented in [MS-LSAT] section 2.1, Transport. Each transport supports only a subset of the total set of opnums.

The Active Directory System does not define any messages or message-processing rules beyond those described in the Technical Documents for the protocols and protocol extensions listed in section 2.2.

6.2.1 Connection Resolution of LDAP Clients

Lightweight Directory Access Protocol (LDAP) client establishes an LDAP connection to the directory server based on the given server information. The server information can be NULL (indicates that the joined domain name should be used), domain name (DNS/NetBIOS), server host name, or server IP address. Below is the connection resolution logic for the given server information:

- NULL (indicates that the joined domain name should be used)
  - LDAP client uses the DC Location algorithm, as described in section 6.2.6.2.2 (Connecting to a Directory Server), to locate a server for the joined domain name.
- Domain name (DNS/NetBIOS)
  - LDAP client uses the DC Location algorithm, as described in section 6.2.6.2.2 (Connecting to a Directory Server), to locate a server for the given domain name.
- Server host name
  - LDAP client uses the given server host name to establish an LDAP connection.
- Server IP address
  - LDAP client uses the given server IP address to establish an LDAP connection.

6.2.2 ADConnection Overview

Windows uses LDAP over TCP as defined in [RFC1777] for LDAP version 2, and [RFC3377] and [RFC2251] for LDAP version 3. The following sections describe only the additional behaviors of the Microsoft LDAP client which are not specified by these RFCs.

For LDAP over TCP, an ADConnection manages the TCP connections that are used for communication between the client and Active Directory servers. The typical sequence of use of an ADConnection is:

1. Initialize an ADConnection, which creates the ADConnection but does not yet connect to the Active Directory server.
2. Set options on the ADConnection as outlined in section 6.2.6.1.2.
3. Establish the ADConnection to an Active Directory server, which establishes the TCP connection with the server ([RFC2251] section 5.2.1, Transmission Control Protocol (TCP)).
4. Perform an LDAP bind ([RFC2251] section 4.2, Bind Operation) on the ADConnection, which authenticates the client to the Active Directory service.
5. Perform one or more LDAP operations such as search ([RFC2251] section 4.5, Search Operation), modify ([RFC2251] section 4.6, Modify Operation), or delete ([RFC2251] section 4.8, Delete Operation) on the ADConnection. An LDAP operation will consist of an LDAP request and the resulting LDAP response(s).
6. Perform an LDAP unbind ([RFC2251] section 4.3, Unbind Operation) on the ADConnection.

This sequence is shown in the following Client Activity diagram.

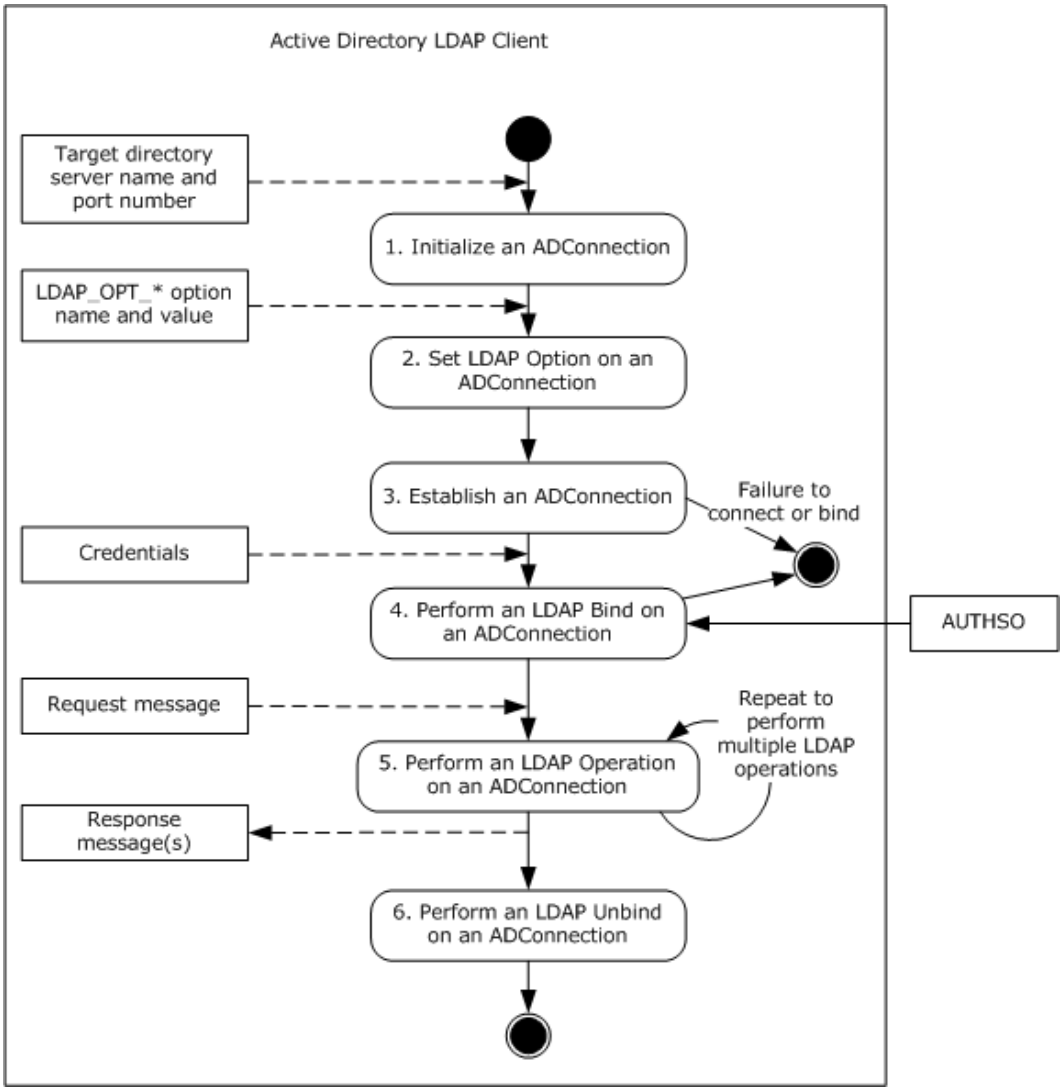

Figure 55: Client activity diagram

An ADConnection allows a client to use the connectivity to an Active Directory server for multiple LDAP operations, thereby reducing both client- and server-side processing costs and reducing the serialized time delays of TCP connection establishment and LDAP bind authentications. The ADConnection abstraction allows a client application to perform an LDAP operation with an Active Directory server, and automatically follow any LDAP referrals ([RFC2251] section 4.1.11, Referral) and continuation references ([RFC2251] section 4.5.3, Continuation References in the Search Result). In the case of referrals and continuation references, the client establishes an additional TCP connection to the directory server specified in each referral (or continuation reference) and sends a request as directed by that referral or reference.

The ADConnection can also attempt to maintain connectivity to the directory service in the event a directory server becomes unresponsive or unreachable, by attempting to reconnect to the directory service and resending pending requests.

6.2.3 ADConnection Abstract Data Model

This section describes a conceptual model of a possible data organization that an implementation maintains. The described organization is provided to facilitate the explanation of how the client behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

The data model defines a set of structures that a client operates on, and also an element which lists all active ADConnection objects. The primary structure is **ADConnection**, with two supporting structures **LDAPRequest** and **ConnectionInfo**. These structures are described below.

ComputerRole: An abstract variable of type **DSROLE_MACHINE_ROLE** that indicates the current domain membership role of the machine on which the LDAP client is executing as described in [MS-DSSP] section 3.2.1, Abstract Data Model.

ADCList: A list of **ADConnection** objects. It is initialized to the empty list.

ADConnection: A structure that tracks a connection to an Active Directory server. Sub-elements with names of the form LDAP_OPT_<rest of name> are ADM elements that can be set by the Setting an LDAP Option on an ADConnection (section 6.2.6.1.2) task. It has the following elements:

LDAP_OPT_REFERRAL_HOP_LIMIT: An unsigned integer indicating limit on the number of referrals or continuation references that the client will follow for a single LDAP request. The default value is 32. A value of zero indicates no limit.

LDAP_OPT_REFERRALS: An enumeration indicating whether the client is to automatically follow referrals and continuation references. Valid values are:

- ON: The default value. The client automatically follows both referrals and continuation references.
- OFF: The client does not follow either referrals or continuation references automatically.
- LDAP_CHASE_CONTINUATION_REFERENCES: The client follows only continuation references automatically.
- LDAP_CHASE_REFERRALS: The client follows only referrals automatically.

LDAP_OPT_TIMELIMIT: An unsigned integer indicating the maximum time in seconds the client will wait for a response to an LDAP request. A value of 0 indicates a limit of 120 seconds for LDAP bind requests and no time limit for all other requests. The default value is 0. This ADM element can be overridden if a value for the time limit is specified in the parameters of the Performing an LDAP Operation on an ADConnection (section 6.2.6.1.6) task.

LDAP_OPT_SIZELIMIT: An unsigned integer indicating the maximum number of results the client will request the directory server to send for a given search request. The valid range is from 0 to 2^32 -1. A value of 0 indicates that there is no limit. The default value is 0. This ADM element can be overridden if a value for the size limit is specified in the parameters of the Performing an LDAP Operation on an ADConnection (section 6.2.6.1.6) task.

LDAP_OPT_AREC_EXCLUSIVE: A Boolean flag indicating whether the client is to skip the DC Location processing during host resolution. A value of TRUE indicates that DC Location will be skipped. The default value is FALSE.

LDAP_OPT_DNSDOMAIN_NAME: A null-terminated string used in constructing a **service principal name (SPN)** when performing an LDAP bind.

LDAP_OPT_GETDSNAME_FLAGS: An unsigned integer indicating the flags to be passed when invoking DsrGetDcNameEx2 to perform DC location. For more information, see [MS-NRPC] section 3.5.4.3.1, DsrGetDcNameEx2 (Opnum 34). By default, the value is 0.

LDAP_OPT_AUTO_RECONNECT: A Boolean flag indicating whether the client will attempt to automatically reconnect to a server when an existing connection is lost. The default value is TRUE.

LDAP_OPT_PING_KEEP_ALIVE: An unsigned integer indicating the maximum time in seconds the client will allow a connection to be idle (that is, receiving no responses) while waiting on outstanding requests on the connection before sending the directory server a sequence of PING requests. Valid range is 5 seconds to 2^32-1 seconds. The default value is 120 seconds.

LDAP_OPT_PING_WAIT_TIME: An unsigned integer indicating the time in milliseconds (ms) the client will wait for a response to a PING request. Valid range is 10 ms to 60000 ms. The default value is 2000 ms.

LDAP_OPT_PING_LIMIT: An unsigned integer indicating the maximum number of consecutive PINGs the client will send without getting an ICMP echo response ([RFC792]) before triggering the Processing Network Errors (section 6.2.6.3.1) event. Valid range is 0 to 2^32-1. A value of 0 indicates that the client never PINGs the server for an idle connection. The default value is 4.

LDAP_OPT_ENCRYPT: A Boolean flag indicating whether SASL layer encryption ([MS-ADTS] section 5.1.2.1, Using SASL) is enabled on the connection. If a non-default value is desired, this must be set prior to performing an LDAP bind on the connection. The default value is FALSE.

LDAP_OPT_SIGN: A Boolean flag indicating whether SASL layer signing ([MS-ADTS] section 5.1.2.1, Using SASL) is enabled on the connection. If a non-default value is desired, this must be set prior to performing an LDAP bind on the connection. The default value is TRUE.

LDAP_OPT_TCP_KEEPALIVE: A Boolean flag indicating whether TCP Keep-alives ([RFC1122] section 4.2.3.6, TCP Keep-Alives) are enabled on **primaryConnection**. The default value is FALSE.

LDAP_OPT_AUTH_INFO: An instance of an **AuthInfo** structure representing the bind method and credentials to use when authenticating to the directory server. By default, the *bindMethod* is SASL using the GSS-SPNEGO mechanism, and the name and password are those of the identity of the protocol or system using the LDAP client. When attempting to use the server in fast bind mode, the bindMethod is set to simple bind prior to performing an LDAP bind on the connection. Subsequently, only simple binds can be performed to the server as long as the connection is in fast bind mode. See [MS-ADTS] section 5.1.1.1, (Supported Authentication Methods) for a list of supported bind methods.

LDAP_OPT_PROTOCOL_VERSION: An unsigned integer indicating which version of the LDAP protocol the connection uses. Valid values are 2 and 3. The default value is 2. If a non-default value is desired, this must be set prior to performing an LDAP bind on the connection.

primaryConnection: A ConnectionInfo structure representing the TCP connection for the target directory server. The target directory server is the directory server specified as a parameter to the Initializing an ADConnection (section 6.2.6.1.1) task.

referralConnections: A list of **ConnectionInfo** structures representing the TCP connections to directory servers used for following referrals or continuation references. It is initialized to an empty list.

LDAPRequest: A structure which tracks an LDAP request ([RFC1777] section 2, Protocol Model, for LDAP version 2 and [RFC2251] section 3.1, Protocol Model, for LDAP version 3). It has the following elements:

requestMessage: An LDAPMessage (as defined in [RFC1777] section 4, Elements of Protocol, for LDAP version 2 and [RFC2251] section 4.1.1, Message Envelope, for LDAP version 3) representing a request that the client sends to the server.

resultMessages: A sequence of LDAPMessage (as defined in [RFC1777] section 4, Elements of Protocol, for LDAP version 2 and [RFC2251] section 4.1.1, Message Envelope, for LDAP version 3) representing the results that the client receives in response to **requestMessage**.

numReferrals: An unsigned integer indicating the number of consecutive referrals or continuation references the client has received when processing a given LDAP response ([RFC1777] section 2, Message Envelope, for LDAP version 2 and [RFC2251] section 4.1.11, Referral, for LDAP version 3). It is initialized to zero.

requestTimer: A timer with second granularity used to track how long the client has waited for a response to **requestMessage**.

numResends: An unsigned integer indicating the number of times this request has been resent by the Autoreconnecting to a Directory Server (section 6.2.6.2.7) task. It is initialized to 0.

ConnectionInfo: A structure which tracks a TCP connection to a directory server. It has the following elements:

networkConnection: An abstract element representing a TCP connection ([RFC793] section 1.5, Operation) to the directory server. It is initialized to NULL.

portNumber: An unsigned integer indicating the TCP port number ([RFC793] section 1.5, Operation) to use when connecting to the directory server.

targetName: A null-terminated string used to locate a directory server. It can be NULL, indicating that the directory server for the joined domain should be located, a domain name (DNS/NetBIOS), a host name, or an IP address.

bindHasHappened: A Boolean flag indicating whether an LDAP bind has been successfully performed on this connection. The default value is FALSE.

pingRetries: An unsigned integer indicating the number of consecutive ICMP echo requests or "pings" ([RFC792]) the client has sent to the directory server but for which it has not received a response. It is initialized to 0.

pingKeepaliveTimer: A timer with second granularity used to track how long it has been since the client last received a response to any request on this connection.

pendingRequestList: A list of **LDAPRequest** elements representing all outstanding requests on this connection. It is initialized to NULL.

AuthInfo: A structure which is used to authenticate to the directory server. It has the following elements:

bindMethod: The bind method that will be used to authenticate to the directory server. See [MS-ADTS] section 5.1.1.1 (Supported Authentication Methods) for a list of supported bind methods.

name: A string containing the user name of the credential that will be used to authenticate to the directory server. When this string is set to NULL or is not set, use the identity of the protocol or system that is using the LDAP client.

password: A string containing the password of the credentials that will be used to authenticate to the directory server. When this string is set to NULL or is not set, use the identity of the protocol or system that is using the LDAP client.

6.2.4 Handling Network Errors

The LDAP client relies extensively on the underlying TCP/IP implementation to detect network errors which indicate that the remote Active Directory server is either unreachable or is unavailable for network operations. These errors depend on the local implementation of TCP/IP but include errors such as network media is unavailable, host is unreachable, port is unreachable, route is unavailable, and TCP keep alive failed. For the purposes of this specification, an error encountered on a network read or write will appear as a signaled event, invoking the handler covered in Processing Network Errors (section 6.2.6.3.1).

6.2.5 ICMP Pings

If a client has pending requests on a connection to an Active Directory server but has not received any responses to any of those requests for a period of time specified by **ADConnection.LDAP_OPT_PING_KEEP_ALIVE**, as tracked by the **pingKeepAliveTimer** on the **ConnectionInfo** structure which holds the TCP connection to that server, the client pings the directory server by sending an ICMP echo message as described in [RFC792] and waiting up to **ADConnection.LDAP_OPT_PING_WAIT_TIME** milliseconds for any echo reply from the corresponding server. If no such reply is received, the client repeats the process of pinging the server up to **ADConnection.LDAP_OPT_PING_LIMIT** times. If the client receives no response after sending the maximum number of pings, or receives an error from the underlying network implementation during this process, it triggers the Processing Network Errors (section 6.2.6.3.1) event.

6.2.6 Tasks and Events

The following sections describe tasks and events involved in the management of **ADConnections**. These sections list parameters and results for each task. These represent data passed to an instance of the task at the time it is invoked or triggered or the result returned by the task. This information is intended to facilitate the reader's conceptual understanding of the specification. While a task's processing rules might depend upon associations established by the structure of its parameters, such association can be achieved in other ways. Implementations can depart from this abstraction so long as their external behavior remains consistent with that described in this document.

The interrelationship between these tasks and events is illustrated in the following Task Relationship diagram.

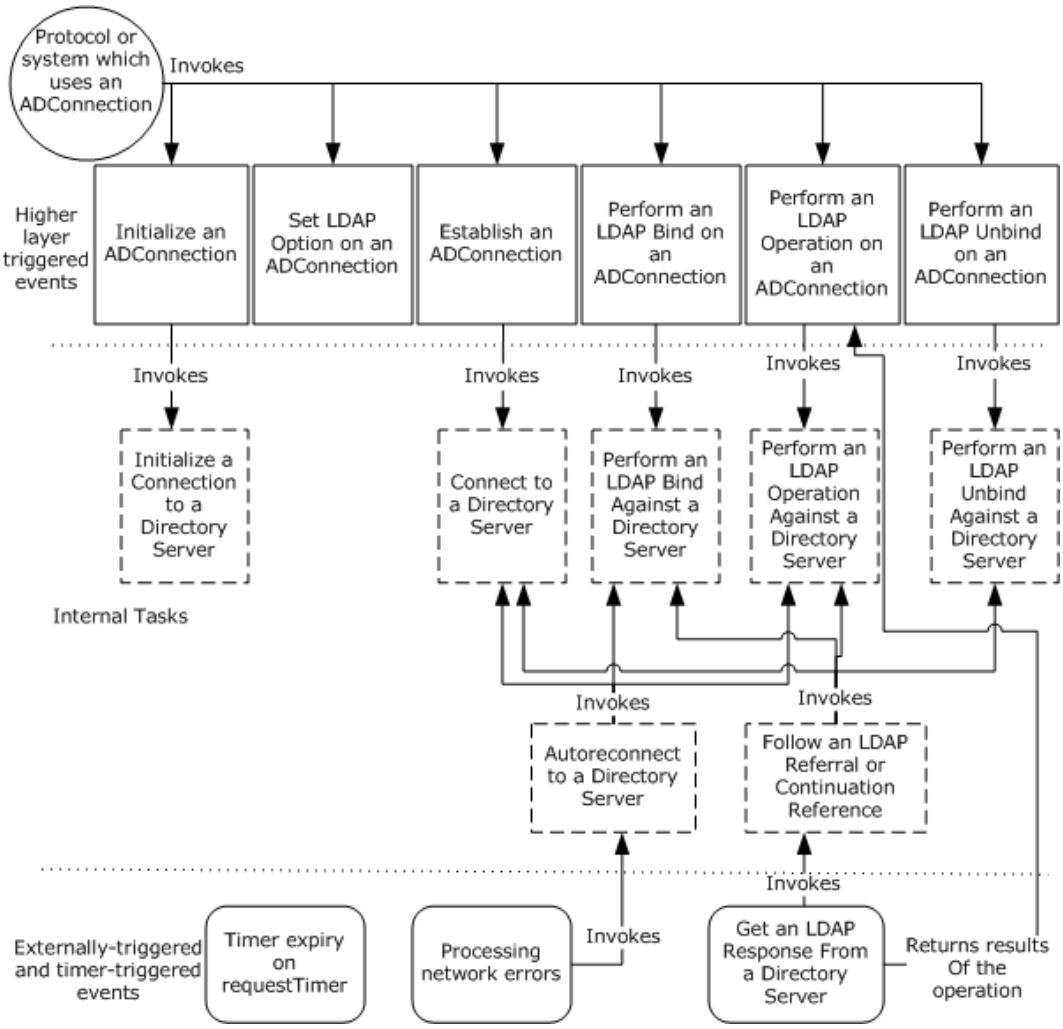

Figure 56: Task relationship diagram

6.2.6.1 Tasks

6.2.6.1.1 Initializing an ADConnection

This task initializes an instance of the **ADConnection** element and returns it to the caller.

The parameters for this task are as follows:

Name	Type	Description	Optional
TaskInputTargetName	String (Unicode)	A string used to locate a domain controller. It may be NULL (indicates that the joined domain name should be used), domain name (DNS/NetBIOS), server host name, or server IP address.	No
TaskInputPortNumber	Unsigned integer	The destination TCP port number ([RFC793] section 1.5, Operation) to use when connecting to the	No

Name	Type	Description	Optional
		directory server specified by *TaskInputTargetName*.	

The task returns the following results to the caller:

Name	Type	Description
TaskReturnADConnection	**ADCONNECTION_HANDLE**	An **ADCONNECTION_HANDLE** which refers to an instance of the **ADConnection** ADM element.

The task performs the following actions:

1. Create an instance of the **ADConnection** ADM structure and initialize the values in the ADM to their default values. This instance is added to the list **ADCList**.
2. Invoke the Initializing a Connection to a Directory Server (section 6.2.6.2.1) task, passing the *TaskInputTargetName* and *TaskInputPortNumbers* parameters provided by the caller of this task. The returned TaskReturnConnectionInfo is assigned to **ADConnection.primaryConnection**.
3. Return an ADCONNECTION_HANDLE ([MS-DTYP] section 2.2.2, ADCONNECTION_HANDLE) referring to the **ADConnection** instance to the caller as TaskReturnADConnection.

6.2.6.1.2 Setting an LDAP Option on an ADConnection

This task sets one of the LDAP_OPT_<optionName> ADM elements on an **ADConnection**.

The parameters for this task are as follows:

Name	Type	Description	Optional
TaskInputADConnection	**ADCONNECTION_HANDLE**	An **ADCONNECTION_HANDLE** which refers to an initialized **ADConnection** on which to set an option.	No
TaskInputOptionName	String (Unicode)	The name of the LDAP_OPT ADM element to set.	No
TaskInputOptionValue	MUST match the type of the ADM element specified by *TaskInputOptionName*.	The value to assign to the specified ADM element.	No

The task returns the following results to the caller:

- This task does not return any results.

The task performs the following actions:

1. Let adConnection be the **ADConnection** instance in **ADCList** referred to by *TaskInputADConnection*.
2. Set the ADM element from adConnection with name *TaskInputOptionName* to the value *TaskInputOptionValue*.

6.2.6.1.3 Establishing an ADConnection

This task establishes a TCP connection to an Active Directory server specified by the elements of an input **ADConnection** instance.

The parameters for this task are as follows:

Name	Type	Description	Optional
TaskInputADConnection	**ADCONNECTION_HANDLE**	An **ADCONNECTION_HANDLE** which refers to an initialized **ADConnection** for which a TCP connection to the directory server will be established.	No

The task returns the following results to the caller:

Name	Type	Description
TaskReturnStatus	Boolean	Returns TRUE if a TCP connection was successfully established and FALSE otherwise.

The task performs the following actions:

1. Let adConnection be the **ADConnection** instance in **ADCList** referred to by *TaskInputADConnection*.
2. Invoke the Connecting to a Directory Server (section 6.2.6.2.2) Task, passing adConnection.primaryConnection as the *TaskInputConnectionInfo* parameter. The return value of the invocation is returned to the caller as TaskReturnStatus.

6.2.6.1.4 Performing an LDAP Bind on an ADConnection

This task authenticates the client to an Active Directory server specified by the elements of an input **ADConnection** instance.

The parameters for this task are as follows:

Name	Type	Description	Optional
TaskInputADConnection	**ADCONNECTION_HANDLE**	An **ADCONNECTION_HANDLE** that refers to an initialized **ADConnection** on which to perform an LDAP bind.	No

The task returns the following results to the caller:

Name	Type	Description
TaskReturnStatus	Unsigned Integer	The LDAP resultCode ([RFC2251] section 4.1.10, Result Message) returned from the directory server in response to the bind request or an error indicating that the directory server could not be contacted.

The task performs the following actions:

1. Let adConnection be the **ADConnection** instance in **ADCList** referred to by *TaskInputADConnection*.

2. Invoke the Performing an LDAP Bind Against a Directory Server (section 6.2.6.2.3) Task with the following parameters: *TaskInputConnectionInfo* is set to adConnection.primaryConnection. This task returns the resulting TaskReturnStatus.

6.2.6.1.5 Performing an LDAP Unbind on an ADConnection

This task closes an **ADConnection's primaryConnection** and any **referralConnections**.

The parameters for this task are as follows:

Name	Type	Description	Optional
TaskInputADConnection	**ADCONNECTION_HANDLE**	An **ADCONNECTION_HANDLE** which refers to an initialized **ADConnection** that the client will close.	No

The task returns the following results to the caller:

- This task does not return any results.

The task performs the following actions:

1. Let adConnection be the **ADConnection** instance in **ADCList** referred to by *TaskInputADConnection*.

2. Invoke the Performing an LDAP Unbind Against a Directory Server (section 6.2.6.2.4) task with the following parameters: *TaskInputConnectionInfo* is set to adConnection.primaryConnection.

3. For each **connectionInfo** in adConnection.referralConnections:

   - Invoke the Performing an LDAP Unbind Against a Directory Server (section 6.2.6.2.4) task with the following parameters: *TaskInputConnectionInfo* is set to **connectionInfo**.

4. Set adConnection.primaryConnection to NULL.

5. Set adConnection.referralConnections to NULL.

6. Remove adConnection from **ADCList**.

6.2.6.1.6 Performing an LDAP Operation on an ADConnection

This task sends an LDAP request to an Active Directory server.

The parameters for this task are as follows:

Name	Type	Description	Optional
TaskInputADConnection	**ADCONNECTION_HANDLE**	An **ADCONNECTION_HANDLE** which refers to an initialized **ADConnection** on which to send the LDAP request.	No

Name	Type	Description	Optional
TaskInputRequestMessage	LDAPMessage	The request to send to the directory server.	No
TaskOutputResultMessages	List of LDAPMessage	The response from the directory server.	No

The task returns the following results to the caller:

Name	Type	Description
TaskReturnStatus	Unsigned Integer	The LDAP resultCode ([RFC2251] section 4.1.10, Result Message) returned from the directory server in response to the request or an error indicating that the directory server could not be contacted or a timeout has occurred.

The task performs the following actions:

1. Let adConnection be the **ADConnection** instance in **ADCList** referred to by *TaskInputADConnection*.
2. Let ldapRequest be a freshly constructed instance of the **LDAPRequest** ADM structure type defined in section 6.2.3, ADConnection Abstract Data Model, initialized with default values as specified in section 6.2.3, ADConnection Abstract Data Model.
3. ldapRequest.requestMessage is set to **TaskInputRequestMessage**, an input parameter to the current task that indicates the request to be sent to the directory server.
4. The current task invokes the task, Performing an LDAP Operation Against a Directory Server (section 6.2.6.2.5), with the following parameters: *TaskInputConnectionInfo* is set to adConnection.primaryConnection and *TaskInputRequestMessage* is set to ldapRequest.
5. The current task waits for responses to arrive on ldapRequest.resultMessages. When the responses for the request have been received (see Getting an LDAP Response from a Directory Server (section 6.2.6.3.2)), *TaskOutputResultMessages* is set to ldapRequest.resultMessages. If no error is encountered, the LDAP resultCode of the last message in *TaskOutputResultMessages* is returned to the caller. Otherwise the error is returned.

6.2.6.2 Internal Tasks

The tasks described in the following sections are supporting tasks for the management of **ADConnections** and are internal to this document; they are invoked only by the other tasks and events described in subsections under section 6.2.6.

6.2.6.2.1 Initializing a Connection to a Directory Server

This task initializes an instance of the **ConnectionInfo** element.

The parameters for this task are as follows:

Name	Type	Description	Optional
TaskInputTargetName	string (Unicode)	A string used to locate a domain controller. It may be NULL (which indicates that the joined domain name should be used), the domain name (DNS/NetBIOS),	No

Name	Type	Description	Optional
		the server host name, or the server IP address.	
TaskInputPortNumber	Unsigned integer	The destination TCP port number ([RFC793] section 1.5, Operation) to use when connecting to the directory server specified by *TaskInputTargetName*.	No

The task returns the following results to the caller:

Name	Type	Description
TaskReturnConnectionInfo	ConnectionInfo	An instance of the **ADConnectionInfo** ADM element.

The task performs the following actions:

1. An instance of the **ConnectionInfo** ADM element is created and the values in the ADM are initialized to their default values. The default value of **ConnectionInfo.networkConnection** causes a new TCP connection to be used that is not associated with an existing **ADConnection**.
2. **ConnectionInfo.targetName** is set to the input parameter *TaskInputTargetName*.
3. **ConnectionInfo.portNumber** is set to the input parameter *TaskInputPortNumber*.
4. The **ConnectionInfo** instance is returned to the caller as TaskReturnConnectionInfo.

6.2.6.6.2.2 Connecting to a Directory Server

This task establishes a TCP connection to an Active Directory server specified by the elements of an input **ConnectionInfo** instance.

The parameters for this task are as follows:

Name	Type	Description	Optional
TaskInputConnectionInfo	**ConnectionInfo**	An initialized **ConnectionInfo** on which to perform an LDAP bind.	No

The task returns the following results to the caller:

Name	Type	Description
TaskReturnStatus	Boolean	This task returns TRUE if a TCP connection was successfully established, FALSE otherwise.

The task performs the following actions:

1. Let connectionSuccessful be a Boolean initialized to FALSE.
2. Let dcAddress be a Unicode string initialized to NULL.
3. Let containingADConnection be the **ADConnection** instance contained in **ADCList**, which either has **containingADConnection.primaryConnection** set to *TaskInputConnectionInfo* or contains *TaskInputConnectionInfo* in the **containingADConnection.referralConnections** list. Note that these conditions are mutually exclusive and there is exactly one **ADConnection** which satisfies these conditions.

4. If **TaskInputConnectionInfo.targetName** is NULL:

   - If **ComputerRole** is DSRole_RoleBackupDomainController or DsRole_RolePrimaryDomainController, then set dcAddress to "localhost" and go to step 8; otherwise go to step 7.

5. If **TaskInputConnectionInfo.targetName** is an IP address, then go to step 9.

6. If **containingADConnection.LDAP_OPT_AREC_EXCLUSIVE** is set to TRUE, then go to step 8, skipping the DC location process described in step 7.

7. Assume that **TaskInputConnectionInfo.targetName** represents a domain name and attempt to locate a domain controller in the specified domain:

   1. Let domainControllerInfo be an instance of the **DOMAIN_CONTROLLER_INFOW** structure ([MS-NRPC] section 2.2.1.2.1, DOMAIN_CONTROLLER_INFOW).

   2. Let addedFlags be an unsigned integer. If **TaskInputConnectionInfo.portNumber** is 3268 or 3269, addedFlags is set to the bitwise OR of the D, M, R flags defined for the *Flags* parameter in [MS-NRPC] section 3.5.4.3.1, DsrGetDcNameEx2. Otherwise addedFlags is set to the bitwise OR of the M, R flags.

   3. The **DsrGetDcName** method ([MS-NRPC] section 3.5.4.3.3, DsrGetDcName) is invoked with the following parameters: *ComputerName* is NULL, *DomainName* is *TaskInputConnectionInfo.targetName*, *DomainGuid* is NULL, *SiteGuid* is NULL, *Flags* is the bitwise OR of **containingADConnection.LDAP_OPT_GETDSNAME_FLAGS**. and addedFlags, *DomainControllerInfo* is a pointer to domainControllerInfo.

   4. If the invocation of **DsrGetDcName** listed in step 3 returned 0 (Success), then:

      - **TaskInputConnectionInfo.targetName** specified a **domain name** and **domainControllerInfo.DomainControllerAddress** now identifies a domain controller in the specified domain; if **domainControllerInfo.DomainControllerAddress** is an IP address, set dcAddress to **domainControllerInfo.DomainControllerAddress** with the "\\" prefix omitted.

   5. If the invocation of **DsrGetDcName** listed in step 3 returned a non-zero value and **TaskInputConnectionInfo.targetName** is NULL, then this task returns FALSE.

8. If dcAddress is NULL, then assume that **TaskInputConnectionInfo.targetName** is a host name, and set dcAddress to **TaskInputConnectionInfo.targetName**.

9. A TCP connection is established to the server whose host name or IP address is specified by dcAddress, with destination port set to **TaskInputConnectionInfo.portNumber**. If dcAddress is a host name, gethostbyname ([MS-WSO] section 3.1.1.7.2, Invocation of Name Resolution Protocols in Windows) is invoked and each of the returned IP addresses is tried in parallel until a connection returns successfully or all IP addresses returned by gethostbyname have been exhausted. If a TCP connection is successfully established, the client sets **TaskInputConnectionInfo.networkConnection** to the TCP connection and connectionSuccessful is set to TRUE.

10. If connectionSuccessful is TRUE:

    1. If **containingADConnection.LDAP_OPT_TCP_KEEPALIVE** is TRUE, the client enables TCP Keep-alives (See [RFC1122] section 4.2.3.6, TCP Keep-Alives) on **TaskInputConnectionInfo.networkConnection**.

2. This task returns TRUE.

11. This task returns FALSE.

In addition to the above, if the LDAP client is unable to establish a TCP connection to an IP address obtained from DC Location (step 7), it will retry DC Location once, this time including the "A" flag in the *Flags* parameter passed to **DsrGetDcName** (in addition to whichever flags were passed in during the first DC Location attempt). It will then try establishing a TCP connection to the IP address obtained. If this fails, the task will return FALSE.

6.2.6.2.3 Performing an LDAP Bind Against a Directory Server

This task authenticates the client to an Active Directory server.

The parameters for this task are as follows:

Name	Type	Description	Optional
TaskInputConnectionInfo	**ConnectionInfo**	An initialized **ConnectionInfo** on which an LDAP bind is to be performed. The **ConnectionInfo** could have been used already to connect or bind to a directory server.	No

The task returns the following results to the caller:

Name	Type	Description
TaskReturnStatus	Unsigned Integer	The LDAP resultCode ([RFC2251] section 4.1.10, Result Message) returned from the directory server in response to the bind request or an error indicating that the directory server failed to respond to the request due to network errors or timer expiration.

The task performs the following actions:

1. Let bindRequestMessage be an LDAPMessage with protocolOp of type BindRequest as described in [RFC2251] section 4.2, Bind Operation.

2. Let containingADConnection be the **ADConnection** instance contained in **ADCList** which either has **containingADConnection.primaryConnection** set to *TaskInputConnectionInfo*, or contains *TaskInputConnectionInfo* in the **containingADConnection.referralConnections** list.

3. If **TaskInputConnectionInfo.networkConnection** is NULL, the task: Connecting to a Directory Server is invoked with the following parameters: *TaskInputConnectionInfo* is set to **TaskInputADConnection.primaryConnection**. If the TaskReturnStatus returned by the invocation of Connecting to a Directory Server (section 6.2.6.2.2) is FALSE, this task returns the error code 81 indicating the directory server could not be contacted.

4. Construct bindRequestMessage from **containingADConnection.LDAP_OPT_AUTH_INFO**, according to [MS-ADTS] section 5.1.1, Authentication.

5. If the bind method specified by **containingADConnection.LDAP_OPT_AUTH_INFO** is SASL:

    1. Set **containingADConnection.LDAP_OPT_PROTOCOL_VERSION** to 3.

2. If **containingADConnection.LDAP_OPT_SIGN** is TRUE, set bindRequestMessage to request SASL-layer integrity. See [MS-ADTS] section 5.1.2.1, Using SASL.

3. If **containingADConnection.LDAP_OPT_ENCRYPT** is TRUE, set bindRequestMessage to request SASL-layer confidentiality. See [MS-ADTS] section 5.1.2.1, Using SASL.

4. If **containingADConnection.LDAP_OPT_DNSDOMAIN_NAME** is not NULL, then the client uses **containingADConnection.LDAP_OPT_DNSDOMAIN_NAME** as the 3rd part of the 3-part Service Principal Name (SPN) supplied to the security packages which authenticate the client with the Active Directory server ([RFC2251] and [MS-ADTS] section 5.1.1, Authentication).

5. The Bind LDAP processing will invoke security packages to authenticate the client with the Active Directory server ([RFC2251] and [MS-ADTS] section 5.1.1, Authentication). Those security packages will take the security identity of the current thread of execution as the identity.<4>

6. Let ldapRequest be a freshly constructed instance of the **LDAPRequest** ADM structure with the values in the ADM initialized to their default values.

7. **ldapRequest.requestMessage** is set to bindRequestMessage.

8. The client invokes the Performing an LDAP Operation Against a Directory Server (section 6.2.6.2.5) task with the following parameters: *TaskIputRequestMessage* is set to ldapRequest and *TaskInputConnectionInfo* is set to the passed *TaskInputConnectionInfo*.

9. The task waits for responses to arrive on **ldapRequest.resultMessages**. When the responses for the request have been received (see task: Getting an LDAP Response from a Directory Server (section 6.2.6.3.2)), if the responses indicate success, **TaskInputConnectionInfo.bindHasHappened** is set to TRUE. This task returns the LDAP resultCode from the last response.

6.2.6.2.4 Performing an LDAP Unbind Against a Directory Server

This task closes an **ADConnection's primaryConnection** and any **referralConnections**.

The parameters for this task are as follows:

Name	Type	Description	Optional
TaskInputConnectionInfo	**ConnectionInfo**	An initialized **ConnectionInfo** that the client will close.	No

The task returns the following results to the caller:

- This task does not return any results.

The task performs the following actions:

1. Let unbindRequest be an LDAPMessage for an unbind request. See [RFC2251] section 4.3, Unbind Operation.

2. Let ldapRequest be a freshly constructed instance of the **LDAPRequest** ADM structure with the values in the ADM initialized to their default values.

3. **ldapRequest.requestMessage** is set to unbindRequest.

4. Invoke the Performing an LDAP Operation Against a Directory Server (section 6.2.6.2.5) task with the following parameters: *TaskInputConnectionInfo* is set to the passed *TaskInputConnectionInfo* and *TaskInputLdapMessage* is set to ldapRequest.

5. The client closes the TCP connection **TaskInputConnectionInfo.networkConnection**.

6.2.6.2.5 Performing an LDAP Operation Against a Directory Server

This task sends an LDAP request to an Active Directory server.

The parameters for this task are as follows:

Name	Type	Description	Optional
TaskInputConnectionInfo	ConnectionInfo	An initialized **ConnectionInfo** on which to send the LDAP request.	No
TaskInputRequestMessage	LDAPRequest	The request to send to the directory server, including any **ExtendedRequest** as documented in [MS-ADTS] section 3.1.1.3.4.2, LDAP Extended Operations.	No

The task returns the following results to the caller:

Name	Type	Description
TaskReturnStatus	Boolean	This task returns TRUE if the client successfully sends the request to the directory server. Otherwise, it returns FALSE.

The task performs the following actions:

1. Let ldapRequest be the **LDAPRequest** instance received in *TaskInputRequestMessage*.

2. Let containingADConnection be the **ADConnection** instance contained in **ADCList**, which either has **containingADConnection.primaryConnection** set to *TaskInputConnectionInfo* or contains *TaskInputConnectionInfo* in the **containingADConnection.referralConnections** list.

3. Let operationTimeout be an unsigned integer initialized to the value of **containingADConnection.LDAP_OPT_TIMELIMIT**.

4. If operationTimeout is 0 and **TaskInputRequestMessage.requestMessage.protocolOp** is of type BindRequest, operationTimeout is set to 120.

5. If **TaskInputRequestMessage.requestMessage.protocolOp** is of type SearchRequest:

    1. If **TaskInputRequestMessage.requestMessage.protocolOp.sizeLimit** was not specified, then it is set to **containingADConnection.LDAP_OPT_SIZELIMIT**.

    2. If **TaskInputRequestMessage.requestMessage.protocolOp.timeLimit** was not specified, then it is set to operationTimeout.

6. ldapRequest.requestMessage.messageID is set, as described in [RFC2251] section 4.1.1.1, Message ID.

7. The client sends ldapRequest to the directory server via **TaskInputConnectionInfo.networkConnection**. If the send is successful:

1. ldapRequest.requestTimer is set to operationTimeout and begins counting down.

2. ldapRequest is appended to **TaskInputConnectionInfo.pendingRequestList**, if not already present.

3. This task returns TRUE.

8. This task returns FALSE.

6.2.6.2.6 Following an LDAP Referral or Continuation Reference

This task follows an LDAP referral or continuation reference ([RFC2251] sections 4.1.11, Referral, and 4.5.3, Continuation References in the Search Result, and [MS-ADTS] section 3.1.1.3.1.4, Referrals in LDAPv2 and LDAPv3).

The parameters for this task are as follows:

Name	Type	Description	Optional
TaskInputLdapRequest	LDAPRequest	The request which originally generated this referral or continuation reference.	No
TaskInputReferralUrl	LDAP URL	An LDAP URL (see [RFC2255]) that was returned by the server.	No

The task returns the following results to the caller:

Name	Type	Description
TaskReturnStatus	Boolean	This task returns TRUE if the client successfully sends the referral request to the directory server. Otherwise, it returns FALSE.

The task performs the following actions:

1. Let containingConnectionInfo be the **ConnectionInfo** instance which contains *TaskInputLdapRequest* in **containingConnectionInfo.pendingRequestList**.

2. Let containingADConnection be the **ADConnection** instance contained in **ADCList** which either has **containingADConnection.primaryConnection** set to containingConnectionInfo, or contains containingConnectionInfo in the **containingADConnection.referralConnections** list.

3. If **TaskInputLdapRequest.numReferrals** is greater than or equal to **containingADConnection.LDAP_OPT_REFERRAL_HOP_LIMIT**, return FALSE.

4. Let newServer be a string initialized to the host portion of the hostport element of TaskInputReferralUrl, or NULL if there is no host portion. See [RFC2255].

5. Let newPort be an unsigned integer initialized to the port portion of the hostport element of TaskInputReferralUrl, or 389 if there is no port portion. See [RFC2255].

6. Let newConnectionInfo be the result of invoking the Initializing a Connection to a Directory Server (section 6.2.6.2.1) task with the following parameters: *TaskInputTargetName* is set to newServer and *TaskInputPortNumber* is set to newPort.

7. Append newConnectionInfo to **containingADConnection.referralConnections**.

8. Invoke the Connecting to a Directory Server (section 6.2.6.2.2) task with the following parameter: *TaskInputConnectionInfo* is set to newConnectionInfo. If the invocation returned FALSE, remove newConnectionInfo from **containingADConnection.referralConnections** and this task returns FALSE.

9. If **containingConnectionInfo.bindHasHappened** is TRUE:

   - Invoke the Performing an LDAP Bind Against a Directory Server (section 6.2.6.2.3) task with the following parameter: *TaskInputConnectionInfo* is set to newConnectionInfo. If this invocation returned FALSE, newConnectionInfo is removed from **containingADConnection.referralConnections** and this task returns FALSE.

10. The fields of **TaskInputLdapRequest.requestMessage** are modified, for example setting the dn or filter, based on TaskInputReferralUrl according to the rules in [RFC2255] section 5, URL Processing, and [RFC2251] sections 4.1.1, Message Envelope, and 4.5.3, Continuation References in the Search Result.

11. Invoke the Performing an LDAP Operation Against a Directory Server (section 6.2.6.2.5) task with the following parameters: *TaskInputConnectionInfo* is set to newConnectionInfo and *TaskInputRequestMessage* is set to **TaskInputLdapRequest**.

12. Invoke the Performing an LDAP Unbind Against a Directory Server (section 6.2.6.2.4) task with the following parameter: *TaskInputConnectionInfo* is set to newConnectionInfo.

13. Remove newConnectionInfo from **containingADConnection.referralConnections**.

14. If the invocation of Performing an LDAP Operation Against a Directory Server (section 6.2.6.2.5) task in step 11 returned TRUE, then **TaskInputLdapRequest.numReferrals** is incremented and this task returns TRUE. Otherwise this task returns FALSE.

6.2.6.2.7 Autoreconnecting to a Directory Server

This task reconnects to a directory server when network errors are encountered on a connection.

The parameters for this task are as follows:

Name	Type	Description	Optional
TaskInputConnectionInfo	ConnectionInfo	An initialized **ConnectionInfo** on which an autoreconnect is to be performed.	No

The task returns the following results to the caller:

Name	Type	Description
TaskReturnStatus	Boolean	This task returns TRUE if the client successfully reconnects to a directory server. Otherwise, it returns FALSE.

The task performs the following actions:

1. Let wasBound be a Boolean set equal to the value of **TaskInputConnectionInfo.bindHasHappened**.

2. Set **TaskInputConnectionInfo.isAutoReconnecting** to TRUE.

3. Set **TaskInputConnectionInfo.bindHasHappened** to FALSE.

4. Invoke the Connecting to a Directory Server (section 6.2.6.2.2) task with the following parameters: *TaskInputConnectionInfo* is set to the passed-in *TaskInputConnectionInfo*. If this invocation returned FALSE, then return FALSE from this task.

5. If wasBound is TRUE, then invoke the Performing an LDAP Bind Against a Directory Server (section 6.2.6.2.3) task with the following parameters: *TaskInputConnectionInfo* is set to the passed-in *TaskInputConnectionInfo*. If this invocation returned an error, then return FALSE from this task.

6. For each LDAPRequest lrq in **TaskInputConnectionInfo.pendingRequestList** for which lrq.resultMessages is empty:

    1. Increment lrq.numResends by 1.

    2. If lrq.numResends is greater than 20 or if lrq.requestMessage is a SearchRequest ([RFC2251] section 4.5.1, Search Request) containing the LDAP_SERVER_NOTIFICATION_OID control ([MS-ADTS] section 3.1.1.3.4.1.9, LDAP_SERVER_NOTIFICATION_OID), then construct an LDAPMessage, lm, containing an LDAPResult, lr, representing a response for lrq.requestMessage, indicating that the directory server was unreachable:

        - Set **lm.messageID** to **lrq.requestMessage.messageID**.
        - Set **lr.resultCode** to a local implementation-specific error code <5> indicating that the directory server was unreachable, using an error code value reserved for APIs as specified in [RFC2251] section 4.1.10, Result Message.
        - Set **lr.errorMessage** and **lr.matchedDN** to a zero-length string.
        - Insert the LDAPMessage lm into **lrq.resultMessages**.

    3. Otherwise:

        1. Invoke the Performing an LDAP Operation Against a Directory Server (section 6.2.6.2.5) task with the following parameters: *TaskInputConnectionInfo* is the passed-in *TaskInputConnectionInfo*, and *TaskInputRequestMessage* is lrq.

        2. If the above invocation returned FALSE, then construct an LDAPMessage, lm, containing an LDAPResult, lr, representing a response for lrq.requestMessage, indicating that the directory server was unreachable:

            - Set **lm.messageID** to **lrq.requestMessage.messageID**.
            - Set **lr.resultCode** to a local implementation-specific error code <6> indicating that the directory server was unreachable, using an error code value reserved for APIs as specified in [RFC2251] section 4.1.10, Result Message.
            - Set **lr.errorMessage** and **lr.matchedDN** to a zero-length string.
            - Insert the LDAPMessage lm into **lrq.resultMessages**.

7. Set **TaskInputConnectionInfo.isAutoReconnecting** to FALSE.

8. Return TRUE.

6.2.6.3 External Triggered Events

6.2.6.3.1 Processing Network Errors

This event is triggered, as described in sections 6.2.4 and 6.2.5, when a network read or write returns an error code indicating an underlying network failure such as media disconnection, host unreachable, route unavailable, or the TCP keepalive failed, or when there is no response to too many consecutive ICMP echo requests.

The parameters for this task are as follows:

Name	Type	Description	Optional
TaskInputConnectionInfo	ConnectionInfo	The **ConnectionInfo** on which the error occurred.	No

The task returns the following results to the caller:

- This task does not return any results.

The task performs the following actions:

1. Let containingADConnection be the **ADConnection** instance contained in **ADCList** which either has **containingADConnection.primaryConnection** set to *TaskInputConnectionInfo*, or contains *TaskInputConnectionInfo* in the **containingADConnection.referralConnections** list.
2. Close the TCP connection represented by **TaskInputConnectionInfo.networkConnection**.
3. If **containingADConnection.LDAP_OPT_AUTO_RECONNECT** is FALSE, then for each LDAPRequest lrq in **TaskInputConnectionInfo.pendingRequestList** which is awaiting a response:
   - Construct an LDAPMessage, lm, containing an LDAPResult, lr, representing a response for lrq.requestMessage, indicating that the directory server was unreachable:
     - Set **lm.messageID** to **lrq.requestMessage.messageID**.
     - Set **lr.resultCode** to a local implementation-specific error code <7> indicating that the directory server was unreachable, using an error code value reserved for APIs as specified in [RFC2251] section 4.1.10, Result Message.
     - Set **lr.errorMessage** and **lr.matchedDN** to a zero-length string.
     - Insert the LDAPMessage lm into **lrq.resultMessages**.
4. If **containingADConnection.LDAP_OPT_AUTO_RECONNECT** is TRUE, then invoke the Autoreconnecting to a Directory Server (section 6.2.6.2.7) task on *TaskInputConnectionInfo*.

6.2.6.3.2 Getting an LDAP Response from a Directory Server

This event occurs when the client receives an LDAPMessage from a directory server.

The parameters for this task are as follows:

Name	Type	Description	Optional
TaskInputLdapResultMessage	LDAPMessage	The LDAPMessage received from the directory server.	No

The task returns the following results to the caller:

- This task does not return any results.

The task performs the following actions:

1. Let originalRequest be the LDAPRequest instance for which the **originalRequest.requestMessage.messageID** is equal to **TaskInputLdapResultMessage.messageID**. If there is no such match because of a timeout (see section 6.2.6.4.1), the task ends.

2. Let containingConnectionInfo be the **ConnectionInfo** instance which contains originalRequest in **containingConnectionInfo.pendingRequestList**.

3. Let containingADConnection be the **ADConnection** instance contained in **ADCList** which either has **containingADConnection.primaryConnection** set to containingConnectionInfo, or contains containingConnectionInfo in the **containingADConnection.referralConnections** list. Because there is a unique networkConnection, and therefore a TCP connection, associated with each **ADConnection**, there will be only one **ADConnection** that matches the criteria in this step.

4. If **TaskInputLdapResultMessage.protocolOp** is an LDAPResult and **TaskInputLdapResultMessage.protocolOp.resultCode** is referral ([RFC2251] section 4.1.10 in the case of LDAP v3) or is the value 9 ([MS-ADTS] section 3.1.1.3.1.4, Referrals in LDAPv2 and LDAPv3, in the case of LDAP v2) and **containingADConnection.LDAP_OPT_REFERRALS** is "ON" or "LDAP_CHASE_REFERRALS":

    1. Let urlList be a list of LDAP URLs ([RFC2255]) contained in **TaskInputLdapResultMessage.protocolOp.referral** ([RFC2251] section 4.1.10, Result Message, in the case of LDAP v3) or in **TaskInputLdapResultMessage.protocolOp.errorMessage** ([MS-ADTS] section 3.1.1.3.1.4, Referrals in LDAPv2 and LDAPv3, in the case of LDAP v2).

    2. For each ldapUrl in urlList:

        - Invoke the task Following an LDAP Referral or Continuation Reference (section 6.2.6.2.6) with the following parameters: *TaskInputLdapRequest* is set to originalRequest and *TaskInputReferralUrl* is set to ldapUrl. If this invocation returns TRUE, this task returns.

5. If **TaskInputLdapResultMessage.protocolOp** is a SearchResultReference and **containingADConnection.LDAP_OPT_REFERRALS** is "ON" or "LDAP_CHASE_CONTINUATION_REFERENCES":

    1. Let urlList be a list of LDAP URLs ([RFC2255]) contained in **TaskInputLdapResultMessage.protocolOp**.

    2. For each ldapUrl in urlList:

        - Invoke the task Following an LDAP Referral or Continuation Reference (section 6.2.6.2.6) with the following parameters: *TaskInputLdapRequest* is set to originalRequest and *TaskInputReferralUrl* is set to ldapUrl. If this invocation returns TRUE, this task returns.

6. Append *TaskInputLdapResultMessage* to **originalRequest.resultMessages**.

7. At this point there is either an LDAPResult or the next search response. If **TaskInputLdapResultMessage.protocolOp** is an LDAPResult, then originalRequest is removed from **containingConnectionInfo.pendingRequestList** and any tasks waiting on results for originalRequest proceed. Otherwise these tasks continue to wait for additional responses.

6.2.6.4 Timer Triggered Events

6.2.6.4.1 Timer Expiry on RequestTimer

This event occurs whenever the **requestTimer** on an instance of an LDAPRequest expires.

The parameters for this task are as follows:

Name	Type	Description	Optional
TaskInputLdapRequest	LDAPRequest	The LDAPRequest instance on which the **requestTimer** has expired.	No

The task returns the following results to the caller:

- This task does not return any results.

The task performs the following actions:

1. For each containingADConnection in **ADCList**:

    1. If *TaskInputLdapRequest* is present in **containingADConnection.primaryConnection.pendingRequestList**, remove *TaskInputLdapRequest* from **containingADConnection.primaryConnection.pendingRequestList**.

    2. For each containingConnectionInfo in **containingADConnection.referralConnections**:

        - If *TaskInputLdapRequest* is present in **containingConnectionInfo.pendingRequestList**, remove *TaskInputLdapRequest* from **containingConnectionInfo.pendingRequestList**.

2. Construct an LDAPMessage, lm, containing an LDAPResult, lr, representing a response for **TaskInputLdapRequest.requestMessage**, indicating that the request has timed out:

    - Set **lm.messageID** to **TaskInputLdapRequest.requestMessage.messageID**.

    - Set **lr.resultCode** to a local implementation-specific error code<8> indicating a timeout has occurred, using an error code value reserved for APIs as specified in [RFC2251] section 4.1.10, Result Message.

    - Set **lr.errorMessage** and **lr.matchedDN** to a zero-length string.

    - Insert the LDAPMessage lm into **TaskInputLdapRequest.resultMessages**.

    Any tasks waiting on results for TaskInputLdapRequest receive the LDAPMessage lm in **TaskInputLdapRequest.resultMessages**, indicating that the request has timed out.

6.2.7 LDAP Over UDP

Windows uses LDAP over UDP as defined in [RFC1798] for LDAP versions 2 and 3. The following sections describe only the additional behaviors of the Microsoft LDAP client which are not specified by [RFC1798].

6.2.7.1 ADUDPHandle Overview

For LDAP over UDP, an **ADUDPHandle** represents the parameters used for communication between the client and Active Directory servers. The typical sequence of use of an **ADUDPHandle** is:

1. Initialize an **ADUDPHandle**, which allocates an **ADUDPHandle**. This step does not perform any network operations.

2. Perform one LDAP operation, which is either a rootDSE search or an LDAP abandon operation, on the **ADUDPHandle**. An LDAP operation will consist of an LDAP request and the resulting LDAP response.

The only tasks which use the **ADUDPHandle** type are those described in section 6.2.7.3.1, Initializing an ADUDPHandle, and section 6.2.7.3.2, Performing an LDAP Operation on an ADUDPHandle.

6.2.7.2 ADUDPHandle Abstract Data Model

This section describes a conceptual model of a possible data organization that an implementation maintains. The described organization is provided to facilitate the explanation of how the client behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

The data model defines a primary structure **ADUDPHandle** that the client operates on.

ComputerRole: An abstract variable of type **DSROLE_MACHINE_ROLE** that indicates the current domain membership role of the machine on which the LDAP client is executing as described in [MS-DSSP] section 3.2.1, Abstract Data Model.

ADUDPHandle: A structure that holds the information necessary to communicate with an Active Directory server over the UDP protocol.

- **protocolVersion**: An unsigned integer indicating which version of the LDAP protocol the connection uses. Valid values are 2 and 3. The default value is 3.

- **portNumber**: An unsigned integer indicating the UDP destination port number to use (see [RFC768]).

- **targetName**: A null-terminated string used to locate a directory server. It can be NULL, indicating that the directory server for the joined domain should be located, a domain name (DNS/NetBIOS), a host name, or an IP address.

6.2.7.3 Tasks

6.2.7.3.1 Initializing an ADUDPHandle

This task initializes an instance of **ADUDPHandle** and returns it to the caller.

The parameters for this task are as follows:

Name	Type	Description	Optional
TaskInputTargetName	String (Unicode)	A string used to locate a domain controller. It may be NULL (indicates that the joined domain name should be used), domain name (DNS/NetBIOS), server host name, or server IP address.	No
TaskInputPortNumber	Unsigned integer	The destination UDP port number ([RFC768]) to use when connecting to the directory server specified by *TaskInputTargetName*.	No
TaskInputProtocolVersion	Unsigned integer	Version of the LDAP protocol used. Valid values are 2 and 3. The default value is 3.	Yes

The task returns the following results to the caller:

Name	Type	Description
TaskReturnADUDPHandle	LDAP_UDP_HANDLE	A void pointer which refers to an instance of the **ADUDPHandle**.

The task performs the following actions:

1. Create an instance adUDPHandle of the **ADUDPHandle** ADM structure and initialize the values in the ADM to their default values.
2. Set the **adUDPHandle.portNumber** field to *TaskInputPortNumber*, the **adUDPHandle.protocolVersion** field to *TaskInputProtocolVersion*, and the **adUDPHandle.targetName** to *TaskInputTargetName*.
3. Return an LDAP_UDP_HANDLE referring to the **ADUDPHandle** instance to the caller as *TaskReturnADUDPHandle*.

6.2.7.3.2 Performing an LDAP Operation on an ADUDPHandle

This task sends an LDAP request to an Active Directory server and returns the response returned by the server.

The parameters for this task are as follows:

Name	Type	Description	Optional
TaskInputADUDPHandle	LDAP_UDP_HANDLE	Refers to an initialized **ADUDPHandle** used to send the LDAP request.	No
TaskInputRequestMessage	LDAPMessage	The request to send to the directory server.	No
TaskOutputResultMessages	LDAPMessage list	A sequence of LDAPMessage (as defined in [RFC1777] section 4 for LDAP version 2 and [RFC2251] section 4.1.1 for LDAP version 3) representing the results that the client receives in response to *TaskInputRequestMessage*.	No
TaskInputRequestTimeout	Unsigned integer	Time in milliseconds the client waits to receive the response from the server.	No

Name	Type	Description	Optional
		Value 0 indicates that there is no time limit.	

The task returns the following results to the caller:

Name	Type	Description
TaskReturnStatus	Unsigned Integer	The LDAP resultCode ([RFC2251] section 4.1.10) returned from the directory server in response to the request, or an error indicating that the directory server could not be contacted or that a timeout has occurred.

The task performs the following actions:

1. Let adUDPHandle be the **ADUDPHandle** instance referred to by *TaskInputADUDPHandle*.
2. Let dcAddress be a Unicode string initialized to NULL.
3. If **TaskInputADUDPHandle.targetName** is NULL:
   - If **ComputerRole** is DSRole_RoleBackupDomainController or DsRole_RolePrimaryDomainController, then set dcAddress to "localhost" and go to step 6; otherwise go to step 5.
4. If **TaskInputADUDPHandle.targetName** is an IP address, then go to step 6.
5. Assume **TaskInputADUDPHandle.targetName** represents a domain name and attempt to locate a domain controller in the specified domain.
   1. Let domainControllerInfo be an instance of the **DOMAIN_CONTROLLER_INFOW** structure ([MS-NRPC] section 2.2.1.2.1, DOMAIN_CONTROLLER_INFOW).
   2. Let addedFlags be an unsigned integer. It is set to the bitwise OR of the M, R flags defined for the *Flags* parameter in [MS-NRPC] section 3.5.4.3.1, DsrGetDcNameEx2.
   3. The **DsrGetDcName** method ([MS-NRPC] section 3.5.4.3.3, DsrGetDcName) is invoked with the following parameters: *ComputerName* is NULL, *DomainName* is **TaskInputADUDPHandle.targetName**, *DomainGuid* is NULL, *SiteGuid* is NULL, *Flags* is set to addedFlags, *DomainControllerInfo* is a pointer to domainControllerInfo.
   4. If the invocation of the **DsrGetDcName** method listed in step 3 returned 0 (Success), then:
      - **TaskInputADUDPHandle.targetName** specified a domain name and **domainControllerInfo.DomainControllerAddress** now identifies a domain controller in the specified domain; if **domainControllerInfo.DomainControllerAddress** is an IP address, set dcAddress to **domainControllerInfo.DomainControllerAddress** with the "\\" prefix omitted and go to step 6.
   5. If the invocation of **DsrGetDcName** listed in step 3 returned a nonzero value and **TaskInputADUDPHandle.targetName** is NULL, then this task returns a local implementation-specific error code<9> indicating that the directory server was unreachable, using an error code value reserved for APIs as specified in [RFC2251] section 4.1.10, Result Message.

6. If dcAddress is NULL, then assume that **TaskInputADUDPHandle.targetName** is a host name, and set dcAddress to **TaskInputADUDPHandle.targetName**.

7. Let ldapRequest be set to *TaskInputRequestMessage*.

8. ldapRequest.messageID is set as described in [RFC2251] section 4.1.1.1.

9. Let ipAddress be a Unicode string initialized to NULL. If dcAddress is a host name, gethostbyname ([MS-WSO] section 3.1.1.7.2, Invocation of Name Resolution Protocols in Windows) is invoked and ipAddress is set to the first address returned. Otherwise, assume that dcAddress is an IP address and set ipAddress to dcAddress.

10. Let **networkUDPHandle** be an abstract element representing the UDP handle used to perform UDP operations (see [RFC768] sections "User Interface" and "IP interface"). The **networkUDPhandle** is created using ipAddress and **TaskInputADUDPHandle.portNumber** parameters.

11. The client sends ldapRequest to the directory server indicated by ipAddress using **networkUDPHandle**.

12. The client creates a timer for the duration specified by *TaskInputRequestTimeout* and begins counting down.

13. The client then waits for either the UDP response to arrive or the timer to expire.

14. If the timer expires and no response has been received from the directory server:

    - Let ldapResponse be a freshly constructed LDAPMessage containing an LDAPResult ldapResult.
    - Set **ldapResponse.messageID** to ldapRequest.messageID.
    - Set **ldapResult.resultCode** to a local implementation-specific error code<10> indicating a timeout has occurred, using an error code value reserved for APIs as specified in [RFC2251] section 4.1.10, Result Message.
    - Insert the ldapResponse into *TaskOutputResultMessages*.

15. Else, the timer is canceled, and for each LDAPMessage **lm** in the response received from the directory server:

    - If lm.messageID equals TaskInputRequestMessage.messageID, append lm to the *TaskOutputResultMessages*.

16. Return the resultCode of the last message in *TaskOutputResultMessages*.

6.3 Transport Requirements

Clients communicate with the directory service using RPC-based, block-structured, and SOAP protocols. The following tables list the transports used by the Active Directory System for each of the protocols listed in section 2.2. In the tables below, SMB refers to the Server Message Block (SMB) transport. For information on how messages are protected when sent over these transports, see section 7.2.

RPC-based protocols

Protocols	Transport	Transport requirements
DS RPC	TCP	[MS-DRSR] section 2, Message Transport
DSSP	SMB	[MS-DSSP] section 2.1, Transport
SAMR	SMB, TCP	[MS-SAMR] section 2.1, Transport
LSAD	SMB	[MS-LSAD] section 2.1, Transport
LSAT	SMB, TCP	[MS-LSAT] section 2.1, Transport

Block structured protocols

Protocols	Transport	Transport requirements
LDAP (versions 2 and 3)	TCP, UDP	[MS-ADTS] section 2.1, Transport [RFC1798] section 3.1, User Datagram Protocol (UDP) (this is used for both LDAPv2 and LDAPv3) [RFC2251] section 5, Protocol Element Encodings and Transfer [RFC1777] section 3, Mapping Onto Transport Services

The Web Services protocols use SOAP 1.2 [SOAP1.2-1/2003] as the message format over the "net.tcp" transport defined in [MC-NMF]. WS-Addressing is bound to SOAP 1.2 as described in [WSASB].

SOAP-based protocols

Protocols	Transport	Transport requirements
WS-Transfer	net.tcp	[MS-ADDM] section 2.1, Endpoints [WSASB] [SOAP1.2-1/2003] [MC-NMF]
WS-Enumeration	net.tcp	[MS-ADDM] section 2.1, Endpoints [WSASB] [SOAP1.2-1/2003] [MC-NMF]
ADCAP	net.tcp	[MS-ADCAP] section 2.1, Transport [MS-ADDM] section 2.1, Endpoints [WSASB] [SOAP1.2-1/2003] [MC-NMF]

The table above excludes the IMDA, WSDS, and WSPELD protocol extensions, which inherit their transport from the protocol (WS-Transfer and/or WS-Enumeration) that they extend.

For information on the authentication mechanisms supported on the **Web Service endpoints**, see [MS-ADDM] section 2.1, Endpoints.

6.4 Timers

The Active Directory System requires the following timers, beyond those described in the technical documents of the protocols that comprise the system, as listed in section 2.2.

- **requestTimer**, as specified in section 6.2.3. The timer is initialized, as specified in section 6.2.6.2.5, and the timer expiration processing logic is specified in section 6.2.6.4.1.
- **pingKeepAliveTimer**, as specified in section 6.2.3.

6.5 Non-Timer Events

There is one non-timer event, Host Name Change, in the Active Directory System (beyond those non-timer events specified in the underlying protocol documents).

6.5.1 Host Name Change

This event occurs when the fully-qualified DNS host name or the NetBIOS host name of the server hosting the directory service is changed.

When this event occurs, if the Active Directory System is running as AD DS, then the system SHOULD update its DNS and NetBIOS records, described in [MS-ADTS] sections 6.3.2, DNS Record Registrations, and 6.3.4, NetBIOS Broadcast and NBNS Background, respectively, to reflect the new name of the hosting server.

When this event occurs, whether the system is running as AD DS or AD LDS, the system SHOULD update the Service Principal Name (SPNs) listed in [MS-DRSR] section 2.2, Protocol Security, which are stored on the servicePrincipalName attribute of the computer object of the server hosting the directory service, to reflect the new name of the hosting server. Note that in the case of the Active Directory System running as AD LDS on a server that is not joined to a domain, there is no such computer object and so no SPNs need to be updated.

6.6 Initialization and Reinitialization Procedures

Upon startup, the Active Directory System SHOULD initialize all of the protocols listed in section 5.4.3 as described in the protocol documents for each listed protocol and SHOULD begin servicing incoming requests on those protocols' interfaces. There is no requirement that the protocols be initialized in a particular sequence.

When hosting an AD DS directory service, the directory server SHOULD register (if not already registered) DNS and NetBIOS records as described in [MS-ADTS] sections 6.3.2, DNS Record Registrations, and 6.3.4, NetBIOS Broadcast and NBNS Background, respectively, to enable clients to locate the directory server. If an AD LDS directory service is hosted on a directory server that is joined to an AD DS domain, the directory server SHOULD publish itself by creating an object in AD DS, as described in [MS-ADTS] section 6.3, Publishing and Locating a Domain Controller.

When operating as AD DS, once the server has initialized the protocols listed in section 5.4.3 and is prepared to process incoming requests for those protocols, the directory server SHOULD begin responding to LDAP and **mailslot** ping requests in the manner described in [MS-ADTS] sections 6.3.3, LDAP Ping, and 6.3.5, Mailslot Ping, respectively.

The Active Directory System does not contain a requirement to reinitialize the protocols, nor does it contain specific procedures for reinitializing the protocols. Implementations that desire to perform such a reinitialization SHOULD do so by performing a regular initialization of the protocols.

6.7 Status and Error Returns

The Active Directory System does not define any error handling requirements beyond those described in the Technical Documents of the protocols supported by the system, as listed in section 2.2, and in the failure scenarios specified in section 5.5.

Various kinds of errors may occur impacting the system. More precisely, an error condition may impact one or more protocols supported by the system. Such error conditions and the resulting protocol semantics are described in the corresponding protocol Technical Documents. The system does not constrain the types of errors that can be received through the member protocols.

6.8 System Management Details

The Active Directory System does not have a dedicated configuration or administration protocol. It is configured using the same protocols as otherwise used by applications for interacting with the system, namely, those of sections 5.3.2.1 through 5.3.2.4. As documented in [MS-ADTS], most of the configuration data and parameters are stored in the directory tree where it can be retrieved and modified using the LDAP protocol, including:

- Schema: [MS-ADTS] section 3.1.1.2.1, Schema NC
- LDAP rootDse modify operations (used to trigger administrative operations such as relative identifier (RID) pool invalidation or **garbage collection**): [MS-ADTS] section 3.1.1.3.3, rootDSE Modify Operations
- LDAP Policies: [MS-ADTS] section 3.1.1.3.4.6, LDAP Policies
- LDAP Configurable Settings: [MS-ADTS] section 3.1.1.3.4.7, LDAP Configurable Settings
- LDAP IP-Deny List: [MS-ADTS] section 3.1.1.3.4.8, LDAP IP-Deny List
- Special Objects: [MS-ADTS] section 6.1.1, Special Objects
- Special Attributes: [MS-ADTS] section 6.1.4, Special Attributes

Policy data can be manipulated using the LSAD protocol ([MS-LSAD] section 3.1.4.4, Policy Object Methods).

Some versions of the Active Directory System MAY<11> make use of the Web Service protocols listed in section 5.3.2.4 to support administrative tools used to manage the system.

7 Security

This section documents system-wide security issues that are not otherwise described in the Technical Documents (TDs) for the Member Protocols. It does not duplicate what is already in the Member Protocol TDs unless there is some unique aspect that applies to the system as a whole.

Security is vital to the Active Directory System. As a central repository of account information for the domain, a compromise of the system could in turn permit the compromise of other systems that are dependent on the Active Directory System for authentication. If a hostile party can create an account in the directory, or gain access to an existing account, they might be granted access to systems to which they should have no access. If they can modify configuration information stored in the directory by other systems, such as the Group Policy system [MS-GPSO] or the Message Queuing system [MS-MQSO], they can exert control over the behavior of those systems. The directory can also contain sensitive information to which read access should be restricted. It is crucial that an implementation of the Active Directory System be resilient to attack. Such an implementation SHOULD be designed and written with the expectation that it will be subject to hostile network traffic from an adversary intent on compromising it.

To mitigate these threats, the system contains a set of security mechanisms to restrict access to information stored in the directory. The system's protocols use these mechanisms to restrict access to only those users who are authorized to have it. Such access restrictions can be specified not only at the level of individual directory objects but at the level of individual attributes on a directory object. The system also provides means for securing the network traffic in the system against eavesdropping and tampering. The following section describes in more detail the security mechanisms in the Active Directory System.

7.1 Security Elements

Directory objects are protected by **security descriptors** that contain **access control lists** that grant or deny permissions to security principals (either directly or through group membership) to read, update, or otherwise manipulate the object, as described in [MS-ADTS] section 5.1.3, Authorization. However, it is the decision of the individual protocol what access checks to enforce when accessing the directory. That is, while some protocols will enforce the authorization checks as described in that document, other protocols will substitute their own access checks as described in that individual protocol's Technical Document.

In the Active Directory System, the following protocols perform access checks as described in [MS-ADTS] section 5.1.3, Authorization:

- LDAP
- WS-Transfer
- WS-Enumeration
- ADCAP

The following protocols substitute, in full or in part, a different access check methodology, as described in the protocol's Technical Document:

- DS RPC
- SAMR
- LSAD

- LSAT
- DSSP

When performing an access check, the identity of the requestor, represented as a SID, is compared to the permissions required to perform a given operation and the permissions granted to that identity, in accord with the access check rules of the protocol in use. Each protocol specifies a means by which a requestor can prove (authenticate) its identity to the directory service, so that the identity can be used in subsequent access check decisions. Each protocol's means of authentication is specified in the corresponding protocol document, except that for WS-Transfer, WS-Enumeration, and ADCAP, it is instead specified in [MS-ADDM] section 2.1, Endpoints. The WSDL in Appendix B also contains the **WS-Policy** elements that specify the authentication mechanisms supported on the WS-Transfer, WS-Enumeration, and ADCAP **Web Service endpoints** in the Active Directory System.

The protocols provide mechanisms to digitally-sign requests and responses to prevent them from tampering while being transferred over the network, and to encrypt the traffic to prevent eavesdropping. See section 7.2.

7.2 Communications Security

The Active Directory System relies on messages passed across the network between the client and the directory service. The system does not require this network to be fully trusted and allows for the possibility that a hostile party may be able to intercept such messages while they are in flight. Most of the protocols in the Active Directory System are designed to protect against two key attacks from such an attacker:

- Eavesdropping on the messages to learn information to which the attacker is not intended to have access.
- Altering the request or response messages to cause the directory service or client, respectively, to take action based on information supplied by the attacker.

To protect against these attacks, the system uses transport- and message-level security features to protect traffic between the clients and the directory service. Transport-level security protects the entire transport, effectively creating a protected "tunnel" between the client and directory service through which the messages are sent, protecting the confidentially and integrity of the messages sent over the tunnel. Message-level security encrypts and/or digitally signs each individual message to provide confidentially and integrity of the message, respectively.

There is no single transport- or message- level mechanism used throughout all the protocols that comprise the Active Directory System. The following table summarizes the mechanisms used in each protocol and includes a reference to the relevant section of the protocol Technical Documents for more information.

Transport- and Message-Level Security Features

Protocol	Mechanisms	Reference
LDAP	Transport-level Protection is provided by signing and encryption over a **Secure Sockets Layer**/**Transport Layer Security (TLS)** (SSL/TLS)-protected connection.	[MS-ADTS] section 5.1.2.2, Using SSL/TLS
	Message-level Protection is provided by signing and/or encryption using SASL.	[MS-ADTS] section 5.1.2.1, Using SASL

Protocol	Mechanisms	Reference
DS RPC	Message-level Protection is provided by use of the SPNEGO security provider ([MS-RPCE] section 2.2.1.1.7) to protect the messages at the RPC layer.	[MS-DRSR] section 2.2.4.1, Security Provider
DSSP	None	
SAMR	Transport-level When using RPC over the SMB transport, protection is provided by the SMB transport.	[MS-SAMR] section 2.1, Transport
	Message-level When using RPC over the TCP transport, protection is provided by use of the SPNEGO security provider ([MS-RPCE] section 2.2.1.1.7) to protect the messages at the RPC layer.	[MS-SAMR] section 2.1, Transport
LSAD	Transport-level Protection is provided by the SMB transport over which the RPC requests are sent.	[MS-LSAD] section 2.1, Transport
LSAT	Transport-level When using RPC over the SMB transport, protection is provided by the SMB transport.	[MS-LSAT] sections 2.1, Transport and 3.1.4, Message Processing Events and Sequencing Rules
	Message-level When using RPC over the TCP transport, protection is provided by use of the Netlogon security provider ([MS-RPCE] section 2.2.1.1.7) to protect the messages at the RPC layer.	[MS-LSAT] sections 2.1, Transport and 3.1.4, Message Processing Events and Sequencing Rules
WS-Transfer	Transport-level Protection is provided by the use of TLS to protect the TCP transport. When using the Windows Integrated authentication endpoints, the SPNEGO security provider is used to negotiate the session key used by TLS. When using the username/password authentication endpoints, TLS is used to negotiate a session key using the server's certificate.	[MS-ADDM] section 2.1, Endpoints
WS-Enumeration	Same as WS-Transfer, above.	[MS-ADDM] section 2.1, Endpoints
ADCAP	Same as WS-Transfer, above.	[MS-ADDM] section 2.1, Endpoints

In addition to these mechanisms for protecting desirable traffic between the client and the server, many of the protocols in the Active Directory System also have mechanisms for rejecting undesirable traffic, that is, traffic that has been judged as potentially harmful to the directory service. The following table lists the protocols that have such mechanisms, a summary of the mechanisms, and a reference to further information. Note that these mechanisms are in addition to any access checks (section 7.1) that are performed by the protocol.

Additional Security Mechanisms

Protocol	Mechanisms	Reference
LDAP	LDAP Policies: establish limits on the size of the operations that a client can request.	[MS-ADTS] section 3.1.1.3.4.6, LDAP Policies
	LDAP IP Deny List: provides a configurable list of IPv4 addresses from which the directory service will ignore requests.	[MS-ADTS] section 3.1.1.3.4.8, LDAP IP-Deny List
DS RPC	Uses **Interface Definition Language (IDL)** "[range]" attributes to limit the size of requests that will be accepted by the directory service.	[MS-DRSR] section 7, Appendix A: Full IDL
DSSP	None	
SAMR	Uses IDL "[range]" attributes to limit the size of requests that will be accepted by the directory service.	[MS-SAMR] section 6, Appendix A: Full IDL
	Configures the RPC runtime to perform a strict **Network Data Representation (NDR)** data consistency check at target level 5.0.	[MS-SAMR] section 2.1, Transport
LSAD	In the Microsoft implementation, uses IDL "[range]" attributes to limit the size of requests that will be accepted by the directory service.	[MS-LSAD] sections 6, Appendix A: Full IDL, and 7, Appendix B: Product Behavior
	In the Microsoft implementation, configures the RPC runtime to perform a strict NDR data consistency check at target level 5.0.	[MS-LSAD] section 7, Appendix B: Product Behavior
	Configures the RPC runtime to enforce Maximum Server Input Data Size.	[MS-LSAD] section 2.2.1, Constant Value Definitions
LSAT	In the Microsoft implementation, uses IDL "[range]" attributes to limit the size of requests that will be accepted by the directory service.	[MS-LSAT] sections 6Appendix A: Full IDL, and 7, Appendix B: Product Behavior
	In the Microsoft implementation, configures the RPC runtime to perform a strict NDR data consistency check at target level 5.0.	[MS-LSAT] section 7, Appendix B: Product Behavior
WS-Transfer	Implementations MAY <12> provide mechanisms to limit the operations that can be performed or the size of the response.	
WS-Enumeration	Implementations MAY <13> provide mechanisms to limit the operations that can be performed or the size of the response.	
ADCAP	Implementations MAY <14> provide mechanisms to limit the operations that can be performed or the size of the response.	

7.3 System Configuration Security

The configuration data and parameters for the Active Directory System are stored in the directory service itself. It is retrieved and manipulated using the same protocols used to manipulate any other data stored in the directory. In particular, much of it is stored in the form of directory objects

accessible via the LDAP protocol. As such, this configuration data is protected by the access checks enforced by these protocols. Therefore, the security descriptors of the directory object on which the settings are stored are vital to protecting the system configuration.

This also means that the security of these configuration settings is dependent on the system's ability to secure the messages as they travel over the network, as described in section 7.2. At a minimum, clients SHOULD use one of the mechanisms documented there to ensure message integrity. Failure to do so could permit an attacker to perform an elevation of privilege attack by intercepting and modifying a request message sent by the client to perform an action (using the client's privileges) of the attacker's choosing.

7.4 Internal Security

Internal security is the means by which the Active Directory System ensures its own security, including the steps that other entities that interact with the system should take in order to protect the security of the system.

To protect its own security, the Active Directory System uses the mechanisms described in sections 7.1, 7.2, and 7.3 to enforce access controls, protect communications, and protect its configuration. The system SHOULD time-out operations that are consuming an excessive amount of directory service resources or that are otherwise interfering with the directory service's ability to respond to requests from other clients.

Other systems interacting with the Active Directory System SHOULD take the following steps to protect the security of this system:

- Use the SPNs specified in [MS-DRSR] sections 2.2.4.3, SPN for a Target DC in AD DS, and 2.2.4.3, SPN for a Target DC in AD LDS, to perform **mutual authentication** against the directory service.
- Use the mechanisms available in the protocols to provide integrity and confidentiality of the messages.
- If performing a request against the directory service on behalf of a less-trusted component, any input from the less-trusted component must be validated to protect against a luring attack where the less-trusted component tries to get the more-trusted component to perform an operation of the less-trusted component's choice against the directory service.
- Avoid performing queries against the directory service that will take an excessive amount of time to satisfy (for example, queries that will require the directory service to walk through tens of thousands of entries to find a matching entry).
- Avoid opening an excessive number of simultaneous connections to the directory service. Each connection consumes resources on the directory service. A single client opening a large number of connections can reduce the number of clients that the directory service can simultaneously service.

7.5 External Security

External security is the means by which the Active Directory System ensures the security of other systems with which it interacts, and the steps that such other systems can take to protect their own security when interacting with the Active Directory System.

The Active Directory System takes the following steps to protect other systems that interact with it:

- Times-out long running operations to prevent clients from hanging indefinitely while waiting for a response from the directory service.
- Honors operation time limits specified by the client in requests made by that client.
- Exposes SPNs that can be used by clients to perform mutual authentication against the directory service.

Other systems that interact with the Active Directory System SHOULD take the following steps to protect their own security:

- Specify a time limit in their requests to the directory service that is not longer than the maximum amount of time the client can afford to wait for a response.
- Enforce client-side timeouts so that even if the directory service does not respond in a timely fashion and/or ignores the time limit specified by the client, the client will abandon the operation and not hang indefinitely waiting for the directory service to respond.
- Avoid performing inefficient operations against the directory service that will take longer to complete than the client can afford to wait for.
- Use the SPNs to perform mutual authentication to ensure that it is communicating with the intended directory service and not an impostor.

8 Appendix A: Binding and Service WSDL

This appendix contains the WSDL bindings and service elements (as defined in [WSDL]) for the WS-Transfer (including the IMDA extensions), WS-Enumeration, and ADCAP protocol interfaces exposed by the Active Directory System. The purpose of this appendix is to outline WSDL bindings and service elements. It does not provide a comprehensive WSDL definition containing all types, messages, and other schema elements that are covered by other Web Service protocol documents. Consequently, this WSDL has dependencies that render it unsuitable for client code-generation in its present form. A compatible implementation of the Web Service protocols MUST expose the **Web Service endpoints** using the bindings and services specified in this appendix. The remaining portions of the WSDL (for example, types, messages, and port types) are specified in the protocol documents [MS-WSTIM] (whose WSDL supersedes the WSDL in [WXFR] for the system), [WSENUM], and [MS-ADCAP]. Additionally, [MS-WSDS] and [MS-WSPELD] contain extensions to the WSDL corresponding to their protocol extensions to WS-Transfer and WS-Enumeration.

Not shown in the WSDL below are the **WS-AddressingAndIdentity** claims (embedded in the WS-Addressing **endpoint reference** for each WSDL port) which MAY<15> be exposed by the server in the WSDL. Also, in the WSDL shown below, the element "Binary Encoding Policy Assertion" (<msb:BinaryEncoding>) is defined in [MS-WSPOL] sections 2.2.3.6, Binary Encoding Policy Assertion, and 6.6, Binary Encoding Policy Assertion, and the element "Message Framing Security Provider Negotiation Policy Assertion" (<msf:WindowsTransportSecurity>) is defined in [MS-WSPOL] sections 2.2.3.8, Message Framing Security Provider Negotiation Policy Assertion, and 6.8, Message Framing Security Provider Negotiation Policy Assertion. The namespace http://schemas.microsoft.com/ws/2005/12/wsdl/contract is defined in [MS-WSPOL].

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:wsp="http://www.w3.org/ns/ws-policy"
                  xmlns:msc="http://schemas.microsoft.com/ws/2005/12/wsdl/contract"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
                  xmlns:wsap="http://schemas.xmlsoap.org/ws/2004/08/addressing/policy"
                  xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
                  xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
                  xmlns:tns="http://schemas.microsoft.com/2008/1/ActiveDirectory"
                  xmlns:wsa10="http://www.w3.org/2005/08/addressing"
                  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
                  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd"
                  xmlns:wxf="http://schemas.xmlsoap.org/ws/2004/09/transfer"
                  xmlns:wsen="http://schemas.xmlsoap.org/ws/2004/09/enumeration"

xmlns:ca="http://schemas.microsoft.com/2008/1/ActiveDirectory/CustomActions"
                  xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
                  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
                  name="ActiveDirectoryWebService"
                  targetNamespace="http://schemas.microsoft.com/2008/1/ActiveDirectory"
                  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsp:Policy wsu:Id="NetTcpBinding_Resource_policy">
    <wsdl:documentation>
      Security policy for WS-Transfer Resource port type using
      Windows Integrated authentication.
    </wsdl:documentation>
    <wsp:ExactlyOne>
      <wsp:All>
        <msb:BinaryEncoding
xmlns:msb="http://schemas.microsoft.com/ws/06/2004/mspolicy/netbinary1"></msb:BinaryEncoding>
```

```
        <sp:TransportBinding xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
          <wsp:Policy>
            <sp:TransportToken>
              <wsp:Policy>
                <msf:WindowsTransportSecurity
  xmlns:msf="http://schemas.microsoft.com/ws/2006/05/framing/policy">
                  <msf:ProtectionLevel>EncryptAndSign</msf:ProtectionLevel>
                </msf:WindowsTransportSecurity>
              </wsp:Policy>
            </sp:TransportToken>
            <sp:AlgorithmSuite>
              <wsp:Policy>
                <sp:Basic256></sp:Basic256>
              </wsp:Policy>
            </sp:AlgorithmSuite>
            <sp:Layout>
              <wsp:Policy>
                <sp:Strict></sp:Strict>
              </wsp:Policy>
            </sp:Layout>
          </wsp:Policy>
        </sp:TransportBinding>
        <wsam:Addressing>
          <wsp:Policy>
            <wsam:AnonymousResponses></wsam:AnonymousResponses>
          </wsp:Policy>
        </wsam:Addressing>
      </wsp:All>
    </wsp:ExactlyOne>
  </wsp:Policy>
  <wsp:Policy wsu:Id="NetTcpBinding_ResourceFactory_policy">
    <wsdl:documentation>
      Security policy for WS-Transfer ResourceFactory port type using
      Windows Integrated authentication.
    </wsdl:documentation>
    <wsp:ExactlyOne>
      <wsp:All>
        <msb:BinaryEncoding
  xmlns:msb="http://schemas.microsoft.com/ws/06/2004/mspolicy/netbinary1"></msb:BinaryEncoding>
        <sp:TransportBinding xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
          <wsp:Policy>
            <sp:TransportToken>
              <wsp:Policy>
                <msf:WindowsTransportSecurity
  xmlns:msf="http://schemas.microsoft.com/ws/2006/05/framing/policy">
                  <msf:ProtectionLevel>EncryptAndSign</msf:ProtectionLevel>
                </msf:WindowsTransportSecurity>
              </wsp:Policy>
            </sp:TransportToken>
            <sp:AlgorithmSuite>
              <wsp:Policy>
                <sp:Basic256></sp:Basic256>
              </wsp:Policy>
            </sp:AlgorithmSuite>
            <sp:Layout>
              <wsp:Policy>
                <sp:Strict></sp:Strict>
              </wsp:Policy>
            </sp:Layout>
```

```
          </wsp:Policy>
        </sp:TransportBinding>
        <wsam:Addressing>
          <wsp:Policy>
            <wsam:AnonymousResponses></wsam:AnonymousResponses>
          </wsp:Policy>
        </wsam:Addressing>
      </wsp:All>
    </wsp:ExactlyOne>
  </wsp:Policy>
  <wsp:Policy wsu:Id="NetTcpBinding_Search_policy">
    <wsdl:documentation>
      Security policy for WS-Enumeration Search port type using
      Windows Integrated authentication.
    </wsdl:documentation>
    <wsp:ExactlyOne>
      <wsp:All>
        <msb:BinaryEncoding
xmlns:msb="http://schemas.microsoft.com/ws/06/2004/mspolicy/netbinary1"></msb:BinaryEncoding>
        <sp:TransportBinding xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
          <wsp:Policy>
            <sp:TransportToken>
              <wsp:Policy>
                <msf:WindowsTransportSecurity
xmlns:msf="http://schemas.microsoft.com/ws/2006/05/framing/policy">
                  <msf:ProtectionLevel>EncryptAndSign</msf:ProtectionLevel>
                </msf:WindowsTransportSecurity>
              </wsp:Policy>
            </sp:TransportToken>
            <sp:AlgorithmSuite>
              <wsp:Policy>
                <sp:Basic256></sp:Basic256>
              </wsp:Policy>
            </sp:AlgorithmSuite>
            <sp:Layout>
              <wsp:Policy>
                <sp:Strict></sp:Strict>
              </wsp:Policy>
            </sp:Layout>
          </wsp:Policy>
        </sp:TransportBinding>
        <wsam:Addressing>
          <wsp:Policy>
            <wsam:AnonymousResponses></wsam:AnonymousResponses>
          </wsp:Policy>
        </wsam:Addressing>
      </wsp:All>
    </wsp:ExactlyOne>
  </wsp:Policy>
  <wsp:Policy wsu:Id="NetTcpBinding_AccountManagement_policy">
    <wsdl:documentation>
      Security policy for ADCAP AccountManagement port type using
      Windows Integrated authentication.
    </wsdl:documentation>
    <wsp:ExactlyOne>
      <wsp:All>
        <msb:BinaryEncoding
xmlns:msb="http://schemas.microsoft.com/ws/06/2004/mspolicy/netbinary1"></msb:BinaryEncoding>
        <sp:TransportBinding xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
```

```
          <wsp:Policy>
            <sp:TransportToken>
              <wsp:Policy>
                <msf:WindowsTransportSecurity
xmlns:msf="http://schemas.microsoft.com/ws/2006/05/framing/policy">
                  <msf:ProtectionLevel>EncryptAndSign</msf:ProtectionLevel>
                </msf:WindowsTransportSecurity>
              </wsp:Policy>
            </sp:TransportToken>
            <sp:AlgorithmSuite>
              <wsp:Policy>
                <sp:Basic256></sp:Basic256>
              </wsp:Policy>
            </sp:AlgorithmSuite>
            <sp:Layout>
              <wsp:Policy>
                <sp:Strict></sp:Strict>
              </wsp:Policy>
            </sp:Layout>
          </wsp:Policy>
        </sp:TransportBinding>
        <wsam:Addressing>
          <wsp:Policy>
            <wsam:AnonymousResponses></wsam:AnonymousResponses>
          </wsp:Policy>
        </wsam:Addressing>
      </wsp:All>
    </wsp:ExactlyOne>
  </wsp:Policy>
  <wsp:Policy wsu:Id="NetTcpBinding_TopologyManagement_policy">
    <wsdl:documentation>
      Security policy for ADCAP TopologyManagement port type using
      Windows Integrated authentication.
    </wsdl:documentation>
    <wsp:ExactlyOne>
      <wsp:All>
        <msb:BinaryEncoding
xmlns:msb="http://schemas.microsoft.com/ws/06/2004/mspolicy/netbinary1"></msb:BinaryEncoding>
        <sp:TransportBinding xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
          <wsp:Policy>
            <sp:TransportToken>
              <wsp:Policy>
                <msf:WindowsTransportSecurity
xmlns:msf="http://schemas.microsoft.com/ws/2006/05/framing/policy">
                  <msf:ProtectionLevel>EncryptAndSign</msf:ProtectionLevel>
                </msf:WindowsTransportSecurity>
              </wsp:Policy>
            </sp:TransportToken>
            <sp:AlgorithmSuite>
              <wsp:Policy>
                <sp:Basic256></sp:Basic256>
              </wsp:Policy>
            </sp:AlgorithmSuite>
            <sp:Layout>
              <wsp:Policy>
                <sp:Strict></sp:Strict>
              </wsp:Policy>
            </sp:Layout>
          </wsp:Policy>
```

```
        </sp:TransportBinding>
        <wsam:Addressing>
          <wsp:Policy>
            <wsam:AnonymousResponses></wsam:AnonymousResponses>
          </wsp:Policy>
        </wsam:Addressing>
      </wsp:All>
    </wsp:ExactlyOne>
  </wsp:Policy>
  <wsp:Policy wsu:Id="NetTcpBinding_Resource1_policy">
    <wsdl:documentation>
      Security policy for WS-Transfer Resource port type using
      username/password authentication.
    </wsdl:documentation>
    <wsp:ExactlyOne>
      <wsp:All>
        <msb:BinaryEncoding
xmlns:msb="http://schemas.microsoft.com/ws/06/2004/mspolicy/netbinary1"></msb:BinaryEncoding>
        <sp:TransportBinding xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
          <wsp:Policy>
            <sp:TransportToken>
              <wsp:Policy>
                <msf:SslTransportSecurity
xmlns:msf="http://schemas.microsoft.com/ws/2006/05/framing/policy"></msf:SslTransportSecurity
>
              </wsp:Policy>
            </sp:TransportToken>
            <sp:AlgorithmSuite>
              <wsp:Policy>
                <sp:Basic256></sp:Basic256>
              </wsp:Policy>
            </sp:AlgorithmSuite>
            <sp:Layout>
              <wsp:Policy>
                <sp:Strict></sp:Strict>
              </wsp:Policy>
            </sp:Layout>
            <sp:IncludeTimestamp></sp:IncludeTimestamp>
          </wsp:Policy>
        </sp:TransportBinding>
        <sp:EndorsingSupportingTokens
xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
          <wsp:Policy>
            <sp:SecureConversationToken
sp:IncludeToken="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy/IncludeToken/AlwaysToRe
cipient">
              <wsp:Policy>
                <sp:BootstrapPolicy>
                  <wsp:Policy>
                    <sp:SignedParts>
                      <sp:Body></sp:Body>
                      <sp:Header Name="To"
Namespace="http://www.w3.org/2005/08/addressing"></sp:Header>
                      <sp:Header Name="From"
Namespace="http://www.w3.org/2005/08/addressing"></sp:Header>
                      <sp:Header Name="FaultTo"
Namespace="http://www.w3.org/2005/08/addressing"></sp:Header>
                      <sp:Header Name="ReplyTo"
Namespace="http://www.w3.org/2005/08/addressing"></sp:Header>
```

```
                    <sp:Header Name="MessageID"
Namespace="http://www.w3.org/2005/08/addressing"></sp:Header>
                    <sp:Header Name="RelatesTo"
Namespace="http://www.w3.org/2005/08/addressing"></sp:Header>
                    <sp:Header Name="Action"
Namespace="http://www.w3.org/2005/08/addressing"></sp:Header>
                  </sp:SignedParts>
                  <sp:EncryptedParts>
                    <sp:Body></sp:Body>
                  </sp:EncryptedParts>
                  <sp:TransportBinding>
                    <wsp:Policy>
                      <sp:TransportToken>
                        <wsp:Policy>
                          <msf:SslTransportSecurity
xmlns:msf="http://schemas.microsoft.com/ws/2006/05/framing/policy"></msf:SslTransportSecurity
>
                        </wsp:Policy>
                      </sp:TransportToken>
                      <sp:AlgorithmSuite>
                        <wsp:Policy>
                          <sp:Basic256></sp:Basic256>
                        </wsp:Policy>
                      </sp:AlgorithmSuite>
                      <sp:Layout>
                        <wsp:Policy>
                          <sp:Strict></sp:Strict>
                        </wsp:Policy>
                      </sp:Layout>
                      <sp:IncludeTimestamp></sp:IncludeTimestamp>
                    </wsp:Policy>
                  </sp:TransportBinding>
                  <sp:SignedSupportingTokens>
                    <wsp:Policy>
                      <sp:UsernameToken
sp:IncludeToken="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy/IncludeToken/AlwaysToRe
cipient">
                        <wsp:Policy>
                          <sp:WssUsernameToken10></sp:WssUsernameToken10>
                        </wsp:Policy>
                      </sp:UsernameToken>
                    </wsp:Policy>
                  </sp:SignedSupportingTokens>
                  <sp:Wss11>
                    <wsp:Policy>
                      <sp:MustSupportRefKeyIdentifier></sp:MustSupportRefKeyIdentifier>
                      <sp:MustSupportRefIssuerSerial></sp:MustSupportRefIssuerSerial>
                      <sp:MustSupportRefThumbprint></sp:MustSupportRefThumbprint>
                      <sp:MustSupportRefEncryptedKey></sp:MustSupportRefEncryptedKey>
                    </wsp:Policy>
                  </sp:Wss11>
                  <sp:Trust10>
                    <wsp:Policy>
                      <sp:MustSupportIssuedTokens></sp:MustSupportIssuedTokens>
                      <sp:RequireClientEntropy></sp:RequireClientEntropy>
                      <sp:RequireServerEntropy></sp:RequireServerEntropy>
                    </wsp:Policy>
                  </sp:Trust10>
                </wsp:Policy>
```

```
                </sp:BootstrapPolicy>
              </wsp:Policy>
            </sp:SecureConversationToken>
            <sp:SignedParts>
              <sp:Header Name="To"
Namespace="http://www.w3.org/2005/08/addressing"></sp:Header>
            </sp:SignedParts>
          </wsp:Policy>
        </sp:EndorsingSupportingTokens>
        <sp:Wss11 xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
          <wsp:Policy>
            <sp:MustSupportRefKeyIdentifier></sp:MustSupportRefKeyIdentifier>
            <sp:MustSupportRefIssuerSerial></sp:MustSupportRefIssuerSerial>
            <sp:MustSupportRefThumbprint></sp:MustSupportRefThumbprint>
            <sp:MustSupportRefEncryptedKey></sp:MustSupportRefEncryptedKey>
          </wsp:Policy>
        </sp:Wss11>
        <sp:Trust10 xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
          <wsp:Policy>
            <sp:MustSupportIssuedTokens></sp:MustSupportIssuedTokens>
            <sp:RequireClientEntropy></sp:RequireClientEntropy>
            <sp:RequireServerEntropy></sp:RequireServerEntropy>
          </wsp:Policy>
        </sp:Trust10>
        <wsam:Addressing>
          <wsp:Policy>
            <wsam:AnonymousResponses></wsam:AnonymousResponses>
          </wsp:Policy>
        </wsam:Addressing>
      </wsp:All>
    </wsp:ExactlyOne>
  </wsp:Policy>
  <wsp:Policy wsu:Id="NetTcpBinding_ResourceFactory1_policy">
    <wsdl:documentation>
      Security policy for WS-Transfer ResourceFactory port type using
      username/password authentication.
    </wsdl:documentation>
    <wsp:ExactlyOne>
      <wsp:All>
        <msb:BinaryEncoding
xmlns:msb="http://schemas.microsoft.com/ws/06/2004/mspolicy/netbinary1"></msb:BinaryEncoding>
        <sp:TransportBinding xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
          <wsp:Policy>
            <sp:TransportToken>
              <wsp:Policy>
                <msf:SslTransportSecurity
xmlns:msf="http://schemas.microsoft.com/ws/2006/05/framing/policy"></msf:SslTransportSecurity
>
              </wsp:Policy>
            </sp:TransportToken>
            <sp:AlgorithmSuite>
              <wsp:Policy>
                <sp:Basic256></sp:Basic256>
              </wsp:Policy>
            </sp:AlgorithmSuite>
            <sp:Layout>
              <wsp:Policy>
                <sp:Strict></sp:Strict>
              </wsp:Policy>
```

```
            </sp:Layout>
            <sp:IncludeTimestamp></sp:IncludeTimestamp>
          </wsp:Policy>
        </sp:TransportBinding>
        <sp:EndorsingSupportingTokens
xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
          <wsp:Policy>
            <sp:SecureConversationToken
sp:IncludeToken="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy/IncludeToken/AlwaysToRe
cipient">
              <wsp:Policy>
                <sp:BootstrapPolicy>
                  <wsp:Policy>
                    <sp:SignedParts>
                      <sp:Body></sp:Body>
                      <sp:Header Name="To"
Namespace="http://www.w3.org/2005/08/addressing"></sp:Header>
                      <sp:Header Name="From"
Namespace="http://www.w3.org/2005/08/addressing"></sp:Header>
                      <sp:Header Name="FaultTo"
Namespace="http://www.w3.org/2005/08/addressing"></sp:Header>
                      <sp:Header Name="ReplyTo"
Namespace="http://www.w3.org/2005/08/addressing"></sp:Header>
                      <sp:Header Name="MessageID"
Namespace="http://www.w3.org/2005/08/addressing"></sp:Header>
                      <sp:Header Name="RelatesTo"
Namespace="http://www.w3.org/2005/08/addressing"></sp:Header>
                      <sp:Header Name="Action"
Namespace="http://www.w3.org/2005/08/addressing"></sp:Header>
                    </sp:SignedParts>
                    <sp:EncryptedParts>
                      <sp:Body></sp:Body>
                    </sp:EncryptedParts>
                    <sp:TransportBinding>
                      <wsp:Policy>
                        <sp:TransportToken>
                          <wsp:Policy>
                            <msf:SslTransportSecurity
xmlns:msf="http://schemas.microsoft.com/ws/2006/05/framing/policy"></msf:SslTransportSecurity
>
                          </wsp:Policy>
                        </sp:TransportToken>
                        <sp:AlgorithmSuite>
                          <wsp:Policy>
                            <sp:Basic256></sp:Basic256>
                          </wsp:Policy>
                        </sp:AlgorithmSuite>
                        <sp:Layout>
                          <wsp:Policy>
                            <sp:Strict></sp:Strict>
                          </wsp:Policy>
                        </sp:Layout>
                        <sp:IncludeTimestamp></sp:IncludeTimestamp>
                      </wsp:Policy>
                    </sp:TransportBinding>
                    <sp:SignedSupportingTokens>
                      <wsp:Policy>
                        <sp:UsernameToken
sp:IncludeToken="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy/IncludeToken/AlwaysToRe
cipient">
```

```
                        <wsp:Policy>
                          <sp:WssUsernameToken10></sp:WssUsernameToken10>
                        </wsp:Policy>
                      </sp:UsernameToken>
                    </wsp:Policy>
                  </sp:SignedSupportingTokens>
                  <sp:Wss11>
                    <wsp:Policy>
                      <sp:MustSupportRefKeyIdentifier></sp:MustSupportRefKeyIdentifier>
                      <sp:MustSupportRefIssuerSerial></sp:MustSupportRefIssuerSerial>
                      <sp:MustSupportRefThumbprint></sp:MustSupportRefThumbprint>
                      <sp:MustSupportRefEncryptedKey></sp:MustSupportRefEncryptedKey>
                    </wsp:Policy>
                  </sp:Wss11>
                  <sp:Trust10>
                    <wsp:Policy>
                      <sp:MustSupportIssuedTokens></sp:MustSupportIssuedTokens>
                      <sp:RequireClientEntropy></sp:RequireClientEntropy>
                      <sp:RequireServerEntropy></sp:RequireServerEntropy>
                    </wsp:Policy>
                  </sp:Trust10>
                </wsp:Policy>
              </sp:BootstrapPolicy>
            </wsp:Policy>
          </sp:SecureConversationToken>
          <sp:SignedParts>
            <sp:Header Name="To"
Namespace="http://www.w3.org/2005/08/addressing"></sp:Header>
          </sp:SignedParts>
        </wsp:Policy>
      </sp:EndorsingSupportingTokens>
      <sp:Wss11 xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
        <wsp:Policy>
          <sp:MustSupportRefKeyIdentifier></sp:MustSupportRefKeyIdentifier>
          <sp:MustSupportRefIssuerSerial></sp:MustSupportRefIssuerSerial>
          <sp:MustSupportRefThumbprint></sp:MustSupportRefThumbprint>
          <sp:MustSupportRefEncryptedKey></sp:MustSupportRefEncryptedKey>
        </wsp:Policy>
      </sp:Wss11>
      <sp:Trust10 xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
        <wsp:Policy>
          <sp:MustSupportIssuedTokens></sp:MustSupportIssuedTokens>
          <sp:RequireClientEntropy></sp:RequireClientEntropy>
          <sp:RequireServerEntropy></sp:RequireServerEntropy>
        </wsp:Policy>
      </sp:Trust10>
      <wsam:Addressing>
        <wsp:Policy>
          <wsam:AnonymousResponses></wsam:AnonymousResponses>
        </wsp:Policy>
      </wsam:Addressing>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
<wsp:Policy wsu:Id="NetTcpBinding_Search1_policy">
  <wsdl:documentation>
    Security policy for WS-Enumeration Search port type using
    username/password authentication.
  </wsdl:documentation>
```

```
    <wsp:ExactlyOne>
      <wsp:All>
        <msb:BinaryEncoding
xmlns:msb="http://schemas.microsoft.com/ws/06/2004/mspolicy/netbinary1"></msb:BinaryEncoding>
        <sp:TransportBinding xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
          <wsp:Policy>
            <sp:TransportToken>
              <wsp:Policy>
                <msf:SslTransportSecurity
xmlns:msf="http://schemas.microsoft.com/ws/2006/05/framing/policy"></msf:SslTransportSecurity
>
              </wsp:Policy>
            </sp:TransportToken>
            <sp:AlgorithmSuite>
              <wsp:Policy>
                <sp:Basic256></sp:Basic256>
              </wsp:Policy>
            </sp:AlgorithmSuite>
            <sp:Layout>
              <wsp:Policy>
                <sp:Strict></sp:Strict>
              </wsp:Policy>
            </sp:Layout>
            <sp:IncludeTimestamp></sp:IncludeTimestamp>
          </wsp:Policy>
        </sp:TransportBinding>
        <sp:EndorsingSupportingTokens
xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
          <wsp:Policy>
            <sp:SecureConversationToken
sp:IncludeToken="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy/IncludeToken/AlwaysToRe
cipient">
              <wsp:Policy>
                <sp:BootstrapPolicy>
                  <wsp:Policy>
                    <sp:SignedParts>
                      <sp:Body></sp:Body>
                      <sp:Header Name="To"
Namespace="http://www.w3.org/2005/08/addressing"></sp:Header>
                      <sp:Header Name="From"
Namespace="http://www.w3.org/2005/08/addressing"></sp:Header>
                      <sp:Header Name="FaultTo"
Namespace="http://www.w3.org/2005/08/addressing"></sp:Header>
                      <sp:Header Name="ReplyTo"
Namespace="http://www.w3.org/2005/08/addressing"></sp:Header>
                      <sp:Header Name="MessageID"
Namespace="http://www.w3.org/2005/08/addressing"></sp:Header>
                      <sp:Header Name="RelatesTo"
Namespace="http://www.w3.org/2005/08/addressing"></sp:Header>
                      <sp:Header Name="Action"
Namespace="http://www.w3.org/2005/08/addressing"></sp:Header>
                    </sp:SignedParts>
                    <sp:EncryptedParts>
                      <sp:Body></sp:Body>
                    </sp:EncryptedParts>
                    <sp:TransportBinding>
                      <wsp:Policy>
                        <sp:TransportToken>
                          <wsp:Policy>
```

```
                          <msf:SslTransportSecurity
xmlns:msf="http://schemas.microsoft.com/ws/2006/05/framing/policy"></msf:SslTransportSecurity
>
                        </wsp:Policy>
                      </sp:TransportToken>
                      <sp:AlgorithmSuite>
                        <wsp:Policy>
                          <sp:Basic256></sp:Basic256>
                        </wsp:Policy>
                      </sp:AlgorithmSuite>
                      <sp:Layout>
                        <wsp:Policy>
                          <sp:Strict></sp:Strict>
                        </wsp:Policy>
                      </sp:Layout>
                      <sp:IncludeTimestamp></sp:IncludeTimestamp>
                    </wsp:Policy>
                  </sp:TransportBinding>
                  <sp:SignedSupportingTokens>
                    <wsp:Policy>
                      <sp:UsernameToken
sp:IncludeToken="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy/IncludeToken/AlwaysToRe
cipient">
                        <wsp:Policy>
                          <sp:WssUsernameToken10></sp:WssUsernameToken10>
                        </wsp:Policy>
                      </sp:UsernameToken>
                    </wsp:Policy>
                  </sp:SignedSupportingTokens>
                  <sp:Wss11>
                    <wsp:Policy>
                      <sp:MustSupportRefKeyIdentifier></sp:MustSupportRefKeyIdentifier>
                      <sp:MustSupportRefIssuerSerial></sp:MustSupportRefIssuerSerial>
                      <sp:MustSupportRefThumbprint></sp:MustSupportRefThumbprint>
                      <sp:MustSupportRefEncryptedKey></sp:MustSupportRefEncryptedKey>
                    </wsp:Policy>
                  </sp:Wss11>
                  <sp:Trust10>
                    <wsp:Policy>
                      <sp:MustSupportIssuedTokens></sp:MustSupportIssuedTokens>
                      <sp:RequireClientEntropy></sp:RequireClientEntropy>
                      <sp:RequireServerEntropy></sp:RequireServerEntropy>
                    </wsp:Policy>
                  </sp:Trust10>
                </wsp:Policy>
              </sp:BootstrapPolicy>
            </wsp:Policy>
          </sp:SecureConversationToken>
          <sp:SignedParts>
            <sp:Header Name="To"
Namespace="http://www.w3.org/2005/08/addressing"></sp:Header>
          </sp:SignedParts>
        </wsp:Policy>
      </sp:EndorsingSupportingTokens>
      <sp:Wss11 xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
        <wsp:Policy>
          <sp:MustSupportRefKeyIdentifier></sp:MustSupportRefKeyIdentifier>
          <sp:MustSupportRefIssuerSerial></sp:MustSupportRefIssuerSerial>
          <sp:MustSupportRefThumbprint></sp:MustSupportRefThumbprint>
```

```
            <sp:MustSupportRefEncryptedKey></sp:MustSupportRefEncryptedKey>
          </wsp:Policy>
        </sp:Wss11>
        <sp:Trust10 xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
          <wsp:Policy>
            <sp:MustSupportIssuedTokens></sp:MustSupportIssuedTokens>
            <sp:RequireClientEntropy></sp:RequireClientEntropy>
            <sp:RequireServerEntropy></sp:RequireServerEntropy>
          </wsp:Policy>
        </sp:Trust10>
        <wsam:Addressing>
          <wsp:Policy>
            <wsam:AnonymousResponses></wsam:AnonymousResponses>
          </wsp:Policy>
        </wsam:Addressing>
      </wsp:All>
    </wsp:ExactlyOne>
  </wsp:Policy>
  <wsp:Policy wsu:Id="NetTcpBinding_AccountManagement1_policy">
    <wsdl:documentation>
      Security policy for ADCAP AccountManagement port type using
      username/password authentication.
    </wsdl:documentation>
    <wsp:ExactlyOne>
      <wsp:All>
        <msb:BinaryEncoding
xmlns:msb="http://schemas.microsoft.com/ws/06/2004/mspolicy/netbinary1"></msb:BinaryEncoding>
        <sp:TransportBinding xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
          <wsp:Policy>
            <sp:TransportToken>
              <wsp:Policy>
                <msf:SslTransportSecurity
xmlns:msf="http://schemas.microsoft.com/ws/2006/05/framing/policy"></msf:SslTransportSecurity
>
              </wsp:Policy>
            </sp:TransportToken>
            <sp:AlgorithmSuite>
              <wsp:Policy>
                <sp:Basic256></sp:Basic256>
              </wsp:Policy>
            </sp:AlgorithmSuite>
            <sp:Layout>
              <wsp:Policy>
                <sp:Strict></sp:Strict>
              </wsp:Policy>
            </sp:Layout>
            <sp:IncludeTimestamp></sp:IncludeTimestamp>
          </wsp:Policy>
        </sp:TransportBinding>
        <sp:EndorsingSupportingTokens
xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
          <wsp:Policy>
            <sp:SecureConversationToken
sp:IncludeToken="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy/IncludeToken/AlwaysToRe
cipient">
              <wsp:Policy>
                <sp:BootstrapPolicy>
                  <wsp:Policy>
                    <sp:SignedParts>
```

```
                    <sp:Body></sp:Body>
                    <sp:Header Name="To"
Namespace="http://www.w3.org/2005/08/addressing"></sp:Header>
                    <sp:Header Name="From"
Namespace="http://www.w3.org/2005/08/addressing"></sp:Header>
                    <sp:Header Name="FaultTo"
Namespace="http://www.w3.org/2005/08/addressing"></sp:Header>
                    <sp:Header Name="ReplyTo"
Namespace="http://www.w3.org/2005/08/addressing"></sp:Header>
                    <sp:Header Name="MessageID"
Namespace="http://www.w3.org/2005/08/addressing"></sp:Header>
                    <sp:Header Name="RelatesTo"
Namespace="http://www.w3.org/2005/08/addressing"></sp:Header>
                    <sp:Header Name="Action"
Namespace="http://www.w3.org/2005/08/addressing"></sp:Header>
                  </sp:SignedParts>
                  <sp:EncryptedParts>
                    <sp:Body></sp:Body>
                  </sp:EncryptedParts>
                  <sp:TransportBinding>
                    <wsp:Policy>
                      <sp:TransportToken>
                        <wsp:Policy>
                          <msf:SslTransportSecurity
xmlns:msf="http://schemas.microsoft.com/ws/2006/05/framing/policy"></msf:SslTransportSecurity
>
                        </wsp:Policy>
                      </sp:TransportToken>
                      <sp:AlgorithmSuite>
                        <wsp:Policy>
                          <sp:Basic256></sp:Basic256>
                        </wsp:Policy>
                      </sp:AlgorithmSuite>
                      <sp:Layout>
                        <wsp:Policy>
                          <sp:Strict></sp:Strict>
                        </wsp:Policy>
                      </sp:Layout>
                      <sp:IncludeTimestamp></sp:IncludeTimestamp>
                    </wsp:Policy>
                  </sp:TransportBinding>
                  <sp:SignedSupportingTokens>
                    <wsp:Policy>
                      <sp:UsernameToken
sp:IncludeToken="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy/IncludeToken/AlwaysToRe
cipient">
                        <wsp:Policy>
                          <sp:WssUsernameToken10></sp:WssUsernameToken10>
                        </wsp:Policy>
                      </sp:UsernameToken>
                    </wsp:Policy>
                  </sp:SignedSupportingTokens>
                  <sp:Wss11>
                    <wsp:Policy>
                      <sp:MustSupportRefKeyIdentifier></sp:MustSupportRefKeyIdentifier>
                      <sp:MustSupportRefIssuerSerial></sp:MustSupportRefIssuerSerial>
                      <sp:MustSupportRefThumbprint></sp:MustSupportRefThumbprint>
                      <sp:MustSupportRefEncryptedKey></sp:MustSupportRefEncryptedKey>
                    </wsp:Policy>
                  </sp:Wss11>
```

```
                    <sp:Trust10>
                      <wsp:Policy>
                        <sp:MustSupportIssuedTokens></sp:MustSupportIssuedTokens>
                        <sp:RequireClientEntropy></sp:RequireClientEntropy>
                        <sp:RequireServerEntropy></sp:RequireServerEntropy>
                      </wsp:Policy>
                    </sp:Trust10>
                  </wsp:Policy>
                </sp:BootstrapPolicy>
              </wsp:Policy>
            </sp:SecureConversationToken>
            <sp:SignedParts>
              <sp:Header Name="To"
Namespace="http://www.w3.org/2005/08/addressing"></sp:Header>
            </sp:SignedParts>
          </wsp:Policy>
        </sp:EndorsingSupportingTokens>
        <sp:Wss11 xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
          <wsp:Policy>
            <sp:MustSupportRefKeyIdentifier></sp:MustSupportRefKeyIdentifier>
            <sp:MustSupportRefIssuerSerial></sp:MustSupportRefIssuerSerial>
            <sp:MustSupportRefThumbprint></sp:MustSupportRefThumbprint>
            <sp:MustSupportRefEncryptedKey></sp:MustSupportRefEncryptedKey>
          </wsp:Policy>
        </sp:Wss11>
        <sp:Trust10 xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
          <wsp:Policy>
            <sp:MustSupportIssuedTokens></sp:MustSupportIssuedTokens>
            <sp:RequireClientEntropy></sp:RequireClientEntropy>
            <sp:RequireServerEntropy></sp:RequireServerEntropy>
          </wsp:Policy>
        </sp:Trust10>
        <wsam:Addressing>
          <wsp:Policy>
            <wsam:AnonymousResponses></wsam:AnonymousResponses>
          </wsp:Policy>
        </wsam:Addressing>
      </wsp:All>
    </wsp:ExactlyOne>
  </wsp:Policy>
  <wsp:Policy wsu:Id="NetTcpBinding_TopologyManagement1_policy">
    <wsdl:documentation>
      Security policy for ADCAP TopologyManagement port type using
      username/password authentication.
    </wsdl:documentation>
    <wsp:ExactlyOne>
      <wsp:All>
        <msb:BinaryEncoding
xmlns:msb="http://schemas.microsoft.com/ws/06/2004/mspolicy/netbinary1"></msb:BinaryEncoding>
        <sp:TransportBinding xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
          <wsp:Policy>
            <sp:TransportToken>
              <wsp:Policy>
                <msf:SslTransportSecurity
xmlns:msf="http://schemas.microsoft.com/ws/2006/05/framing/policy"></msf:SslTransportSecurity
>
              </wsp:Policy>
            </sp:TransportToken>
            <sp:AlgorithmSuite>
```

```
              <wsp:Policy>
                <sp:Basic256></sp:Basic256>
              </wsp:Policy>
            </sp:AlgorithmSuite>
            <sp:Layout>
              <wsp:Policy>
                <sp:Strict></sp:Strict>
              </wsp:Policy>
            </sp:Layout>
            <sp:IncludeTimestamp></sp:IncludeTimestamp>
          </wsp:Policy>
        </sp:TransportBinding>
        <sp:EndorsingSupportingTokens
xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
          <wsp:Policy>
            <sp:SecureConversationToken
sp:IncludeToken="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy/IncludeToken/AlwaysToRe
cipient">
              <wsp:Policy>
                <sp:BootstrapPolicy>
                  <wsp:Policy>
                    <sp:SignedParts>
                      <sp:Body></sp:Body>
                      <sp:Header Name="To"
Namespace="http://www.w3.org/2005/08/addressing"></sp:Header>
                      <sp:Header Name="From"
Namespace="http://www.w3.org/2005/08/addressing"></sp:Header>
                      <sp:Header Name="FaultTo"
Namespace="http://www.w3.org/2005/08/addressing"></sp:Header>
                      <sp:Header Name="ReplyTo"
Namespace="http://www.w3.org/2005/08/addressing"></sp:Header>
                      <sp:Header Name="MessageID"
Namespace="http://www.w3.org/2005/08/addressing"></sp:Header>
                      <sp:Header Name="RelatesTo"
Namespace="http://www.w3.org/2005/08/addressing"></sp:Header>
                      <sp:Header Name="Action"
Namespace="http://www.w3.org/2005/08/addressing"></sp:Header>
                    </sp:SignedParts>
                    <sp:EncryptedParts>
                      <sp:Body></sp:Body>
                    </sp:EncryptedParts>
                    <sp:TransportBinding>
                      <wsp:Policy>
                        <sp:TransportToken>
                          <wsp:Policy>
                            <msf:SslTransportSecurity
xmlns:msf="http://schemas.microsoft.com/ws/2006/05/framing/policy"></msf:SslTransportSecurity
>
                          </wsp:Policy>
                        </sp:TransportToken>
                        <sp:AlgorithmSuite>
                          <wsp:Policy>
                            <sp:Basic256></sp:Basic256>
                          </wsp:Policy>
                        </sp:AlgorithmSuite>
                        <sp:Layout>
                          <wsp:Policy>
                            <sp:Strict></sp:Strict>
                          </wsp:Policy>
                        </sp:Layout>
```

```
                        <sp:IncludeTimestamp></sp:IncludeTimestamp>
                      </wsp:Policy>
                    </sp:TransportBinding>
                    <sp:SignedSupportingTokens>
                      <wsp:Policy>
                        <sp:UsernameToken
sp:IncludeToken="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy/IncludeToken/AlwaysToRe
cipient">
                          <wsp:Policy>
                            <sp:WssUsernameToken10></sp:WssUsernameToken10>
                          </wsp:Policy>
                        </sp:UsernameToken>
                      </wsp:Policy>
                    </sp:SignedSupportingTokens>
                    <sp:Wss11>
                      <wsp:Policy>
                        <sp:MustSupportRefKeyIdentifier></sp:MustSupportRefKeyIdentifier>
                        <sp:MustSupportRefIssuerSerial></sp:MustSupportRefIssuerSerial>
                        <sp:MustSupportRefThumbprint></sp:MustSupportRefThumbprint>
                        <sp:MustSupportRefEncryptedKey></sp:MustSupportRefEncryptedKey>
                      </wsp:Policy>
                    </sp:Wss11>
                    <sp:Trust10>
                      <wsp:Policy>
                        <sp:MustSupportIssuedTokens></sp:MustSupportIssuedTokens>
                        <sp:RequireClientEntropy></sp:RequireClientEntropy>
                        <sp:RequireServerEntropy></sp:RequireServerEntropy>
                      </wsp:Policy>
                    </sp:Trust10>
                  </wsp:Policy>
                </sp:BootstrapPolicy>
              </wsp:Policy>
            </sp:SecureConversationToken>
            <sp:SignedParts>
              <sp:Header Name="To"
Namespace="http://www.w3.org/2005/08/addressing"></sp:Header>
            </sp:SignedParts>
          </wsp:Policy>
        </sp:EndorsingSupportingTokens>
        <sp:Wss11 xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
          <wsp:Policy>
            <sp:MustSupportRefKeyIdentifier></sp:MustSupportRefKeyIdentifier>
            <sp:MustSupportRefIssuerSerial></sp:MustSupportRefIssuerSerial>
            <sp:MustSupportRefThumbprint></sp:MustSupportRefThumbprint>
            <sp:MustSupportRefEncryptedKey></sp:MustSupportRefEncryptedKey>
          </wsp:Policy>
        </sp:Wss11>
        <sp:Trust10 xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
          <wsp:Policy>
            <sp:MustSupportIssuedTokens></sp:MustSupportIssuedTokens>
            <sp:RequireClientEntropy></sp:RequireClientEntropy>
            <sp:RequireServerEntropy></sp:RequireServerEntropy>
          </wsp:Policy>
        </sp:Trust10>
        <wsam:Addressing>
          <wsp:Policy>
            <wsam:AnonymousResponses></wsam:AnonymousResponses>
          </wsp:Policy>
        </wsam:Addressing>
```

```
      </wsp:All>
    </wsp:ExactlyOne>
  </wsp:Policy>
  <wsdl:import namespace="http://schemas.xmlsoap.org/ws/2004/09/transfer" location="" />
  <wsdl:import namespace="http://schemas.xmlsoap.org/ws/2004/09/enumeration" location="" />
  <wsdl:import namespace="http://schemas.microsoft.com/2008/1/ActiveDirectory/CustomActions"
location="" />
  <wsdl:types />
  <wsdl:binding name="NetTcpBinding_Resource" type="wxf:Resource">
    <wsdl:documentation>
      Binding for the WS-Transfer Resource port type on the net.tcp transport using SOAP 1.2
      using Windows Integrated authentication.
    </wsdl:documentation>
    <wsp:PolicyReference URI="#NetTcpBinding_Resource_policy"></wsp:PolicyReference>
    <soap12:binding transport="http://schemas.microsoft.com/soap/tcp" />
    <wsdl:operation name="Get">
      <soap12:operation soapAction="http://schemas.xmlsoap.org/ws/2004/09/transfer/Get"
style="document" />
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="Put">
      <soap12:operation soapAction="http://schemas.xmlsoap.org/ws/2004/09/transfer/Put"
style="document" />
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="Delete">
      <soap12:operation soapAction="http://schemas.xmlsoap.org/ws/2004/09/transfer/Delete"
style="document" />
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="NetTcpBinding_ResourceFactory" type="wxf:ResourceFactory">
    <wsdl:documentation>
      Binding for the WS-Transfer ResourceFactory port type on the net.tcp transport using
SOAP 1.2
      using Windows Integrated authentication.
    </wsdl:documentation>
    <wsp:PolicyReference URI="#NetTcpBinding_ResourceFactory_policy"></wsp:PolicyReference>
    <soap12:binding transport="http://schemas.microsoft.com/soap/tcp" />
    <wsdl:operation name="Create">
      <soap12:operation soapAction="http://schemas.xmlsoap.org/ws/2004/09/transfer/Create"
style="document" />
      <wsdl:input>
        <soap12:body use="literal" />
```

```
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="NetTcpBinding_Search" type="wsen:Search">
    <wsdl:documentation>
      Binding for the WS-Enumeration Search port type on the net.tcp transport using SOAP 1.2
      using Windows Integrated authentication.
    </wsdl:documentation>
    <wsp:PolicyReference URI="#NetTcpBinding_Search_policy"></wsp:PolicyReference>
    <soap12:binding transport="http://schemas.microsoft.com/soap/tcp" />
    <wsdl:operation name="Enumerate">
      <soap12:operation
soapAction="http://schemas.xmlsoap.org/ws/2004/09/enumeration/Enumerate" style="document" />
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="Pull">
      <soap12:operation soapAction="http://schemas.xmlsoap.org/ws/2004/09/enumeration/Pull"
style="document" />
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="Renew">
      <soap12:operation soapAction="http://schemas.xmlsoap.org/ws/2004/09/enumeration/Renew"
style="document" />
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="GetStatus">
      <soap12:operation
soapAction="http://schemas.xmlsoap.org/ws/2004/09/enumeration/GetStatus" style="document" />
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="Release">
      <soap12:operation
soapAction="http://schemas.xmlsoap.org/ws/2004/09/enumeration/Release" style="document" />
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>
```

```
      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="NetTcpBinding_AccountManagement" type="ca:AccountManagement">
    <wsdl:documentation>
      Binding for the ADCAP AccountManagement port type on the net.tcp transport using SOAP
1.2
      using Windows Integrated authentication.
    </wsdl:documentation>
    <wsp:PolicyReference URI="#NetTcpBinding_AccountManagement_policy"></wsp:PolicyReference>
    <soap12:binding transport="http://schemas.microsoft.com/soap/tcp" />
    <wsdl:operation name="GetADGroupMember">
      <soap12:operation
soapAction="http://schemas.microsoft.com/2008/1/ActiveDirectory/CustomActions/AccountManageme
nt/GetADGroupMember" style="document" />
      <wsdl:input name="GetADGroupMemberRequest">
        <soap12:header message="ca:GetADGroupMemberRequest_Headers" part="Server"
use="literal" />
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output name="GetADGroupMemberResponse">
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="GetADPrincipalGroupMembership">
      <soap12:operation
soapAction="http://schemas.microsoft.com/2008/1/ActiveDirectory/CustomActions/AccountManageme
nt/GetADPrincipalGroupMembership" style="document" />
      <wsdl:input name="GetADPrincipalGroupMembershipRequest">
        <soap12:header message="ca:GetADPrincipalGroupMembershipRequest_Headers"
part="Server" use="literal" />
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output name="GetADPrincipalGroupMembershipResponse">
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="SetPassword">
      <soap12:operation
soapAction="http://schemas.microsoft.com/2008/1/ActiveDirectory/CustomActions/AccountManageme
nt/SetPassword" style="document" />
      <wsdl:input name="SetPasswordRequest">
        <soap12:header message="ca:SetPasswordRequest_Headers" part="Server" use="literal" />
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output name="SetPasswordResponse">
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="ChangePassword">
      <soap12:operation
soapAction="http://schemas.microsoft.com/2008/1/ActiveDirectory/CustomActions/AccountManageme
nt/ChangePassword" style="document" />
      <wsdl:input name="ChangePasswordRequest">
        <soap12:header message="ca:ChangePasswordRequest_Headers" part="Server" use="literal"
/>
        <soap12:body use="literal" />
      </wsdl:input>
```

```
      <wsdl:output name="ChangePasswordResponse">
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="GetADPrincipalAuthorizationGroup">
      <soap12:operation
soapAction="http://schemas.microsoft.com/2008/1/ActiveDirectory/CustomActions/AccountManageme
nt/GetADPrincipalAuthorizationGroup" style="document" />
      <wsdl:input name="GetADPrincipalAuthorizationGroupRequest">
        <soap12:header message="ca:GetADPrincipalAuthorizationGroupRequest_Headers"
part="Server" use="literal" />
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output name="GetADPrincipalAuthorizationGroupResponse">
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="TranslateName">
      <soap12:operation
soapAction="http://schemas.microsoft.com/2008/1/ActiveDirectory/CustomActions/AccountManageme
nt/TranslateName" style="document" />
      <wsdl:input name="TranslateNameRequest">
        <soap12:header message="ca:TranslateNameRequest_Headers" part="Server" use="literal"
/>
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output name="TranslateNameResponse">
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="NetTcpBinding_TopologyManagement" type="ca:TopologyManagement">
    <wsdl:documentation>
      Binding for the ADCAP TopologyManagement port type on the net.tcp transport using SOAP
1.2
      using Windows Integrated authentication.
    </wsdl:documentation>
    <wsp:PolicyReference
URI="#NetTcpBinding_TopologyManagement_policy"></wsp:PolicyReference>
    <soap12:binding transport="http://schemas.microsoft.com/soap/tcp" />
    <wsdl:operation name="GetADDomainController">
      <soap12:operation
soapAction="http://schemas.microsoft.com/2008/1/ActiveDirectory/CustomActions/TopologyManagem
ent/GetADDomainController" style="document" />
      <wsdl:input name="GetADDomainControllerRequest">
        <soap12:header message="ca:GetADDomainControllerRequest_Headers" part="Server"
use="literal" />
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output name="GetADDomainControllerResponse">
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="GetADDomain">
      <soap12:operation
soapAction="http://schemas.microsoft.com/2008/1/ActiveDirectory/CustomActions/TopologyManagem
ent/GetADDomain" style="document" />
      <wsdl:input name="GetADDomainRequest">
        <soap12:header message="ca:GetADDomainRequest_Headers" part="Server" use="literal" />
        <soap12:body use="literal" />
```

```
      </wsdl:input>
      <wsdl:output name="GetADDomainResponse">
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="MoveADOperationMasterRole">
      <soap12:operation
soapAction="http://schemas.microsoft.com/2008/1/ActiveDirectory/CustomActions/TopologyManagem
ent/MoveADOperationMasterRole" style="document" />
      <wsdl:input name="MoveADOperationMasterRoleRequest">
        <soap12:header message="ca:MoveADOperationMasterRoleRequest_Headers" part="Server"
use="literal" />
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output name="MoveADOperationMasterRoleResponse">
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="GetADForest">
      <soap12:operation
soapAction="http://schemas.microsoft.com/2008/1/ActiveDirectory/CustomActions/TopologyManagem
ent/GetADForest" style="document" />
      <wsdl:input name="GetADForestRequest">
        <soap12:header message="ca:GetADForestRequest_Headers" part="Server" use="literal" />
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output name="GetADForestResponse">
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="ChangeOptionalFeature">
      <soap12:operation
soapAction="http://schemas.microsoft.com/2008/1/ActiveDirectory/CustomActions/TopologyManagem
ent/ChangeOptionalFeature" style="document" />
      <wsdl:input name="ChangeOptionalFeatureRequest">
        <soap12:header message="ca:ChangeOptionalFeatureRequest_Headers" part="Server"
use="literal" />
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output name="ChangeOptionalFeatureResponse">
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="NetTcpBinding_Resource1" type="wxf:Resource">
    <wsdl:documentation>
      Binding for the WS-Transfer Resource port type on the net.tcp transport using SOAP 1.2
      using username/password authentication.
    </wsdl:documentation>
    <wsp:PolicyReference URI="#NetTcpBinding_Resource1_policy"></wsp:PolicyReference>
    <soap12:binding transport="http://schemas.microsoft.com/soap/tcp" />
    <wsdl:operation name="Get">
      <soap12:operation soapAction="http://schemas.xmlsoap.org/ws/2004/09/transfer/Get"
style="document" />
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
```

```
    </wsdl:operation>
    <wsdl:operation name="Put">
      <soap12:operation soapAction="http://schemas.xmlsoap.org/ws/2004/09/transfer/Put"
style="document" />
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="Delete">
      <soap12:operation soapAction="http://schemas.xmlsoap.org/ws/2004/09/transfer/Delete"
style="document" />
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="NetTcpBinding_ResourceFactory1" type="wxf:ResourceFactory">
    <wsdl:documentation>
      Binding for the WS-Transfer ResourceFactory port type on the net.tcp transport using
SOAP 1.2
      using username/password authentication.
    </wsdl:documentation>
    <wsp:PolicyReference URI="#NetTcpBinding_ResourceFactory1_policy"></wsp:PolicyReference>
    <soap12:binding transport="http://schemas.microsoft.com/soap/tcp" />
    <wsdl:operation name="Create">
      <soap12:operation soapAction="http://schemas.xmlsoap.org/ws/2004/09/transfer/Create"
style="document" />
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="NetTcpBinding_Search1" type="wsen:Search">
    <wsdl:documentation>
      Binding for the WS-Enumeration Search port type on the net.tcp transport using SOAP 1.2
      using username/password authentication.
    </wsdl:documentation>
    <wsp:PolicyReference URI="#NetTcpBinding_Search1_policy"></wsp:PolicyReference>
    <soap12:binding transport="http://schemas.microsoft.com/soap/tcp" />
    <wsdl:operation name="Enumerate">
      <soap12:operation
soapAction="http://schemas.xmlsoap.org/ws/2004/09/enumeration/Enumerate" style="document" />
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="Pull">
```

```
      <soap12:operation soapAction="http://schemas.xmlsoap.org/ws/2004/09/enumeration/Pull"
style="document" />
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="Renew">
      <soap12:operation soapAction="http://schemas.xmlsoap.org/ws/2004/09/enumeration/Renew"
style="document" />
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="GetStatus">
      <soap12:operation
soapAction="http://schemas.xmlsoap.org/ws/2004/09/enumeration/GetStatus" style="document" />
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="Release">
      <soap12:operation
soapAction="http://schemas.xmlsoap.org/ws/2004/09/enumeration/Release" style="document" />
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="NetTcpBinding_AccountManagement1" type="ca:AccountManagement">
    <wsdl:documentation>
      Binding for the ADCAP AccountManagement port type on the net.tcp transport using SOAP
1.2
      using username/password authentication.
    </wsdl:documentation>
    <wsp:PolicyReference
URI="#NetTcpBinding_AccountManagement1_policy"></wsp:PolicyReference>
    <soap12:binding transport="http://schemas.microsoft.com/soap/tcp" />
    <wsdl:operation name="GetADGroupMember">
      <soap12:operation
soapAction="http://schemas.microsoft.com/2008/1/ActiveDirectory/CustomActions/AccountManageme
nt/GetADGroupMember" style="document" />
      <wsdl:input name="GetADGroupMemberRequest">
        <soap12:header message="ca:GetADGroupMemberRequest_Headers" part="Server"
use="literal" />
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output name="GetADGroupMemberResponse">
        <soap12:body use="literal" />
```

```
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="GetADPrincipalGroupMembership">
      <soap12:operation
soapAction="http://schemas.microsoft.com/2008/1/ActiveDirectory/CustomActions/AccountManageme
nt/GetADPrincipalGroupMembership" style="document" />
      <wsdl:input name="GetADPrincipalGroupMembershipRequest">
        <soap12:header message="ca:GetADPrincipalGroupMembershipRequest_Headers"
part="Server" use="literal" />
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output name="GetADPrincipalGroupMembershipResponse">
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="SetPassword">
      <soap12:operation
soapAction="http://schemas.microsoft.com/2008/1/ActiveDirectory/CustomActions/AccountManageme
nt/SetPassword" style="document" />
      <wsdl:input name="SetPasswordRequest">
        <soap12:header message="ca:SetPasswordRequest_Headers" part="Server" use="literal" />
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output name="SetPasswordResponse">
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="ChangePassword">
      <soap12:operation
soapAction="http://schemas.microsoft.com/2008/1/ActiveDirectory/CustomActions/AccountManageme
nt/ChangePassword" style="document" />
      <wsdl:input name="ChangePasswordRequest">
        <soap12:header message="ca:ChangePasswordRequest_Headers" part="Server" use="literal"
/>
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output name="ChangePasswordResponse">
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="GetADPrincipalAuthorizationGroup">
      <soap12:operation
soapAction="http://schemas.microsoft.com/2008/1/ActiveDirectory/CustomActions/AccountManageme
nt/GetADPrincipalAuthorizationGroup" style="document" />
      <wsdl:input name="GetADPrincipalAuthorizationGroupRequest">
        <soap12:header message="ca:GetADPrincipalAuthorizationGroupRequest_Headers"
part="Server" use="literal" />
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output name="GetADPrincipalAuthorizationGroupResponse">
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="TranslateName">
      <soap12:operation
soapAction="http://schemas.microsoft.com/2008/1/ActiveDirectory/CustomActions/AccountManageme
nt/TranslateName" style="document" />
      <wsdl:input name="TranslateNameRequest">
        <soap12:header message="ca:TranslateNameRequest_Headers" part="Server" use="literal"
/>
```

```
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output name="TranslateNameResponse">
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="NetTcpBinding_TopologyManagement1" type="ca:TopologyManagement">
    <wsdl:documentation>
      Binding for the ADCAP TopologyManagement port type on the net.tcp transport using SOAP
1.2
      using username/password authentication.
    </wsdl:documentation>
    <wsp:PolicyReference
URI="#NetTcpBinding_TopologyManagement1_policy"></wsp:PolicyReference>
    <soap12:binding transport="http://schemas.microsoft.com/soap/tcp" />
    <wsdl:operation name="GetADDomainController">
      <soap12:operation
soapAction="http://schemas.microsoft.com/2008/1/ActiveDirectory/CustomActions/TopologyManagem
ent/GetADDomainController" style="document" />
      <wsdl:input name="GetADDomainControllerRequest">
        <soap12:header message="ca:GetADDomainControllerRequest_Headers" part="Server"
use="literal" />
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output name="GetADDomainControllerResponse">
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="GetADDomain">
      <soap12:operation
soapAction="http://schemas.microsoft.com/2008/1/ActiveDirectory/CustomActions/TopologyManagem
ent/GetADDomain" style="document" />
      <wsdl:input name="GetADDomainRequest">
        <soap12:header message="ca:GetADDomainRequest_Headers" part="Server" use="literal" />
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output name="GetADDomainResponse">
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="MoveADOperationMasterRole">
      <soap12:operation
soapAction="http://schemas.microsoft.com/2008/1/ActiveDirectory/CustomActions/TopologyManagem
ent/MoveADOperationMasterRole" style="document" />
      <wsdl:input name="MoveADOperationMasterRoleRequest">
        <soap12:header message="ca:MoveADOperationMasterRoleRequest_Headers" part="Server"
use="literal" />
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output name="MoveADOperationMasterRoleResponse">
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="GetADForest">
      <soap12:operation
soapAction="http://schemas.microsoft.com/2008/1/ActiveDirectory/CustomActions/TopologyManagem
ent/GetADForest" style="document" />
      <wsdl:input name="GetADForestRequest">
        <soap12:header message="ca:GetADForestRequest_Headers" part="Server" use="literal" />
```

```
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output name="GetADForestResponse">
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="ChangeOptionalFeature">
      <soap12:operation
soapAction="http://schemas.microsoft.com/2008/1/ActiveDirectory/CustomActions/TopologyManagem
ent/ChangeOptionalFeature" style="document" />
      <wsdl:input name="ChangeOptionalFeatureRequest">
        <soap12:header message="ca:ChangeOptionalFeatureRequest_Headers" part="Server"
use="literal" />
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output name="ChangeOptionalFeatureResponse">
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="ActiveDirectoryWebService">
    <wsdl:documentation>
      Service consisting of the WS-Transfer, WS-Enumeration, and ADCAP
      protocols operating on the endpoints specified in section 2.1
      of [MS-ADDM].
    </wsdl:documentation>
    <wsdl:port name="NetTcpBinding_Resource" binding="tns:NetTcpBinding_Resource">
      <soap12:address
location="net.tcp://localhost:9389/ActiveDirectoryWebServices/Windows/Resource" />
      <wsa10:EndpointReference>

<wsa10:Address>net.tcp://localhost:9389/ActiveDirectoryWebServices/Windows/Resource</wsa10:Ad
dress>
      </wsa10:EndpointReference>
    </wsdl:port>
    <wsdl:port name="NetTcpBinding_ResourceFactory"
binding="tns:NetTcpBinding_ResourceFactory">
      <soap12:address
location="net.tcp://localhost:9389/ActiveDirectoryWebServices/Windows/ResourceFactory" />
      <wsa10:EndpointReference>

<wsa10:Address>net.tcp://localhost:9389/ActiveDirectoryWebServices/Windows/ResourceFactory</w
sa10:Address>
      </wsa10:EndpointReference>
    </wsdl:port>
    <wsdl:port name="NetTcpBinding_Search" binding="tns:NetTcpBinding_Search">
      <soap12:address
location="net.tcp://localhost:9389/ActiveDirectoryWebServices/Windows/Enumeration" />
      <wsa10:EndpointReference>

<wsa10:Address>net.tcp://localhost:9389/ActiveDirectoryWebServices/Windows/Enumeration</wsa10
:Address>
      </wsa10:EndpointReference>
    </wsdl:port>
    <wsdl:port name="NetTcpBinding_AccountManagement"
binding="tns:NetTcpBinding_AccountManagement">
      <soap12:address
location="net.tcp://localhost:9389/ActiveDirectoryWebServices/Windows/AccountManagement" />
      <wsa10:EndpointReference>
```

```
<wsa10:Address>net.tcp://localhost:9389/ActiveDirectoryWebServices/Windows/AccountManagement<
/wsa10:Address>
      </wsa10:EndpointReference>
    </wsdl:port>
    <wsdl:port name="NetTcpBinding_TopologyManagement"
binding="tns:NetTcpBinding_TopologyManagement">
      <soap12:address
location="net.tcp://localhost:9389/ActiveDirectoryWebServices/Windows/TopologyManagement" />
      <wsa10:EndpointReference>

<wsa10:Address>net.tcp://localhost:9389/ActiveDirectoryWebServices/Windows/TopologyManagement
</wsa10:Address>
      </wsa10:EndpointReference>
    </wsdl:port>
    <wsdl:port name="NetTcpBinding_Resource1" binding="tns:NetTcpBinding_Resource1">
      <soap12:address
location="net.tcp://localhost:9389/ActiveDirectoryWebServices/UserName/Resource" />
      <wsa10:EndpointReference>

<wsa10:Address>net.tcp://localhost:9389/ActiveDirectoryWebServices/UserName/Resource</wsa10:A
ddress>
      </wsa10:EndpointReference>
    </wsdl:port>
    <wsdl:port name="NetTcpBinding_ResourceFactory1"
binding="tns:NetTcpBinding_ResourceFactory1">
      <soap12:address
location="net.tcp://localhost:9389/ActiveDirectoryWebServices/UserName/ResourceFactory" />
      <wsa10:EndpointReference>

<wsa10:Address>net.tcp://localhost:9389/ActiveDirectoryWebServices/UserName/ResourceFactory</
wsa10:Address>
      </wsa10:EndpointReference>
    </wsdl:port>
    <wsdl:port name="NetTcpBinding_Search1" binding="tns:NetTcpBinding_Search1">
      <soap12:address
location="net.tcp://localhost:9389/ActiveDirectoryWebServices/UserName/Enumeration" />
      <wsa10:EndpointReference>

<wsa10:Address>net.tcp://localhost:9389/ActiveDirectoryWebServices/UserName/Enumeration</wsa1
0:Address>
      </wsa10:EndpointReference>
    </wsdl:port>
    <wsdl:port name="NetTcpBinding_AccountManagement1"
binding="tns:NetTcpBinding_AccountManagement1">
      <soap12:address
location="net.tcp://localhost:9389/ActiveDirectoryWebServices/UserName/AccountManagement" />
      <wsa10:EndpointReference>

<wsa10:Address>net.tcp://localhost:9389/ActiveDirectoryWebServices/UserName/AccountManagement
</wsa10:Address>
      </wsa10:EndpointReference>
    </wsdl:port>
    <wsdl:port name="NetTcpBinding_TopologyManagement1"
binding="tns:NetTcpBinding_TopologyManagement1">
      <soap12:address
location="net.tcp://localhost:9389/ActiveDirectoryWebServices/UserName/TopologyManagement" />
      <wsa10:EndpointReference>

<wsa10:Address>net.tcp://localhost:9389/ActiveDirectoryWebServices/UserName/TopologyManagemen
t</wsa10:Address>
```

```
        </wsa10:EndpointReference>
      </wsdl:port>
    </wsdl:service>
  </wsdl:definitions>
```

9 Appendix B: Product Behavior

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

- Active Directory Application Mode (ADAM)
- Active Directory Lightweight Directory Services (AD LDS) for Windows Vista
- Active Directory Lightweight Directory Services (AD LDS) for Windows 7
- Active Directory Lightweight Directory Services (AD LDS) for Windows 8 operating system
- Active Directory Lightweight Directory Services (AD LDS) for Windows 8.1 operating system
- Windows 2000 operating system
- Windows XP operating system
- Windows Server 2003 operating system
- Windows Server 2003 R2 operating system
- Windows Vista operating system
- Windows Server 2008 operating system
- Windows 7 operating system
- Windows Server 2008 R2 operating system
- Windows 8 operating system
- Windows Server 2012 operating system
- Windows 8.1 operating system
- Windows Server 2012 R2 operating system

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

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

<1> Section 2.1: Active Directory Lightweight Directory Services is available in ADAM, Windows Server 2008, AD LDS for Windows Vista, Windows Server 2008 R2, AD LDS for Windows 7, Windows Server 2012, AD LDS for Windows 8, Windows Server 2012 R2, and AD LDS for Windows 8.1. In Windows XP and Windows Server 2003, ADAM was available as a web download and was later packaged with Windows Server 2003 R2.

<2> Section 4.1: [RFC2136] allows dynamic update responses to be formed in the following two ways.

1. Respond with the ZOCOUNT, PRCOUNT, UPCOUNT and ADCOUNT fields and corresponding sections copied from the request.

2. Respond with the ZOCOUNT, PRCOUNT, UPCOUNT and ADCOUNT fields set to 0 and without copying the corresponding sections from the request.

The Windows DNS server in Windows NT, Windows 2000, Windows Server 2003,Windows Server 2003 R2, Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, and Windows Server 2012 R2 use Method 1 when formatting dynamic update responses. The Windows DNS client in Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 of the Windows DNS client, expect Method 1 when parsing dynamic update responses and may log an error when parsing dynamic update responses that use Method 2. The Windows DNS client in Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, and Windows Server 2012 R2 will accept either method of formatting dynamic update responses.

<3> Section 4.5: Web services support is included in the Windows Server 2008 R2, Windows Server 2012, and Windows Server 2012 R2 versions of Active Directory Domain Services (AD DS) and Active Directory Lightweight Directory Services (AD LDS). Web services support for AD DS and AD LDS is also available as a separate installable package (namely, Active Directory Management Gateway Service) for the following operating systems: Windows Server 2003 SP2, Windows Server 2003 R2 SP2, and Windows Server 2008.

<4> Section 6.2.6.2.3: This is indicated to the Windows SSPI packages with an identity of NULL.

<5> Section 6.2.6.2.7: The Microsoft implementation of LDAP client sets the resultCode of LDAPResult to LDAP_SERVER_DOWN (0x51) when the directory server is unreachable ([MS-ERREF] section 2.4, LDAP error to Win32 mapping).

<6> Section 6.2.6.2.7: The Microsoft implementation of LDAP client sets the resultCode of LDAPResult to LDAP_SERVER_DOWN (0x51) when the directory server is unreachable ([MS-ERREF] section 2.4, LDAP error to Win32 mapping).

<7> Section 6.2.6.3.1: The Microsoft implementation of LDAP client sets the resultCode of LDAPResult to LDAP_SERVER_DOWN (0x51) when the directory server is unreachable ([MS-ERREF] section 2.4, LDAP error to Win32 mapping).

<8> Section 6.2.6.4.1: The Microsoft implementation of LDAP client sets the resultCode of LDAPResult to LDAP_TIMEOUT (0x55) on timer expiration ([MS-ERREF] section 2.4, LDAP error to Win32 mapping).

<9> Section 6.2.7.3.2: The Microsoft implementation of LDAP client sets the resultCode of LDAPResult to LDAP_SERVER_DOWN (0x51) when the directory server is unreachable ([MS-ERREF] section 2.4, LDAP error to Win32 mapping).

<10> Section 6.2.7.3.2: The Microsoft implementation of LDAP client sets the resultCode of LDAPResult to LDAP_TIMEOUT (0x55) on timer expiration ([MS-ERREF] section 2.4, LDAP error to Win32 mapping).

<11> Section 6.8: The Web Services protocol group is used in the Windows Server 2008 R2, Windows Server 2012, and Windows Server 2012 R2 versions of the Active Directory System for administering the system. It can also be used on Windows Server 2003 SP2, Windows Server 2003 R2 SP2, and Windows Server 2008 versions upon installation of Active Directory Management Gateway Service.

<12> Section 7.2: The Microsoft implementation of Active Directory Web Services permits administrators to configure settings used to limit the amount of server resources that can be consumed in processing a request. Examples of such settings are included below for illustrative

purposes. Implementations are free to implement some, all, or none of these settings, as well as to implement other settings of their own devising.

Common to implementations of WS-Transfer, WS-Enumeration, and ADCAP:

- Maximum size of a SOAP request message that the directory service will accept.
- Maximum level of nested XML elements in the SOAP request message that the directory service will accept.
- Maximum length of strings and maximum number of elements in arrays in the SOAP request message that the directory service will accept.
- Maximum number of concurrent requests that the directory service will process at one time.
- Maximum number of concurrent requests from one user that the directory service will process at one time.
- Length of time the directory service will wait when performing an operation before the operation will be timed-out.

<13> Section 7.2: The Microsoft implementation of Active Directory Web Services permits administrators to configure settings used to limit the amount of server resources that can be consumed in processing a request. Examples of such settings are included below for illustrative purposes. Implementations are free to implement some, all, or none of these settings, as well as to implement other settings of their own devising.

Common to implementations of WS-Transfer, WS-Enumeration, and ADCAP:

- Maximum size of a SOAP request message that the directory service will accept.
- Maximum level of nested XML elements in the SOAP request message that the directory service will accept.
- Maximum length of strings and maximum number of elements in arrays in the SOAP request message that the directory service will accept.
- Maximum number of concurrent requests that the directory service will process at one time.
- Maximum number of concurrent requests from one user that the directory service will process at one time.
- Length of time the directory service will wait when performing an operation before the operation will be timed-out.

Specific to implementations of WS-Enumeration:

- Maximum total number of **enumeration contexts** that can exist at one time.
- Length of time an **enumeration context** can be left idle by a client before the directory service will automatically release it.
- Length of time an **enumeration context** can be kept open (whether idle or in use) by a client before the directory service will automatically release it.
- Maximum expiration time the directory service will permit a client to specify in an Enumerate or Renew request (via the Expires element).

- Maximum time the directory service will permit a client to specify in a Pull request (via the MaxTime element).
- Maximum number of elements the directory service will permit a client to specify in a Pull request (via the MaxElements element).

<14> Section 7.2: The Microsoft implementation of Active Directory Web Services permits administrators to configure settings used to limit the amount of server resources that can be consumed in processing a request. Examples of such settings are included below for illustrative purposes. Implementations are free to implement some, all, or none of these settings, as well as to implement other settings of their own devising.

Common to implementations of WS-Transfer, WS-Enumeration, and ADCAP:

- Maximum size of a SOAP request message that the directory service will accept.
- Maximum level of nested XML elements in the SOAP request message that the directory service will accept.
- Maximum length of strings and maximum number of elements in arrays in the SOAP request message that the directory service will accept.
- Maximum number of concurrent requests that the directory service will process at one time.
- Maximum number of concurrent requests from one user that the directory service will process at one time.
- Length of time the directory service will wait when performing an operation before the operation will be timed-out.

<15> Section 8: In the Microsoft implementation, SPN claims are supplied by the server in the WSDL for the following WSDL ports:

- NetTcpBinding_Resource
- NetTcpBinding_ResourceFactory
- NetTcpBinding_Search
- NetTcpBinding_AccountManagement
- NetTcpBinding_TopologyManagement

These SPN claims contain an SPN of the form "host/*serverName*", where *serverName* is the fully-qualified DNS host name of the server hosting the directory service.

DNS claims are supplied by the server in the WSDL for the following ports:

- NetTcpBinding_Resource1
- NetTcpBinding_ResourceFactory1
- NetTcpBinding_Search1
- NetTcpBinding_AccountManagement1
- NetTcpBinding_TopologyManagement1

These DNS claims contain the fully-qualified DNS host name of the server hosting the directory service.

10 Change Tracking

No table of changes is available. The document is either new or has had no changes since its last release.

11 Index

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W