[MS-ADOD]:
Active Directory Protocols Overview

This document provides an overview of the Active Directory Protocols 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 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

This document provides an overview of the functionality and relationship of the protocols that make up the client-server and server-to-server behavior of Active Directory. The Active Directory protocols provide directory services 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 foundation for authentication services in a domain environment, domain services, and directory replication services in Windows. The Active Directory protocols are specified in [LDAP], [MS-ADTS], [MS-SRPL], [MS-DRSR], [MS-SNTP], [MS-LSAD], [MS-LSAT], [MS-DSSP], [MS-SAMR], [MS-SAMS], [MS-WSDS], [WXFR], [WSENUM], [MS-WSTIM], [MS-ADDM], [MS-WSPELD] and [MS-ADCAP].

This document describes the relationships of the Active Directory protocols that make up the client-server and server-to-server behavior of Active Directory. It describes the intended client-server and server-to-server functionality of Active Directory, such as directory replication. It describes how the protocols interact with each other. In the context of the Active Directory protocols, "clients" can include both Microsoft Windows client and server operating systems, other Microsoft software, and third-party software and operating systems. It provides examples of some common use cases. It does not restate the processing rules and other details that are specific for each protocol. Those details are described in the protocol specifications for each of the protocols and data structures that belong to this protocols group.

Revision Summary

Date	Revision History	Revision Class	Comments
03/30/2012	1.0	New	Released new document.
07/12/2012	2.0	Major	Significantly changed the technical content.
10/25/2012	2.1	Minor	Clarified the meaning of the technical content.
01/31/2013	2.1	No change	No changes to the meaning, language, or formatting of the technical content.
08/08/2013	3.0	Major	Significantly changed the technical content.

Contents

1 Introduction ... 6
1.1 Glossary ... 12
1.2 References ... 15

2 Functional Overview ... 20
2.1 Components and Capabilities ... 20
2.2 Relevant Standards ... 21
2.3 Protocol Relationships ... 23
2.4 Protocol Summary ... 25
2.5 Environment ... 27
2.5.1 Active Directory Protocols Dependencies ... 28
2.5.2 Dependencies on Active Directory Protocols ... 29
2.6 Assumptions and Preconditions ... 30
2.7 Use Cases ... 31
2.7.1 Object Management ... 32
2.7.1.1 Create Directory Object - Client Application ... 33
2.7.1.2 Search for Directory Object - Client Application ... 36
2.7.1.3 Modify Directory Object - Client Application ... 39
2.7.1.4 Delete Directory Object - Client Application ... 41
2.7.1.5 Create Organizational Unit - Client Application ... 43
2.7.1.6 Cross-Domain Move - Client Application ... 45
2.7.2 Identity Lifecycle Management ... 47
2.7.2.1 Create a New Account - Client Application ... 48
2.7.2.2 Reset an Existing Account's Password - Client Application ... 51
2.7.2.3 Change an Existing Account's Password (PDC) - Client Application ... 54
2.7.2.4 Change an Existing Account's Password (DC) - Client Application ... 56
2.7.2.5 Change User Account Password Against an RODC - Client Application ... 59
2.7.2.6 User Login to Domain Services Using an RODC and Updating the User LastLogonTimeStamp - Client Application ... 62
2.7.2.7 Query an Account's Group Membership - Client Application ... 63
2.7.2.8 Delete an Account - Client Application ... 65
2.7.2.9 Create a Security Group - Client Application ... 67
2.7.2.10 Modify Group Member List - Client Application ... 70
2.7.2.11 Query Members of a Group - Client Application ... 72
2.7.3 Schema Management ... 74
2.7.3.1 Add a New Class to the Schema - Client Application ... 75
2.7.3.2 Add a New Attribute to the Schema - Client Application ... 78
2.7.3.3 Add an Attribute to a Class - Client Application ... 81
2.7.4 Name Translation ... 84
2.7.4.1 Convert a SID to/from a Human-Readable Format - Client Application ... 84
2.7.5 Directory Replication ... 87
2.7.5.1 Replicate Changes Within a Domain - Domain Controller ... 88
2.7.5.2 Replicate Changes to a GC or a Partial Replica by Using RPC - Domain Controller ... 90
2.7.5.3 Transferring a FSMO Role - Domain Controller ... 92
2.7.6 Trust Management ... 94
2.7.6.1 Create a Trust - Domain Controller ... 94
2.7.7 Domain Services ... 96
2.7.7.1 Join a Domain with a New Account - Domain Client ... 96
2.7.7.2 Unjoin from the Domain - Domain Client ... 99

2.7.7.3 Supporting Use Cases 100
2.7.7.3.1 Locate a Domain Controller - Domain Client 100
2.8 Versioning, Capability Negotiation, and Extensibility 103
2.9 Error Handling 104
2.9.1 Transient Unavailability of Durable Storage 105
2.9.2 Permanent Unavailability of Durable Storage 106
2.9.3 Data Corruption 106
2.9.4 Unavailability of Networking 106
2.9.5 Unavailability of DNS 107
2.9.6 Failures while Joining or Unjoining a Domain..107
2.10 Coherency Requirements 108
2.11 Security 108
2.11.1 Security Elements..109
2.11.2 Communications Security 110
2.11.3 System Configuration Security 112
2.11.4 Internal Security 112
2.11.5 External Security 113
2.12 Additional Considerations...114

3 Examples 115
3.1 Domain-Join Examples 115
3.1.1 Example 1: Locate a Domain Controller...115
3.1.2 Example 2: Joining a Domain by Creating an Account via SAMR 118
3.1.3 Example 3: Joining a Domain by Creating an Account via LDAP..........................121
3.1.4 Example 4: Unjoining a Domain Member 125
3.2 Directory Examples 127
3.2.1 Example 1: Provision a User Account Using the LDAP Protocol 128
3.2.2 Example 2: Provision a User Account Using the SAMR Protocol...........................131
3.2.3 Example 3: Provision a User Account Using the SAMR Protocol Including the Need for a RID Allocation Request 135
3.2.4 Example 4: Change a User Account's Password 138
3.2.5 Example 5: Change a User Account's Password Against a Non-PDC DC 140
3.2.6 Example 6: Update the User's lastLogOnTimeStamp Against an RODC When the User Binds to an LDAP Server. 144
3.2.7 Example 7: Determine the Group Membership of a User....................................145
3.2.8 Example 8: Delete a User Account 148
3.2.9 Example 9: Obtain a List of User Accounts Using the Web Services Protocols 150
3.2.10 Example 10: Obtain a List of User Accounts Using the LDAP Protocol 152
3.2.11 Example 11: Manage Groups and Their Memberships......................................153
3.2.12 Example12: Delete a Group 157
3.2.13 Example 13: Extend the Schema to Support an Application by Adding a New Class 159
3.2.14 Example 14: Extend the Schema to Support an Application by Adding a New Attribute 160
3.2.15 Example 15: Extend the Schema to Support an Application by Adding an Attribute to a Class..161
3.2.16 Example 16: Partition Directory Data with Organizational Units 163
3.2.17 Example 17: Store Application Data in the Directory 165
3.2.18 Example 18: Manage Access Control on Directory Objects 167
3.2.19 Example 19: Raise the Domain Functional Level 169
3.2.20 Example 20: Replicate Changes within a Domain 171
3.2.21 Example 21: Transferring FSMO roles 173
3.2.22 Example 22: Replicate Changes to a GC or a Partial Replica by Using SMTP 176

3.2.23 Example 23: Cross-Domain Move 177

4 Microsoft Implementations 178
4.1 Product Behavior 178

5 Change Tracking.... 181

6 Index 183

1 Introduction

Active Directory® is a **directory service (DS)**. Directory services can be used to provide a central store for **identity** and **account** information as well as storage of information for other systems and applications. Use of the Active Directory system is appropriate when there is a requirement for a DS. It is also appropriate when building another system that has a dependency on the **Active Directory** protocols. An example of such a system is the Group Policy system, which is described in the Group Policy Protocols Overview document [MS-GPOD].

This document describes the member protocols that comprise the Active Directory system. It also describes the abstract state that is shared between the system's protocols. This document is intended for anyone who plans to implement the Active Directory system because it provides a high-level introduction to the functionality of the system and also describes the protocols that an implementation of the Active Directory system must support.

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

A DS 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 DS. The directory objects are organized in a hierarchical manner with regards to other directory objects. For example, a DS might 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 the user name, password, or telephone number. The following diagram shows this example.

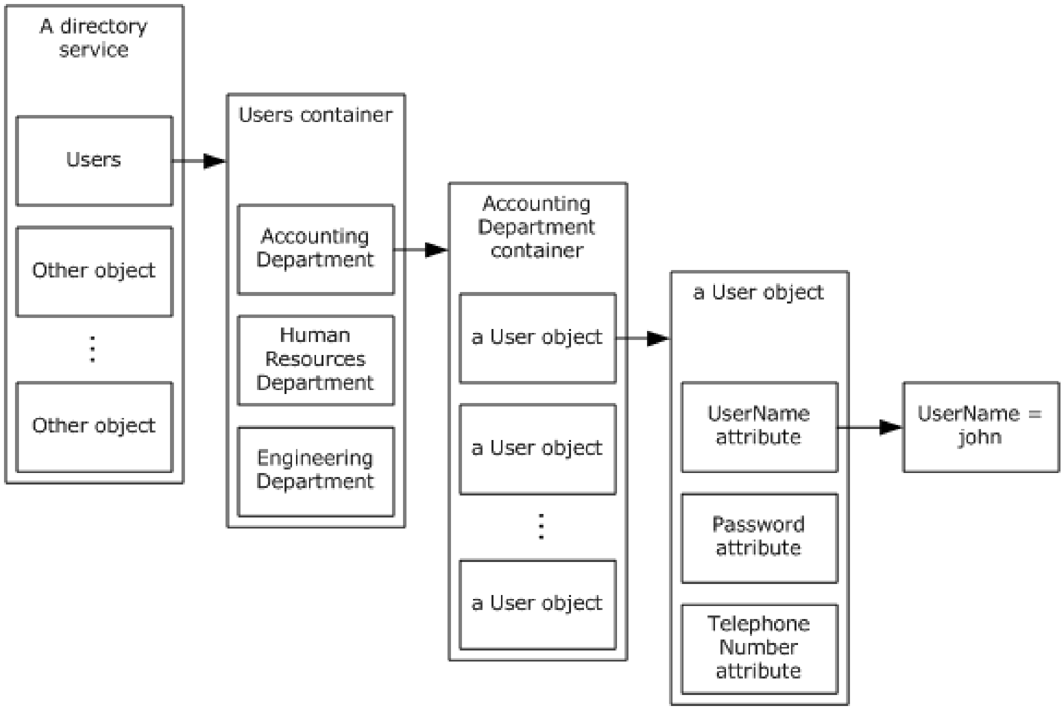

Figure 1: Example of directory organization

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.<1>

AD DS is also accessible via LDAP versions 2 and 3, but it extends the basic DS to include additional capabilities, such as the ability to host **domain naming contexts (domain NCs)**, and additional protocols. This permits AD DS to store the account information for the users of a computer network. The collection of accounts that are 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 of 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 be 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** that run a directory service. In the case of both AD DS and AD LDS, these computers are referred to as **domain controllers (DCs)**. Even though the directory service can run on multiple computers, these computers replicate the contents of the directory so that a **client** sees a consistent view of the directory no matter which directory server or DC it communicates with. The network protocols that perform this replication are described in [MS-DRSR], ([MS-NRPC] section 3.6, and [MS-SRPL].<2>

Note This document, like [MS-ADTS] and [MS-DRSR], uses the term "domain controller" to refer to a DS that runs as either Active Directory Domain Services (AD DS) or Active Directory Lightweight Directory Services (AD LDS). Both AD DS and AD LDS are considered to be directory services.

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 that match client-specified criteria.

The core protocol that clients use to perform operations is LDAP version 3 as described in [MS-ADTS] which is preferred over version 2. Clients can also communicate with the Active Directory system by using the network protocols described in [MS-DRSR], [MS-SRPL], [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 that the Active Directory system supports depends on whether the system is running as AD DS or AD LDS (see section 2.8).

This document provides a description of the client-server functionality of the Active Directory system and additional specific benefits that a client realizes from being associated with an AD DS directory service (that is, "joined to a domain") that are described later in this document. This document does not describe identity and security concepts that are defined as part of the Windows Protocols Overview Document [MS-WPO].

A directory service is a service that stores and organizes directory objects in a centralized, hierarchical data store. 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 specific object. For more information, see [MSDN-DomainTrees].

Domain Interaction

Active Directory Domain Services (AD DS) implements one or more domains within a **forest**. A domain provides a number of services to its clients, primarily related to security and management. The **security principals** of the domain are all available from the AD DS domain controller; therefore, the domain serves as the primary source of identity for the clients of the domain. The domain, through the relevant security protocols, provides the basis for authentication within the domain, allowing **principals** within the domain to establish authenticated connections with each other. During the authentication process, the domain provides authorization information in the form of additional identities that represent groups to enable authorization decisions to be made.

Active Directory Domain Services (AD DS) stores directory data and manages communication between users and domains. This includes user logon processes, authentication, and directory searches. AD DS provides a distributed database that stores and manages information about network resources and application-specific data from directory-enabled applications. Administrators can use AD DS to organize elements of a network, such as users, computers, and other devices, into a hierarchical containment structure. The hierarchical containment structure includes the Active Directory forest, domains in the forest, and organizational units (OUs) in each domain.

The Active Directory system that runs as the AD DS service on an AD DS domain controller provides certain services and benefits to domain-joined clients. 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, and automatic deployment of administrator-configured **policy** settings.

Many network-related operations depend on domains in order to complete various tasks. This document describes some of these tasks, including:

- Locating a domain controller using **DNS** and **NetBIOS**.
- Joining a domain by creating an account via the Security Account Manager **remote procedure call (RPC)** protocol [MS-SAMR].
- Joining a domain by creating an account via LDAP.
- Removing a **domain member**.

This document includes protocols that are used to communicate with a domain controller and maintain state. It also includes protocols that are used to augment authentication and authorization actions, and protocols that are used to interact with domain controllers.

The domain controller serves a central **role** in an enterprise network by functioning as the root of authority for sets of users and computers. A domain controller aggregates functionality that relates to identity management, authentication, authorization, and other management policy. Clients of the domain functionality in turn rely on the domain controller to establish secure communication, authorize requests, and apply policy. A client of the domain may itself be a server of some other role, for example, a file server that is handling the file storage requirements of other client workstations.

Directory Replication

Active Directory is a distributed directory service that stores objects that represent real-world entities, such as users, computers, services, and network resources. Objects in the directory are distributed among all domain controllers in a forest, and all domain controllers can be updated directly. Active Directory replication is the process by which the changes that originate on one domain controller are automatically transferred to other domain controllers that store the same data.

The Active Directory replication model ensures that Active Directory data on one domain controller eventually converge with the **replica** of the same data on other domain controllers in the same domain. The Active Directory replication model determines how changes to Active Directory data are propagated and tracked automatically between domain controllers. The replication model allows directory data on each domain controller to be updated directly. Each domain controller maintains replication metadata that indicates the update status both for itself and relative to other domain controllers. In addition, the replication model allows each domain controller to request (pull) only changes that have to be replicated and to forward changes to other domain controllers that require them.

When a change is made to an object in a directory partition, the value of the changed attribute or attributes must be updated on all domain controllers that store a replica of the same directory partition. Domain controllers communicate data updates automatically through Active Directory replication. Their communication about updates is always specific to a single directory partition at a time.

A **domain** that is run by AD DS can consist of many partitions or **naming contexts (NCs)**. The distinguished name (DN) of an object includes enough information to locate a replica of the partition that holds the object. The **global catalog (GC)** contains a partial replica of every NC in the directory. It also contains the **schema** and **configuration naming contexts (config NCs)**. This means that the GC holds a replica of every object in the directory but with only a small number of their attributes. The attributes in the GC are those that are most frequently used in search operations, such as a user's first and last names or logon names, and those required to locate the full replica of the object.

Active Directory objects are instances of schema-defined classes, which consist of named sets of attributes. Schema definitions determine whether an attribute can be administratively changed. Attributes that cannot be changed are never updated and therefore never replicated. However, most Active Directory objects have attribute values that can be updated.

Replication within a **site** occurs as a response to changes. On its **NTDS** Settings object, the source domain controller stores a repsTo attribute that lists all servers in the same site that pull replication from it. The Knowledge Consistency Checker (KCC) updates these attributes, as described later in this section.

When a change occurs on a source domain controller, it notifies its destination replication partner, prompting the destination domain controller to request the changes from the source domain controller. The source domain controller either responds to the change request with a replication operation or places the request in a queue if requests are already pending. Each domain controller has a queue of pending replication operations that are processed one at a time.

Directory Replication Service (DRS) Remote Protocol is a remote procedure call (RPC) protocol for replication and management of data in Active Directory [MS-DRSR]. Domain controllers (DCs) use the Security Account Manager (SAM) Remote Protocol (Server-to-Server) [MS-SAMS] to forward time-critical database changes to the **primary domain controller (PDC)**, and to forward time-critical database changes from a **read-only domain controller (RODC)** to a **writable NC replica** within the same domain outside the normal replication protocol. This protocol is used only between Active Directory servers in the same domain. Beginning with Windows Server 2008 operating system, this protocol was extended to forward certain nontime-critical write operations from an RODC to a writable NC replica. The SAMS protocol addresses the requirement to propagate a subset of database changes to the PDC more quickly than the Directory Replication Service (DRS) Remote Protocol, as specified in [MS-DRSR]. This rapid propagation is used for sensitive information when the delay imposed by standard Active Directory replication creates an unwelcome burden on the user or creates a risk to the enterprise. An example of the former is a password change operation; if the password is not made available rapidly, a user can experience unpredictable authentication failures when the new password is tried against domain controllers that have not yet replicated it. An example of the latter is when an account is locked out due to multiple password failures; the lockout condition, and, equally important, the lockout-cleared condition, must be propagated rapidly throughout the domain.

Replication can be event-driven or schedule-driven as explained in [MS-ADTS] section 3.1.1.1.14.

Knowledge Consistency Checker (KCC)

A domain controller (DC) that runs Active Directory is part of a distributed system that performs replication. The Knowledge Consistency Checker (KCC) is a component that reduces the administrative burden of maintaining a functioning replication topology. The KCC ensures that a replication path exists between the same NCs that are present in different DCs. The KCC is explained in [MS-ADTS] sections 3.1.1.1.13 and 6.2. The KCC helps administrators build a replication topology that incurs minimal cost. This cost is defined by the administrator as explained in [MS-ADTS] section 3.1.1.1.13.

FSMO Roles

Each DC accepts **originating updates** for most attributes of most objects within its writable NC replicas. However, certain updates are accepted only if the DC is the single designated "master" DC for the update. This mechanism is called **flexible single master operation (FSMO)**.

If some or all of the updates to an object are single-mastered, that object belongs to a defined set of objects. [MS-DRSR] section 4.1.10.5.3 (GetReplScope) specifies these sets, which are called **FSMO roles**. Each FSMO role is applicable to a certain scope: either domain-wide, or forest-wide. The domain-wide FSMO roles include the *Infrastructure Master FSMO*, the *Rid Master FSMO*, and the *PDC Emulator FSMO*. The forest-wide FSMO roles include the *Domain Naming FSMO* and the *Schema Master FSMO*. There are no FSMO roles that apply strictly to **application NCs**.

Because a server that is operating as AD LDS does not host domain NCs, it cannot own any of the three domain-specific FSMO roles. It can own the Schema Master FSMO and Domain Naming FSMO roles.

In a given NC, each FSMO role is represented by an object. [MS-DRSR] section 4.1.10.5.3 (GetReplScope) specifies these objects, which are called **FSMO role objects**.

The fSMORoleOwner attribute of each FSMO role object is an object reference to the nTDSDSA object of the DC that owns the role; that is, the DC that performs updates to objects in the role. Information about nTDSDSA objects and how they represent DCs are specified in [MS-ADTS] section 6.1.

An originating update to an object within a FSMO role generates an LDAP referral if the DC that receives the request cannot perform the update; the referral is to the DC represented by the nTDSDSA object referenced by the FSMO role object's fSMORoleOwner attribute on the DC that received the request.

The processing of updates affected by FSMO roles is fully specified in [MS-ADTS] section 3.1.1.5.

The IDL_DRSGetNCChanges method ([MS-DRSR] section 4.1.10) makes an originating update to the fSMORoleOwner attribute of a FSMO role object while preserving single-mastering of updates to the FSMO role. The ability to update the fSMORoleOwner attribute in this way is exposed through LDAP as the root DSE updates becomeDomainMaster, becomeInfrastructureMaster, becomePdc, becomePdcWithCheckPoint, becomeRidMaster, and becomeSchemaMaster, as specified in [MS-ADTS] section 3.1.1.3.

Reading the rootDSE attribute validFSMOs on a DC returns the set of all FSMO roles (represented as FSMO role objects) that the DC will update, as specified in [MS-ADTS] section 3.1.1.3.

Active Directory Trust Management

Active Directory domains rarely exist in isolation. Many Active Directory deployments in customer sites consist of two or more domains that represent boundaries between different geographical, managerial, organizational, or administrative layouts. For example, when company "A" acquires company "B", it quickly becomes necessary for preexisting domains to start trusting each other. Alternatively, in some deployments, servers that have a specific role (such as a mail server) may be members of a "resource domain", easing the management burden by combining like roles under one administrative domain.

Communication between disparate domains, especially secure communication that involves authentication and authorization, requires that some stateful knowledge is shared between the peer domains for them to trust one another. Some of this knowledge is sensitive, forming the cryptographic basis of trust mechanisms used in protocols such as Kerberos and **Netlogon** RPC. Other state is public knowledge, such as the NetBIOS name of a peer domain, or which security identifiers are owned by the peer domain. Information like this plays a crucial role to perform name lookups, which are essential for authorization, to locate user accounts, or simply to display information in some type of user interface.

Active Directory stores trust information in **trusted domain objects (TDOs)** ([MS-ADTS] section 6.1.6.2) and, depending on the kind of trust established, in associated user accounts, **interdomain**

trust accounts, for the **trusted domain**. There are different types of trusts that exist between Active Directory domains, as described in [MS-ADTS] section 6.1.6.2. TDOs can be managed through the LSAD protocol ([MS-LSAD] section 3.1.4.7).

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)
binary large object (BLOB)
binding
certificate
credential
directory
directory object
directory service (DS)
distinguished name (DN)
domain
domain account
domain controller (DC)
domain member (member machine)
Domain Name System (DNS)
domain naming context (domain NC)
domain object
endpoint
fully qualified domain name (FQDN) (1)(2)
global catalog (GC)
group
Interface Definition Language (IDL)
Kerberos
Key Distribution Center (KDC)
KRB_PRIV exchange
Lightweight Directory Access Protocol (LDAP)
mailslot
mutual authentication
naming context (NC)
NetBIOS
Netlogon
Network Data Representation (NDR)
NTDS
object class
originating update
policy
primary domain controller (PDC)
read-only domain controller (RODC)
relative distinguished name (RDN)
relative identifier (RID)
remote procedure call (RPC)
role

SASL
schema
Secure Sockets Layer (SSL)
security descriptor
security identifier (SID)
security principal
Server Message Block (SMB)
service principal name (SPN)
service (SRV) resource record
service ticket
site
SOAP
ticket-granting service (TGS)
ticket-granting ticket (TGT)
tombstone
Transmission Control Protocol (TCP)
Transport Layer Security (TLS)
trusted domain
trusted domain object (TDO)
User Datagram Protocol (UDP)
user principal name (UPN)
XML

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

Active Directory Lightweight Directory Services (AD LDS)
application NC
attribute
configuration naming context (config NC)
deleted-object
domain functional level
flexible single master operation (FSMO)
forest
FSMO role
FSMO role object
interdomain trust account
replica
schema NC
writable NC replica

The following terms are defined in [WSENUM]:

enumeration context

The following terms are specific to this document:

account: A synonym for security principal or principal.

account database: The portion of the directory that maintains the accounts for the principals of the domain. In Windows NT 4.0-style domains, the account database includes all information in the domain; in Active Directory-style domains, the account database contains a subset of the entire LDAP-accessible directory that the Active Directory-style domain hosts.

Active Directory-style domain: A domain that is created from Windows 2000 Server, Windows Server 2003, Windows Server 2003 R2, Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, or Windows Server 2012 R2 server computers. Active Directory-style domains implement Active Directory, LDAP, Kerberos authentication, and advanced configurations and features that are not supported in Windows NT 4.0-style domains.

client: A synonym for client computer.

client computer: A computer that is not a domain controller server. The computer may or may not be joined to a domain.

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**.

domain client: A client computer that is joined to a domain. The domain client can be a client or a server that offers other services to its clients. When the domain client acts as a supplicant to another domain client, the supplicant is referred to as a domain client in a workstation role, and the other domain client is referred to as a domain client in a server role.

domain client in a workstation role: A domain member that offers other services to other domain clients.

domain controller server: A domain member, which can be a client or a server, that offers other services to its clients. When the domain client acts as a supplicant to another domain client, the supplicant is referred to as a domain client in a workstation role, and the other domain client is referred to as a domain client in a server role.

extended control: A mechanism that specifies 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 a "control".

identity: An account that represents a person (user account), an application (service account), and computers that participate in the domain (machine accounts). The system uses a password as proof of an identity.

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

member server: A server that is joined to a domain, but is not a domain controller. Member servers typically function as file servers, application servers, and so on, and defer user authentication to the domain controller.

principal: A synonym of security principal.

replica domain controller / replica directory server: A server that contains a replicated copy of the directory and is able to answer client requests over any protocol that the directory service supports.

server: A domain controller. Used as a synonym for domain controller in this document.

security principal: An entity that is associated with a human user or a program that can be authenticated. At a minimum, it has two basic attributes, a name and an identifier, that uniquely identifies it and makes it meaningful to the system, administrators, and users. A security principal is also known as a principal or an account.

Windows NT 4.0-style domain: A domain that is comprised of Windows NT 4.0 servers with an account database that includes all the information in the domain. Windows NT 4.0-style domains do not implement Active Directory, LDAP directories, or Kerberos authentication.

WS-Addressing: The WS-Addressing protocol, as specified in [WSA].

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 specified in [MS-ADCAP].

DRSR: The Directory Replication Service (DRS) Remote Protocol, as specified in [MS-DRSR].

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

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

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

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

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

NRPC: The Netlogon Remote Protocol, as specified in [MS-NRPC].

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

SAMS: The server-to-server Security Account Manager (SAM) Remote Protocol, as specified in [MS-SAMS].

SRPL: The Directory Replication Service (DRS) Protocol Extensions for SMTP, as specified in [MS-SRPL].

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

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

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

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

1.2 References

References to Microsoft Open Specification documents 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.

[MSDN-DomainTrees] Microsoft Corporation, "Domain Trees", http://msdn.microsoft.com/en-us/library/windows/desktop/ms675914(v=vs.85).aspx

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

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

[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-APDS] Microsoft Corporation, "Authentication Protocol Domain Support".

[MS-AUTHSOD] Microsoft Corporation, "Authentication Services Protocols Overview".

[MS-CIFS] Microsoft Corporation, "Common Internet File System (CIFS) Protocol".

[MS-DFSC] Microsoft Corporation, "Distributed File System (DFS): Referral Protocol".

[MS-DPSP] Microsoft Corporation, "Digest Protocol Extensions".

[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-FASOD] Microsoft Corporation, "File Access Services Protocols Overview".

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

[MS-GPOD] Microsoft Corporation, "Group Policy Protocols Overview".

[MS-KILE] Microsoft Corporation, "Kerberos Protocol Extensions".

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

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

[MS-MAIL] Microsoft Corporation, "Remote Mailslot Protocol".

[MS-NAPOD] Microsoft Corporation, "Network Access Protection Protocols Overview".

[MS-NLMP] Microsoft Corporation, "NT LAN Manager (NTLM) Authentication Protocol".

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

[MS-PAC] Microsoft Corporation, "Privilege Attribute Certificate Data Structure".

[MS-RCMP] Microsoft Corporation, "Remote Certificate Mapping Protocol".

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

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

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

[MS-SMB] Microsoft Corporation, "Server Message Block (SMB) Protocol".

[MS-SMB2] Microsoft Corporation, "Server Message Block (SMB) Protocol Versions 2 and 3".

[MS-SNTP] Microsoft Corporation, "Network Time Protocol (NTP) Authentication Extensions".

[MS-SRPL] Microsoft Corporation, "Directory Replication Service (DRS) Protocol Extensions for SMTP".

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

[MS-WPO] Microsoft Corporation, "Windows Protocols Overview".

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

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

[MSDN-SUBSYSDSGN] Microsoft Corporation, "Trusted Subsystem Design", http://msdn.microsoft.com/en-us/library/aa905320.aspx

[RFC1001] Network Working Group, "Protocol Standard for a NetBIOS Service on a TCP/UDP Transport: Concepts and Methods", STD 19, RFC 1001, March 1987, http://www.ietf.org/rfc/rfc1001.txt

[RFC1002] Network Working Group, "Protocol Standard for a NetBIOS Service on a TCP/UDP Transport: Detailed Specifications", STD 19, RFC 1002, March 1987, http://www.ietf.org/rfc/rfc1002.txt

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

[RFC1035] Mockapetris, P., "Domain Names - Implementation and Specification", STD 13, RFC 1035, November 1987, http://www.ietf.org/rfc/rfc1035.txt

[RFC1305] Mills, D. L., "Network Time Protocol (Version 3) Specification, Implementation and Analysis", RFC 1305, March 1992, http://www.ietf.org/rfc/rfc1305.pdf

[RFC1769] Mills, D., "Simple Network Time Protocol (SNTP)", RFC 1769, March 1995, http://www.ietf.org/rfc/rfc1769.txt

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

[RFC2030] Mills, D., "Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI", RFC 2030, October 1996, http://www.rfc-editor.org/rfc/rfc2030.txt

[RFC2052] Gulbrandsen, A., and Vixie, P., "A DNS RR for specifying the location of services (DNS SRV)", RFC 2052, October 1996, http://www.ietf.org/rfc/rfc2052.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

[RFC2247] Kille, S., Wahl, M., Grimstad, A., et al., "Using Domains in LDAP/X.500 Distinguished Names", RFC 2247, January 1998, http://www.ietf.org/rfc/rfc2247.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

[RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., et al., "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999, http://www.ietf.org/rfc/rfc2617.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

[RFC2821] Klensin, J., "Simple Mail Transfer Protocol", STD 10, RFC 2821, April 2001, http://www.ietf.org/rfc/rfc2821.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

[RFC3596] Thomson, S., Huitema, C., Ksinant, V., and Souissi, M., "DNS Extensions to Support IP version 6", RFC 3596, October 2003, http://www.ietf.org/rfc/rfc3596.txt

[RFC3629] Yergeau, F., "UTF-8, A Transformation Format of ISO 10646", STD 63, RFC 3629, November 2003, http://www.ietf.org/rfc/rfc3629.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

[RFC3961] Raeburn, K., "Encryption and Checksum Specifications for Kerberos 5", RFC 3961, February 2005, http://www.ietf.org/rfc/rfc3961.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

[RFC4556] Zhu, L., and Tung, B., "Public Key Cryptography for Initial Authentication in Kerberos", RFC 4556, June 2006 http://www.ietf.org/rfc/rfc4556.txt

[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/

[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/

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

[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/

2 Functional Overview

The Active Directory protocols provide a centralized directory service with the ability to integrate into the Windows domain security model. They 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 other Active Directory Services protocols, 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 domain integration capabilities are required. When deploying the Active Directory System to provide these capabilities, the AD DS mode of operation is used.
- For use in scenarios where other systems that have a dependency on the Active Directory System, such as Group Policy or Message Queuing, 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 domain integration capabilities are not required, the AD LDS mode of operation can be a particularly good choice because it does not require support for the protocols such as SAMR, LSAD, and LSAT that are not used by the client application in these scenarios.
- For replication of objects. Active Directory is a distributed directory service that stores objects that represent real-world entities such as users, computers, services, and network resources. Objects in the directory are distributed among all domain controllers in a forest. Directory replication protocols DRSR, SRPL, and SAMS are used to replicate directory objects between different domain controllers.

The Active Directory protocols 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.

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

2.1 Components and Capabilities

From an abstract point of view, the functionality provided by the Active Directory protocols can be represented by the components shown in the following diagram.

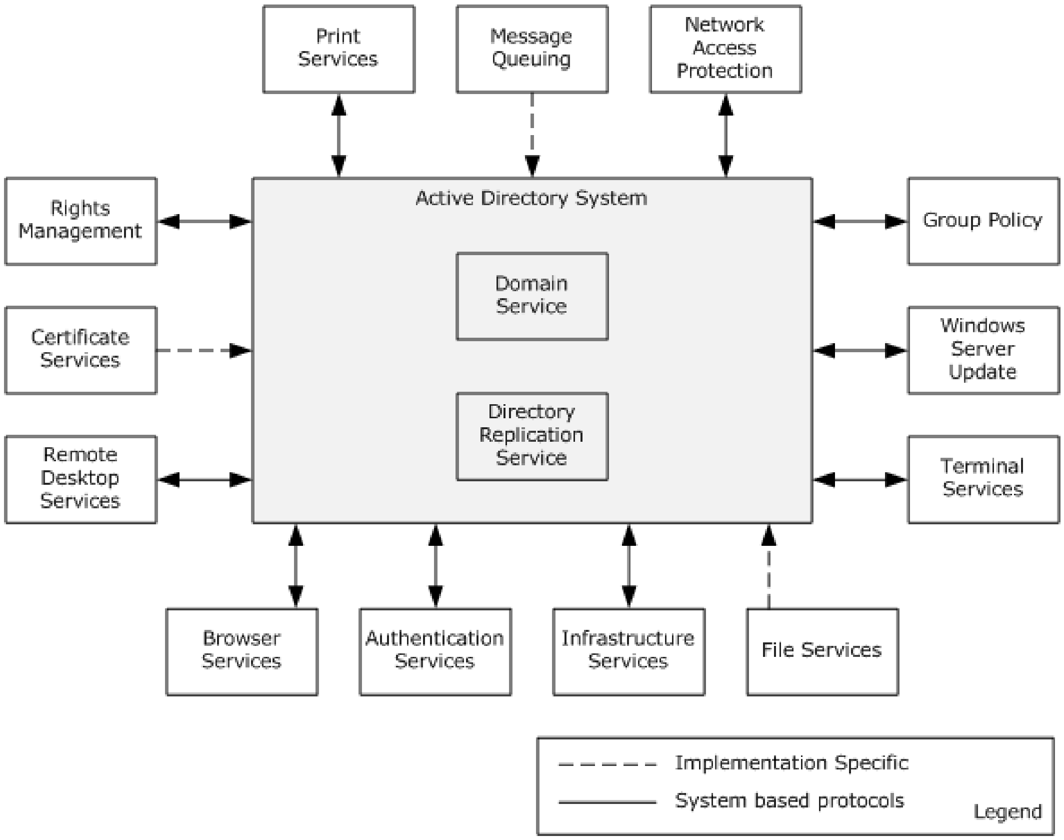

Figure 2: System Component Diagram

2.2 Relevant Standards

The following standards are relevant to the Active Directory System.

Lightweight Directory Access Protocol (LDAP), as specified in [RFC2247]: 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.

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.

This document uses and extends the following standards:

Encryption and Checksum Specifications for Kerberos 5, as specified in [RFC3961]. This standard is used for encryption and checksum mechanisms.

Kerberos Authentication Protocol, as specified in [RFC4120]. This standard is used for authentication.

Public Key Cryptography for Initial Authentication in Kerberos (PKINIT) , as specified in [RFC4556]. This standard is used for initial authentication in Kerberos.

Simple Network Time Protocol (SNTP), as specified in [RFC1769]. This standard is an adaptation of the Network Time Protocol (NTP) used to synchronize computer clocks in the Internet. SNTP can be used when the ultimate performance of the full NTP implementation described in [RFC1305] is not needed or justified.

Domain Names - Concepts and Facilities, as specified in [RFC1034]. This standard is used for DNS and domain naming concepts.

Domain Names - Implementation and Specification, as specified in [RFC1035]. This standard is used for DNS and provides details of the domain system and protocol.

DNS Extensions to Support IP Version 6, as specified in [RFC3596]. This standard is used for DNS to support hosts running IP version 6 (IPv6).

UTF-8, A Transformation Format of ISO 10646, as specified in [RFC3629]. This standard is used for string data type specification.

Protocol Standard for a NetBIOS Service on a TCP/UDP Transport: Concepts and Methods, as specified in [RFC1001]. This standard is used for NetBIOS services.

Protocol Standard for a NetBIOS Service on a TCP/UDP Transport: Detailed Specifications, as specified in [RFC1002]. This standard is used for NetBIOS services.

2.3 Protocol Relationships

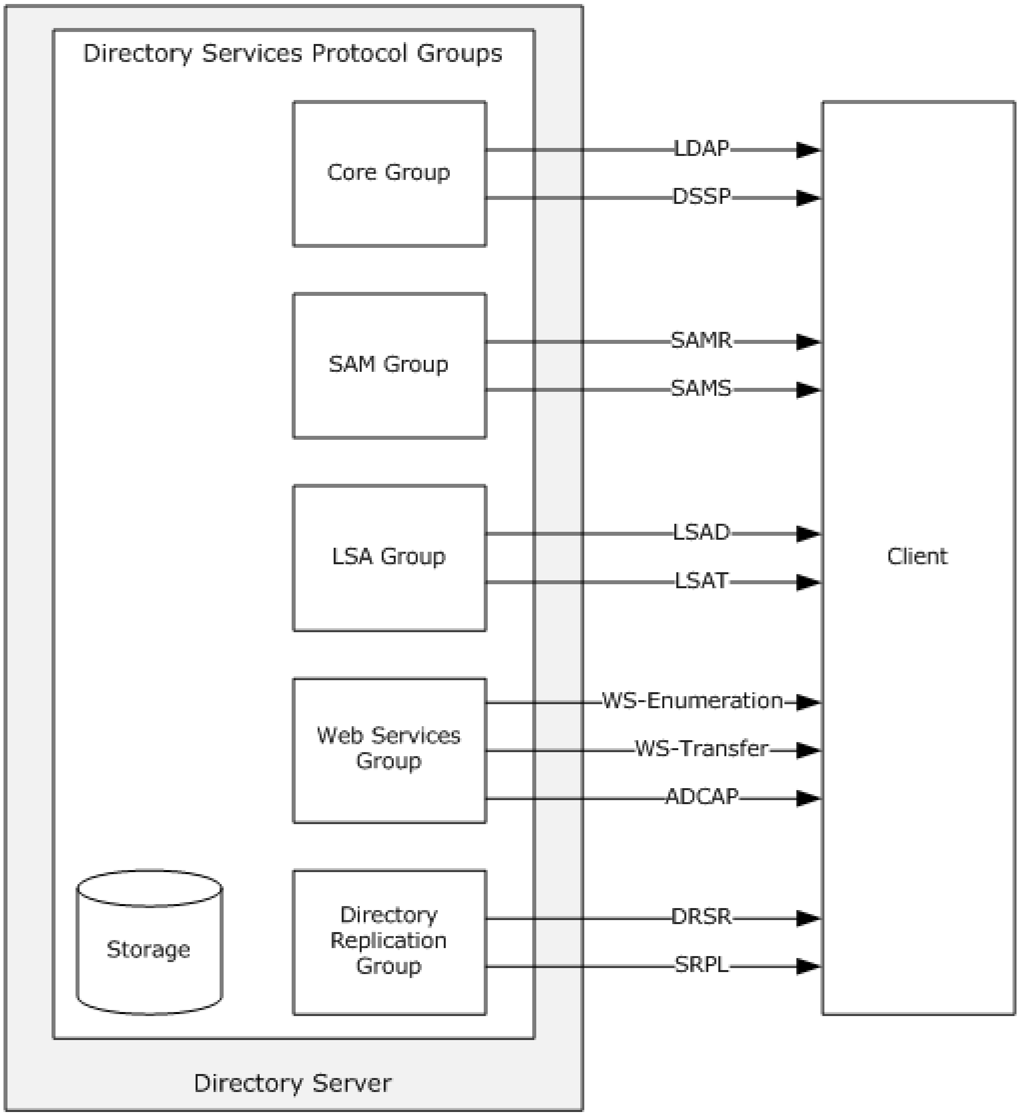

Figure 3: Active Directory Protocol grouping

As illustrated in the preceding figure, the member protocols that make up the Active Directory Services Protocol Groups can be divided into five 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. 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 Services Protocols.

- The SAM group includes SAMR and SAMS. SAMR is used to perform account maintenance and operates on the same directory tree as the core group of protocols, but it provides access to only a subset of the objects in that tree, and, further, provides access to only a subset of the attributes on those objects. It is supported only when operating in AD DS mode. SAMS is used to perform account maintenance and time-critical database changes between Active Directory servers that are in the same domain.
- The LSA group contains the LSAD and LSAT protocols. Both protocols are serviced by the same RPC interface and **endpoint** (see [MS-LSAD] section 1.8, Vendor-Extensible Fields, and [MS-LSAT] section 1.8, Vendor-Extensible Fields).
- The Web Services group consists of the WS-Transfer, WS-Enumeration, and ADCAP protocols along with the WSDS, and WSPELD protocol extensions. This protocol group is only supported on some versions of the Active Directory Services. 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.
- The Directory Replication group contains the DRSR and SRPL protocols. DRSR is an RPC protocol that is used for management of replication and management of data in Active Directory. SRPL is the extension to the DRS protocol for transport over the Simple Mail Transfer Protocol (SMTP).

The following diagram depicts the relationship among the protocols of the Active Directory System.

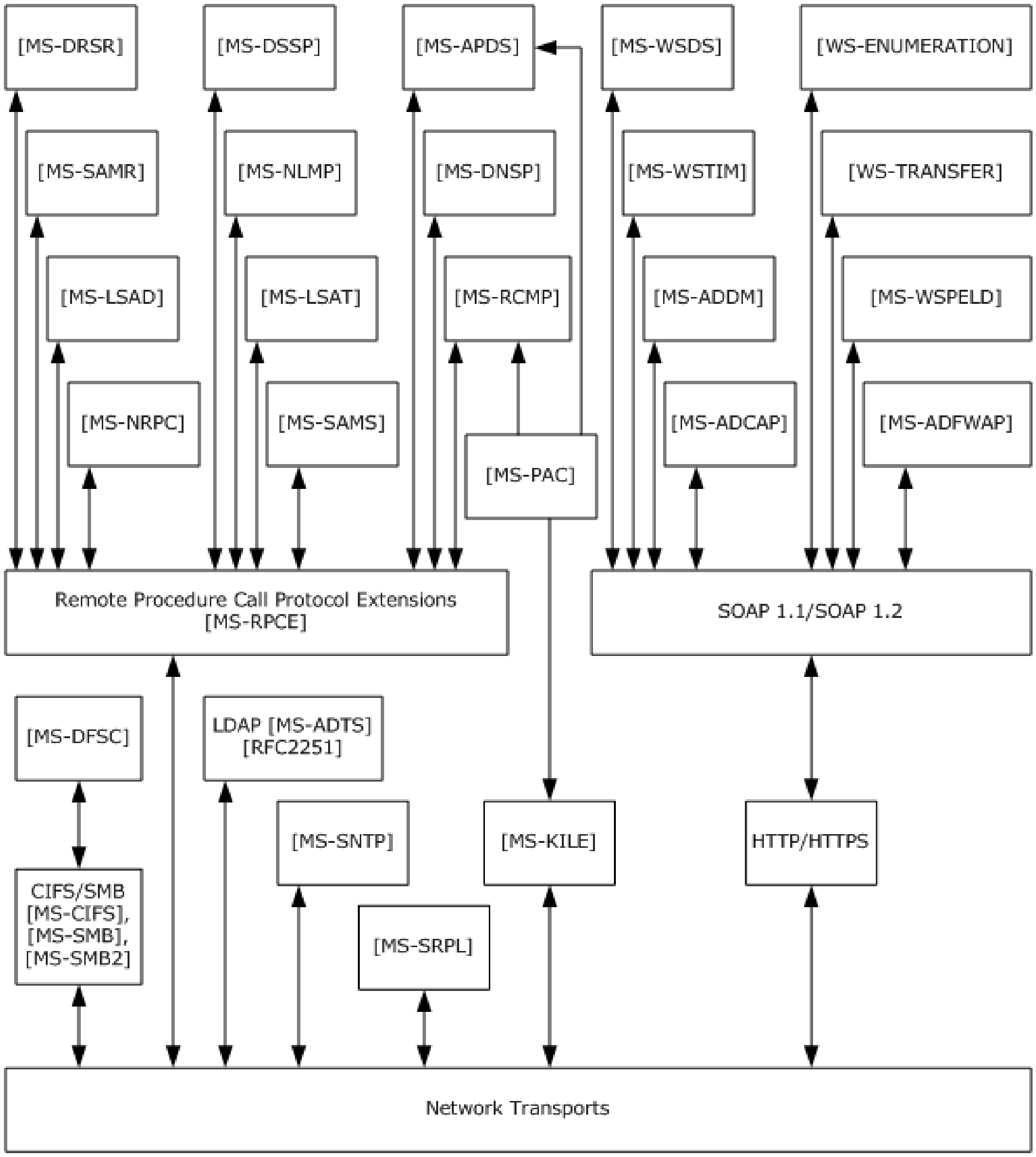

Figure 4: Protocol relationships

2.4 Protocol Summary

The following tables provide a comprehensive list of the member protocols of the Active Directory system. Section 2.8 provides details about which protocols or protocol subsets are supported in the different modes of operation.

The protocols in the following table enable the core functionality of the Active Directory system, including access to the directory tree, replication, name translation, determination of group membership, and domain controller status. These protocols are supported by all directory servers in the Active Directory System, whether running in Active Directory Domain Services (AD DS) or Active Directory Lightweight Directory Services (AD LDS) mode.

Protocol name	Description	Short name
Active Directory extensions for Lightweight Directory Access Protocol (LDAP), versions 2 and 3	Active Directory is a server for LDAP. [MS-ADTS] section 3.1.1.3 specifies the extensions and variations of LDAP that are supported by Active Directory. **Note** In a reference to LDAP without a version number, LDAP refers to both versions 2 and 3.	[MS-ADTS] section 3.1.1.3.1
Directory Replication Service Remote Protocol (drsuapi) - Replication	The Directory Replication Service (DRS) Remote Protocol. This protocol includes the drsuapi and dsaop remote procedure call (RPC) interfaces. Methods on these interfaces provide replication of directory information among the domain controllers of an AD DS domain. Methods on these interfaces also provide a variety of functionality to clients, such as converting names between formats and retrieving information about AD DS domain controllers. This protocol also supports DC cloning operations.<3>	[MS-DRSR]
SMTP Replication Protocol Extensions	The Directory Replication Service (DRS) Protocol Extensions for SMTP. This protocol provides Simple Mail Transfer Protocol (SMTP) transport of replication information as an alternative to RPC.	[MS-SRPL]
Directory Services Setup Remote Protocol	The Directory Services Setup Remote Protocol, as defined in [MS-DSSP]. This protocol can be used to retrieve information about the state of a computer in a domain or a non-domain workgroup.	[MS-DSSP]

The protocols in the following table enable account maintenance when the Active Directory system is operating in AD DS mode. This includes the creation, modification, retrieval, and deletion of users and groups.

Protocol name	Description	Short name
Security Account Manager (SAM) Remote Protocol (Client-to-Server)	The Security Account Manager (SAM) Remote Protocol. Clients can use this protocol to perform account maintenance, for example, to create and delete accounts. The capabilities of this protocol are a subset of the capabilities of the LDAP protocol.	[MS-SAMR]
Security Account Manager (SAM) Remote Protocol (Server-to-Server)	The Security Account Manager (SAM) Remote Protocol. Domain controllers (DCs) use this protocol to forward time-critical database changes to the primary domain controller (PDC), and to forward time-critical database changes from a read-only domain controller (RODC) to a writable NC replica within the same domain outside the normal replication protocol. This protocol is used only between Active Directory servers in the same domain.	[MS-SAMS]

The protocols in the following table allow clients to retrieve security policy information and translate **security identifiers (SIDs)** that identity **security principals**, such as users, to human-readable names.

Protocol name	Description	Short name
Local Security Authority	The Local Security Authority (Domain Policy) Remote Protocol.	[MS-

Protocol name	Description	Short name
(Domain Policy) Remote Protocol	Clients can use this protocol to retrieve security policy information.	LSAD]
Local Security Authority (Translation Methods) Remote Protocol	The Local Security Authority (Translation Methods) Remote Protocol. Clients can use this protocol to translate security identifiers (SIDs) of security principals to human-readable names, and vice versa.	[MS-LSAT]

The protocols in the following table enable Web services for the Active Directory system that grant access to the directory tree and the management of Active Directory account information and topologies.

Protocol name	Description	Short name
Active Directory Web Services Custom Action Protocol	The Active Directory Web Services Custom Action Protocol. It is a SOAP-based Web Services protocol for managing account and topology information.	[MS-ADCAP]
WS-Transfer: Identity Management Operations for Directory Access Extensions	The WS-Transfer: Identity Management Operations for Directory Access Protocol Extensions. 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.	[MS-WSTIM]
WS-Enumeration	The WS-Enumeration protocol. This protocol allows directory objects to be queried by using a SOAP-based Web Services protocol.	[WSENUM]
WS-Transfer	The WS-Transfer protocol. This protocol allows directory objects to be created, removed, modified, and read by using a SOAP-based Web Services protocol.	[WXFR]
WS-Enumeration: Directory Services Protocol Extensions	The WS-Enumeration Directory Services Protocol Extensions. This is a set of protocol extensions to WS-Enumeration that, among other things, allows a client to request that 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.	[MS-WSDS]
WS-Transfer Lightweight Directory Access Protocol (LDAP) v3 Control Extension	WS-Transfer Lightweight Directory Access Protocol (LDAP) v3 Control Extension. 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.	[MS-WSPELD]
Active Directory Web Services: Data Model and Common Elements	The Active Directory Web Services: Data Model and Common Elements. Although not a protocol itself, this defines an XML data model that is shared by the other Web Service protocols and protocol extensions, and common protocol elements referenced by the other documents.	[MS-ADDM]

2.5 Environment

Because domain interactions are distributed among many computers for different, but related, purposes, enumerating the dependencies of the Active Directory protocols is complex. The rough diagram in the following figure serves as a very high-level depiction of how the dependencies among components can be visualized.

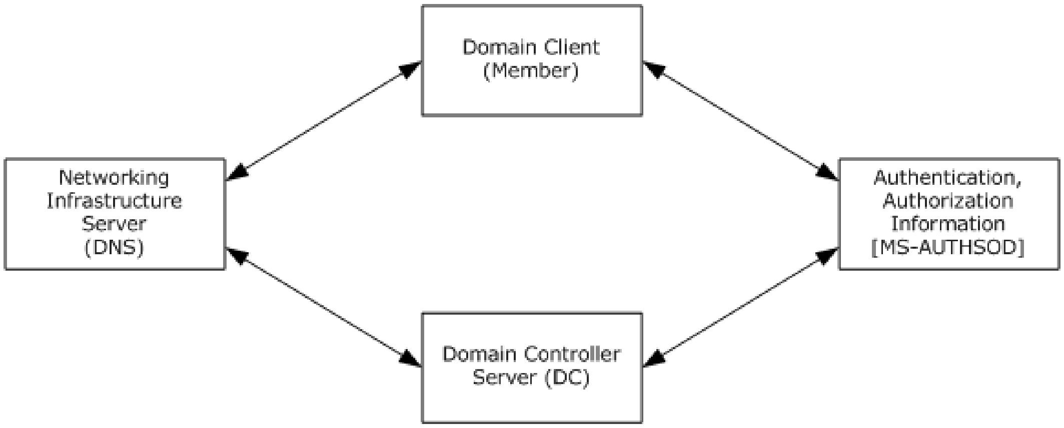

Figure 5: Dependencies among domain components

In this diagram, the dependencies of the system are symmetrical between the **domain client** and the **domain controller server**. Both the domain client and the domain controller server rely upon infrastructure servers such as DNS, and leverage those servers for locating each other (rendezvous). During this rendezvous process, the domain controller server publishes its name and the domain client locates the domain controller server through DNS. The details of this rendezvous process are described in section 2.7.7.3.1.

In addition to service location, the rendezvous process between a domain client and a domain controller server relies upon authentication and authorization information. The domain controller server, for example, leverages the authorization information that it contains for controlling access to its resources. A more complete description of this authentication and authorization information can be found in [MS-AUTHSOD] and related documents.

2.5.1 Active Directory Protocols Dependencies

This section describes the dependencies that the Active Directory protocols have on other entities. The Active Directory protocols require 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 protocols also require a networking system that clients can use to send requests to the directory server and to receive responses. This networking system must support the **Transmission Control Protocol (TCP)**, **User Datagram Protocol (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 **service (SRV) resource records** [RFC2782], CNAME and A resource records [RFC1034], and AAAA resource records [RFC3596]. It is a best practice that an implementation of the Active Directory system uses dynamic updates [RFC2136] and [RFC3645] to update DNS with the records, as described in [MS-ADTS] section 6.3.2, DNS Record Registrations, but any alternative method that creates the DNS records described there is permissible.<4> DNS can be used by clients of the Active Directory System in order to locate directory servers by using the algorithms described in [MS-ADTS] section 6.3, Publishing and Locating a Domain Controller.

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

A system of domain interactions forms the framework that other systems leverage in their environments. As such, this system requires comparatively little in terms of services available for use, because its purpose is to create a useful environment for other scenarios. Services that the Active Directory protocols require from their environment include the following:

- **Network Infrastructure.** This system of domain interactions requires that a viable network system is available. This includes a networked environment that supports TCP/IP and UDP/IP. Additionally, a name resolution system must be available for use by both the domain controller server and domain members. The name resolution system must support Domain Name System (DNS) form if the domain is to support **Active Directory-style domain** functionality, and NetBIOS form if the domain is to support **Windows NT 4.0-style domain** functionality.
- **Coexistence.** Any given domain on a network must be uniquely named. There is no architectural limit to the number of domains that are possible on a network.

Even at this relatively high level, the system of domain interactions is a complex aggregation.

2.5.2 Dependencies on Active Directory Protocols

This section lists entities that depend on the interfaces provided by the Active Directory protocols, as well as other entities that the Active Directory protocols depend on. Other entities take dependencies on the AD protocols, and in particular on AD DS, not only by making use of the AD protocols' external interfaces, but also by sharing state with this system; that is, there are overlapping abstract data models.

The Active Directory protocols depend on the Windows Authentication Services [MS-AUTHSOD] to authenticate clients that are accessing the system. The system controls access based on the identity of the client.

The following protocol groups 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, which is vital to the functioning of the Active Directory System in the AD DS mode of operation.

The Active Directory protocols influence the behavior of the following components:

Print Services: The Print Services service 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.

Message Queuing: The Message Queuing service makes use of the directory service to store information such as queue and system metadata.

Network Access Protection: The Network Access Protection (NAP) service [MS-NAPOD] determines how machines can be examined for access to a network. The machines need to be members of a domain in order to authenticate to the NAP servers.

Group Policy: The Group Policy service [MS-GPOD] defines how domain clients can retrieve group policy information from the domain controller, which is based on the group memberships of a **domain account** as well as a domain account's location in the LDAP directory structure.

Windows Server Update: This component specifies how different machines in a domain can have different update policies for patch management, which relies on domain interactions to specify the domain authorization information.

File Services: This component specifies how file servers present a unified view of files and other resources, and rely upon [MS-AUTHSOD] and domain interactions for authentication when the file server is part of a domain.

Infrastructure Service: The Infrastructure Service includes services such as name resolution (DNS, WINS) and network maintenance services (routers). The Directory Service uses such services to make Domain Controllers available to their clients.

Authentication System: The Authentication System [MS-AUTHSOD] defines how other protocols take advantage of authentication protocols such as NTLM and Kerberos to secure their communications, and also defines the authentication services that support the client-to-server communication. The Authentication System depends on domain interactions for specifying how those protocols are used in a domain context to authenticate clients to servers when both are members of a domain.

Several protocol groups leverage the domain controller as the source of identity and authorization information for the domain. These include:

- **Browser Services**, which define how the browser service leverages the directory service to browse and locate the shared resources in the domain based on information associated with the accounts in the domain.
- **Certificate Services**, which specifies how the certificate authority leverages the domain infrastructure to manage certificate distribution and enrollment, and makes authorization decisions based on information associated with the accounts in the domain.
- **Rights Management**, which determines how content can be protected against offline access based on authorization information from the directory.

2.6 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 (that is, "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. 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 and 6.3.4, 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 6.3.8.
- When operating as AD DS, after the server has initialized the protocols listed in section 2.4 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 and 6.3.5, respectively.
- For AD DS, member clients assume basic network connectivity and the availability of basic network infrastructure services such as DNS. Prior to being associated with a domain, there are no other preconditions of note for member clients. Once a client has been associated with a domain, it is under the assumption that the domain controller also has an entry in its directory

corresponding to the client. Should this assumption be proven wrong, the system (from the client's perspective) becomes unusable until the association is reestablished.

- 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 described in section 2.5, 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 the technical documents of the Active Directory System's member protocols 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.
- 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.

Upon startup, the Active Directory System should initialize all of the protocols listed in section 2.4 as described in the protocol documents for each listed protocol and should also begin servicing requests coming in on those protocols' interfaces. There is no requirement that the protocols be initialized in a particular sequence.

Since member clients treat all domain controller instances as equivalent, each domain controller operating as AD DS needs to ensure that it is synchronized with its peer AD DS domain controllers, if any are supported in the implementation, through implementation-specific means such as directory replication.

2.7 Use Cases

The following use cases span the functionality of the Active Directory System.

Use case category	Use cases
Object Management	Create Directory Object - Client Application (section 2.7.1.1) Search for Directory Object - Client Application (section 2.7.1.2) Modify Directory Object - Client Application (section 2.7.1.3) Delete Directory Object - Client Application (section 2.7.1.4) Create Organizational Unit - Client Application (section 2.7.1.5) Cross-Domain Move - Client Application (section 2.7.1.6)
Identity Lifecycle Management	Create a New Account - Client Application (section 2.7.2.1) Reset an Existing Account's Password - Client Application (section 2.7.2.2) Change an Existing Account's Password (PDC) - Client Application (section 2.7.2.3) Change an Existing Account's Password (DC) - Client Application (section

Use case category	Use cases
	2.7.2.4) Change User Account Password Against an RODC - Client Application (section 2.7.2.5) User Login to Domain Services using an RODC and Updating the User LastLogonTimeStamp - Client Application (section 2.7.2.6) Query an Account's Group Membership - Client Application (section 2.7.2.7) Delete an Account - Client Application (section 2.7.2.8) Create a Security Group - Client Application (section 2.7.2.9) Modify Group Member List - Client Application (section 2.7.2.10) Query Members of a Group - Client Application (section 2.7.2.11)
Schema Management	Add a New Class to the Schema - Client Application (section 2.7.3.1) Add a New Attribute to the Schema - Client Application (section 2.7.3.2) Add an Attribute to a Class - Client Application (section 2.7.3.3)
Name Translation	Convert a SID to/from a Human-Readable Format - Client Application (section 2.7.4.1)
Directory Replication	Replicate Changes within a Domain - Domain Controller (section 2.7.5.1) Replicate Changes to a GC or a Partial Replica by using RPC - Domain Controller (section 2.7.5.2) Transferring a FSMO Role - Domain Controller (section 2.7.5.3)
Trust Management	Create a Trust - Domain Controller (section 2.7.6.1)
Domain Services	Join a Domain with a New Account - Domain Client (section 2.7.7.1) Unjoin from the Domain - Domain Client (section 2.7.7.2) Locate a Domain Controller - Domain Client (section 2.7.7.3.1)

Detailed descriptions for these use cases are provided in subsequent sections.

2.7.1 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, which, in a general sense, are used to access and manipulate data that is stored in the Active Directory System, perform object management 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 an object is no longer needed, it can be deleted.

The following use case diagram illustrates the use cases of object management.

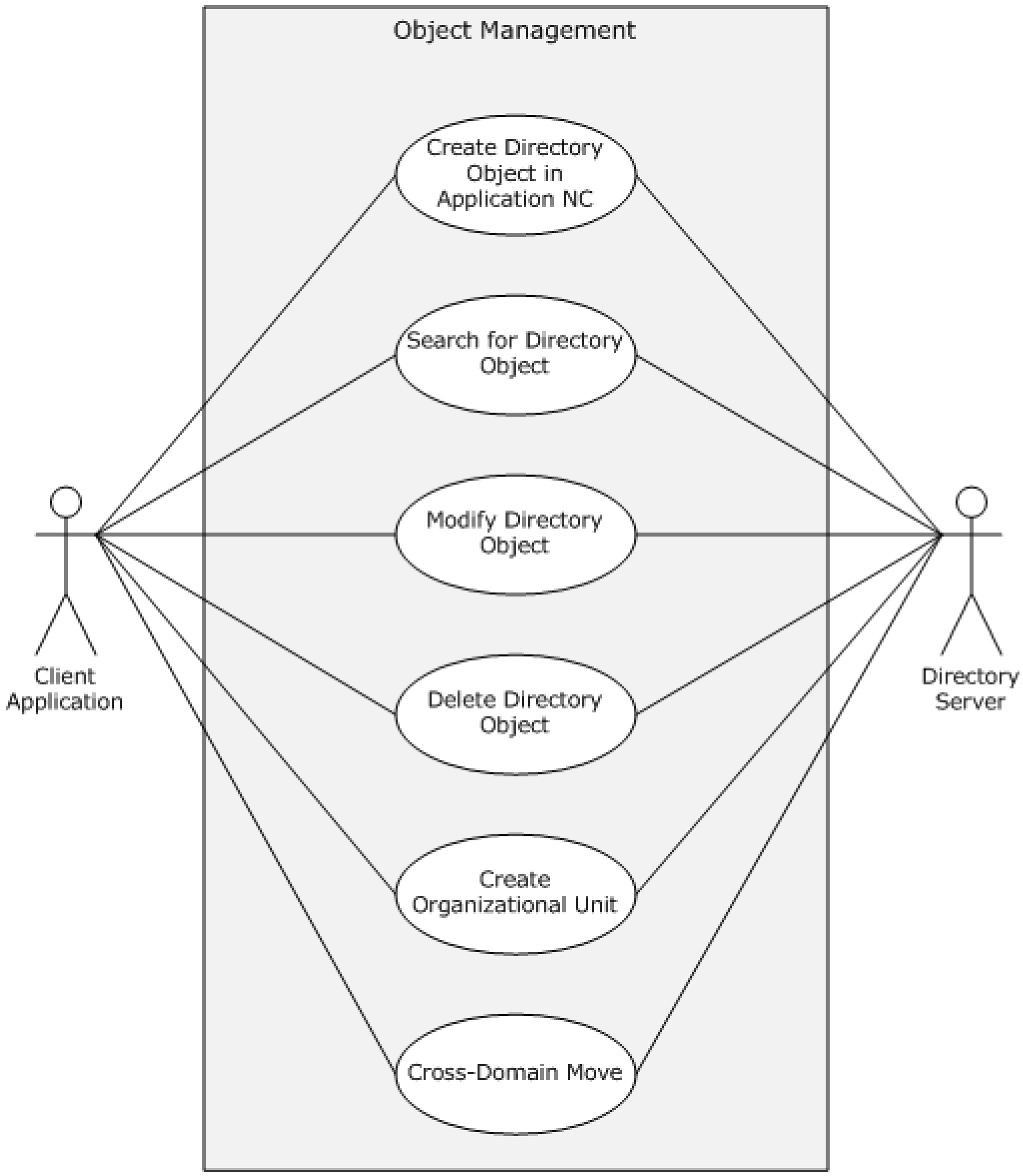

Figure 6: Use cases for object management

2.7.1.1 Create Directory Object - Client Application

In this use case, an administrator wants to create a new directory object on an existing application NC to store information that could be used by applications on the client. 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 using the 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 .

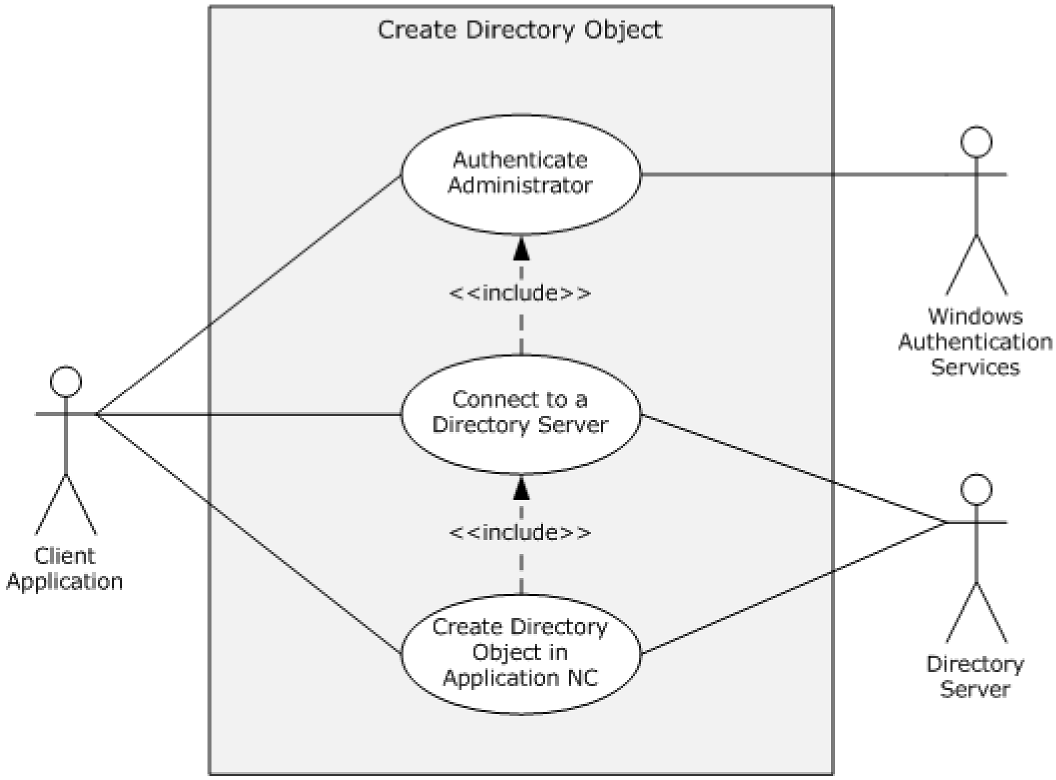

Figure 7: Use case diagram for creating a directory object on an application NC

Actors

- Client Application

  The Client Application is the primary actor. It is the entity that prepares the connection to the Directory Server, submits the request to create the object, and relays the response to the administrator.

- Windows Authentication Services

  The Windows Authentication Services [MS-AUTHSOD] is the supporting actor that authenticates the administrator's identity. This is done so that access control decisions can be made by the Active Directory System.

- Directory Server

  The Directory Server is the supporting actor that receives the creation request and creates the application directory object.

Stakeholders

- Administrator

  The administrator is the one who initiates operations such as create, search, modify, and delete on the application directory object. The administrator is primarily interested in whether the operations are successfully completed and in receiving an error message otherwise.

- Applications

  Applications on the client are the entities that store information in the application directory for later retrieval and use in various operations.

- Application NC

  The application NC is the naming context of the directory that contains the application-specific directory objects.

Preconditions

- The system-wide preconditions described in section 2.6 must be satisfied. The Active Directory System must complete initialization, as described in section 2.6.
- The Client Application 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 2.7.3 describes schema extensions in detail.
- The Active Directory System must host an application NC on which the Client Application is configured to store its application data.

Main Success Scenario

1. **Trigger:** The administrator initiates this activity by providing the name of the directory object as input to the Client Application with **credentials**. The administrator then invokes the operation that creates the application directory object.
2. The Client Application establishes a connection to the Directory Server. Windows Authentication Services authenticates the Client Application using the supplied credentials ([MS-AUTHSOD] section 2).
3. 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.
4. The Directory Server verifies that the credentials supplied through the Client Application have the necessary access-control rights to complete the operation ([MS-ADTS] section 5.1.3).
5. The Directory Server creates an object under the application NC with the name and other attributes supplied by the Client Application. The directory object is also populated with attributes that are mandated by the server's processing rules and constraints ([MS-ADTS] sections 3.1.1.5.1 and 3.1.1.5.2).
6. The Directory Server sends a response to the Client Application indicating that the new application directory object has been successfully created.

Postconditions: The new application directory object is created and ready for use.

Extensions

- If the credentials passed through the Client Application have 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 to the Client Application indicating that it supplied credentials with insufficient access-control rights to create the new application directory object.

- If the RDN value (that is, the name of the directory object 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:

  1-5. Same as Main Success Scenario.

  6. The Directory Server sends a response to the Client Application indicating that the provided object name 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:

  1-5. Same as Main Success Scenario.

  6. The Directory Server sends a response 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 Client Application is a security principal in AD DS:

  1-5. Same as Main Success Scenario.

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

2.7.1.2 Search for Directory Object - Client Application

In this use case, an administrator or User wants to inspect the attribute values for a given set of directory objects in order to make informed decisions concerning the Active Directory System. To achieve this task, 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.

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

Context of Use: An administrator or User wants to search\retrieve the attribute values of the existing directory object.

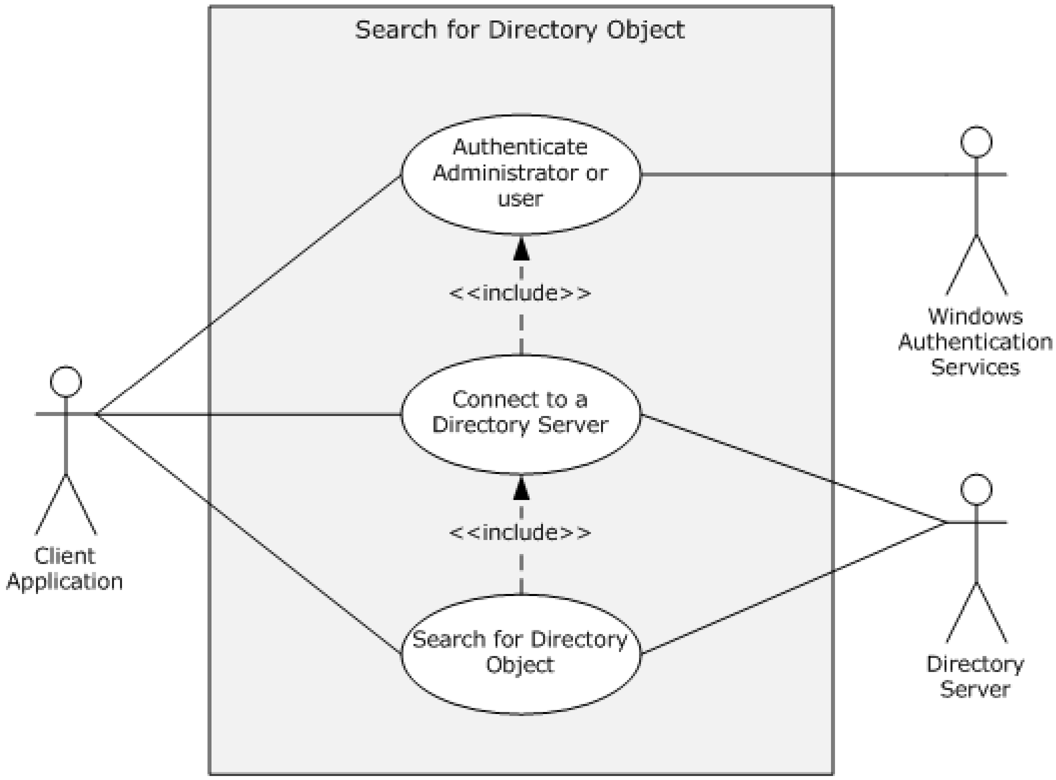

Figure 8: Use case diagram for searching for a directory object

Actors

- Client Application

  The Client Application is the primary actor. It is the entity that prepares the connection to the Directory Server, submits search requests to the directory on behalf of the user, and returns the results to the user.

- Windows Authentication Services

  The Windows Authentication Services [MS-AUTHSOD] is the supporting actor that authenticates the administrator's or user's identity. This is done so that access control decisions can be made by the Active Directory System.

- Directory Server

  The Directory Server is the supporting actor that receives the search request and performs the search of the application directory.

Stakeholders

- Administrator or User

  The administrator or user is the entity that initiates the search in the application directory. The administrator or user is primarily interested in obtaining the search results.

- Directory

The application directory is the directory that contains the application-specific directory objects.

In this operation, the directory is left unchanged.

Preconditions

- The system-wide preconditions described in section 2.6 must be satisfied. The Active Directory System must complete initialization, as described in section 2.6.
- The Client Application must have access to a directory server to which it can establish a connection (if it is not already connected) and send the request.

Main Success Scenario

1. **Trigger:** The administrator or user initiates a search by providing the search criteria for the directory object(s) that are of interest as input to the Client Application with credentials. The administrator or user then invokes the operation that searches for a directory object. The search criteria also specify what information about each object is to be returned.
2. The Client Application establishes a connection to the Directory Server. Windows Authentication Services authenticates the Client Application using the supplied credentials ([MS-AUTHSOD] section 2).
3. The Client Application sends a request to the Directory Server asking it to search for the directory object(s) and specifying the search criteria.
4. The Directory Server verifies that the credentials supplied through the Client Application have the necessary access-control rights to complete the operation ([MS-ADTS] section 5.1.3).
5. 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.
6. The Directory Server sends a response to the Client Application that contains the extracted information.

Postconditions: Information for the directory object is available to the Client Application.

Extensions

- If the search criteria supplied by the Client Application returns a result set that is larger than the configured MaxPageSize ([MS-ADTS] section 3.1.1.3.4.6):

  1-5. Same as Main Success Scenario.

  6. The Directory Server sends a response 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 Client Application would potentially return objects located on a different NC:

  1-4. Same as Main Success Scenario.

  5. Since the Directory Server that the Client Application is connected to does not host the objects specified by the search criteria, 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).

6. The Directory Server sends a response to the Client Application indicating that a referral error has occurred.

2.7.1.3 Modify Directory Object - Client Application

A common activity for an administrator is to make modifications to objects. Timely updates on these directory objects ensure that the data in the system is current, which enables the proper functioning of the Active Directory System. 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 to an existing directory object by using the Client Application.

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

Context of Use: An administrator want to modify attributes of already existing directory objects

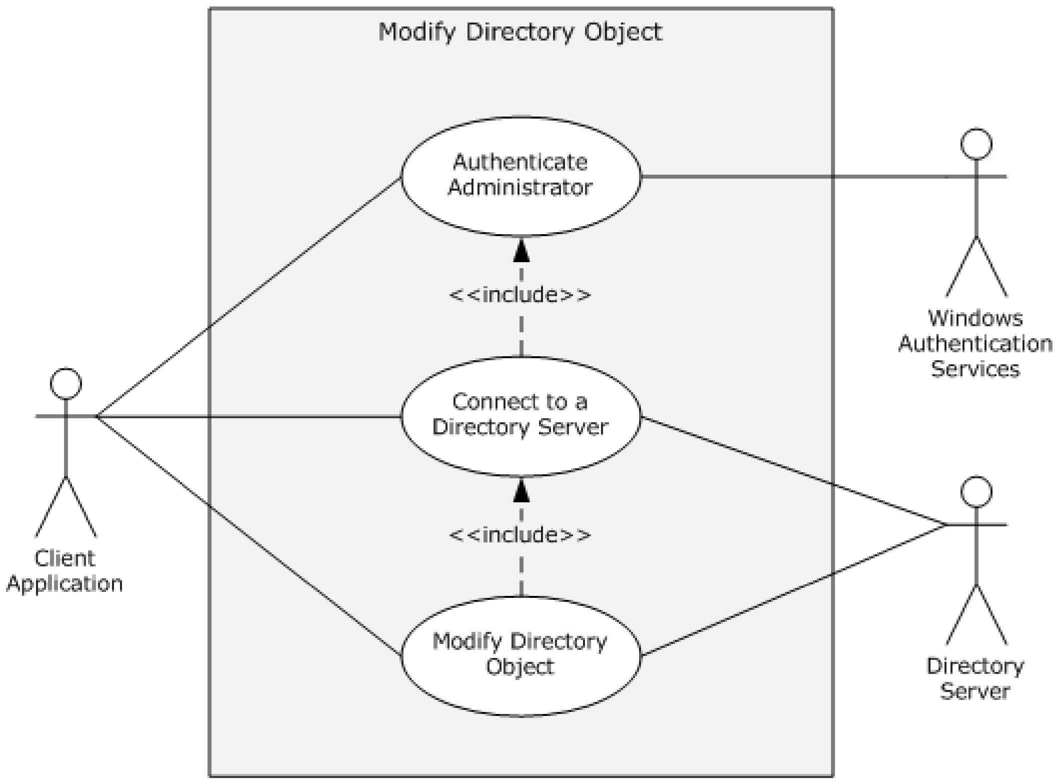

Figure 9: Use case diagram for modifying a directory object

Actors

- Client Application

  The Client Application is the primary actor. It is the entity that prepares the connection to the Directory Server, submits the modification request, and relays the response to the administrator.

- Windows Authentication Services

The Windows Authentication Services [MS-AUTHSOD] is the supporting actor that authenticates the administrator's identity. This is done so that access control decisions can be made by the Active Directory System.

- Directory Server

  The Directory Server is the supporting actor that receives the modification request and modifies the directory object.

Stakeholders

- Administrator

  The administrator is the one who initiates operations such as create, search, modify, and delete on the application directory object. The administrator is primarily interested in whether the operations are successfully completed and in receiving an error message otherwise.

- Directory

  The directory is the entity that contains the object being modified.

Preconditions

- The system-wide preconditions described in section 2.6 must be satisfied. The Active Directory System must complete initialization, as described in section 2.6.
- The Client Application 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.

Main Success Scenario

1. **Trigger:** The administrator initiates the modify operation by providing the name of the directory object to modify as input to the Client Application with credentials, including the attribute(s) being modified on the object and the list of modifications to be made to those attributes.
2. The Client Application establishes a connection to the Directory Server. Windows Authentication Services authenticates the Client Application using the supplied credentials ([MS-AUTHSOD] section 2).
3. The Client Application sends a modify request to the Directory Server asking it to make the appropriate modifications on the directory object.
4. The Directory Server verifies that the credentials supplied through the Client Application have the necessary access-control rights to complete the operation ([MS-ADTS] section 5.1.3).
5. The Directory Server modifies the object as specified by the Client Application and makes any additional modifications that are mandated by the server's processing rules and constraints ([MS-ADTS] sections 3.1.1.5.1, 3.1.1.5.3, and 3.1.1.5.4).
6. The Directory Server sends a response to the Client Application indicating that the modifications were successfully completed.

Postconditions: The directory object is modified.

Extensions

- 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 described in [MS-ADTS] sections 3.1.1.5.1 and 3.1.1.5.3.

2.7.1.4 Delete Directory Object - Client Application

An administrator can perform maintenance on an Active Directory System by removing objects that are no longer needed by the applications on the client. 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 (not a leaf object).

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

Context of Use: An administrator wants to delete an already existing directory object.

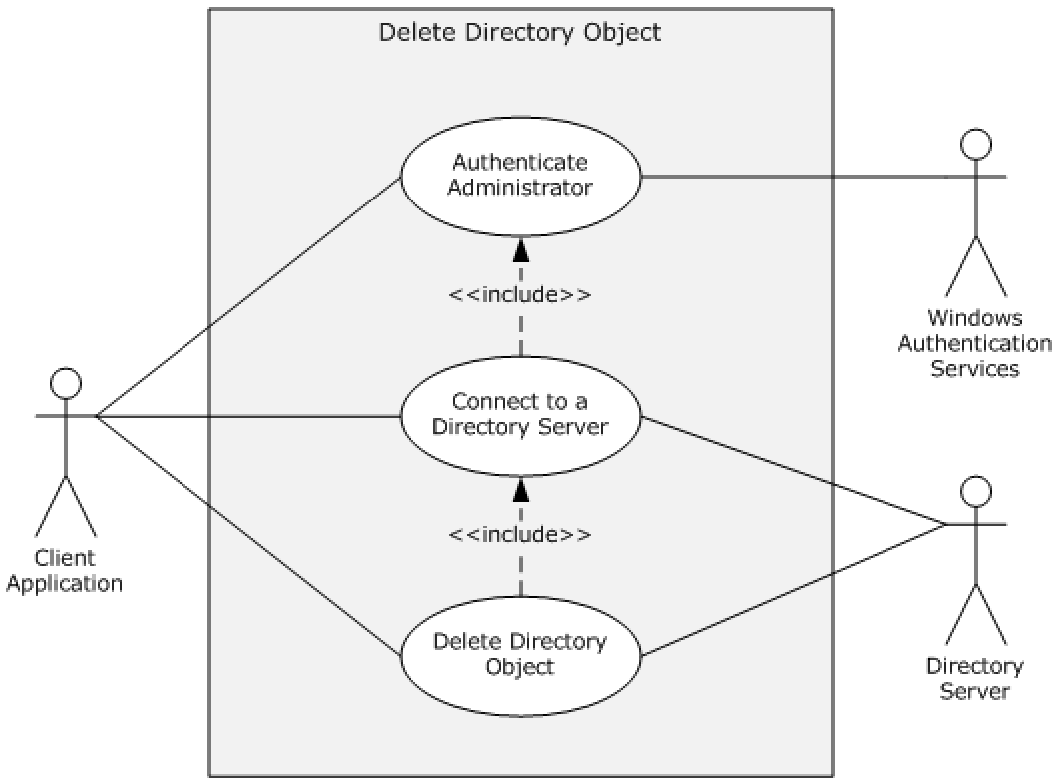

Figure 10: Use case diagram for deleting a directory object

Supporting Actors

- Client Application

  The Client Application is the primary actor. It is the entity that prepares the connection to the Directory Server, submits the request to delete an object, and relays the response to the administrator.

- Windows Authentication Services

The Windows Authentication Services [MS-AUTHSOD] is the supporting actor that authenticates the administrator's identity. This is done so that access control decisions can be made by the Active Directory System.

- Directory Server

  The Directory Server is the supporting actor that receives the deletion request and deletes the directory object.

Stakeholders

- Administrator

  The administrator is the one who initiates operations such as create, search, modify, and delete on the application directory object. The administrator is primarily interested in whether the operations are successfully completed and in receiving an error message otherwise.

- Directory

  The directory is the entity that contains the object being deleted.

Preconditions

- The system-wide preconditions described in section 2.6 must be satisfied. The Active Directory System must complete initialization, as described in section 2.6.
- The Client Application 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.

Main Success Scenario

1. **Trigger:** The administrator initiates the delete operation by providing the name of the directory object to delete to the Client Application with credentials. The administrator then selects the directory object to delete and submits the deletion request to the Active Directory System.
2. The Client Application establishes a connection to the Directory Server. Windows Authentication Services authenticates the Client Application using the supplied credentials ([MS-AUTHSOD] section 2).
3. The Client Application sends a delete request to the Directory Server asking it to delete the specified directory object.
4. The Directory Server verifies that the credentials supplied through the Client Application have the necessary access-control rights to complete the operation ([MS-ADTS] section 5.1.3)
5. The Directory Server deletes the object specified by the client and makes any additional modifications that are mandated by the server's processing rules and constraints ([MS-ADTS] sections 3.1.1.5.1 and 3.1.1.5.5).
6. The Directory Server sends a response to the Client Application indicating that the deletion was successfully completed.

Postconditions: The directory object is no longer available.

Extensions

- If the Client Application attempted to delete a non-leaf directory object:

  1-5. Same as Main Success Scenario

  6. The Directory Server sends a response to the Client Application indicating that it cannot delete a non-leaf object ([MS-ADTS] section 3.1.1.5.5.5).

- If the Client Application attempted to delete a directory object that is owned by the system ([MS-ADTS] section 3.1.1.5.5.3):

  1-5. Same as Main Success Scenario.

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

2.7.1.5 Create Organizational Unit - Client Application

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 through the Client Application.

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

Context of Use: An administrator wants to create a different group to represent a specific set of directory objects. On these objects the administrator can apply a common policy.

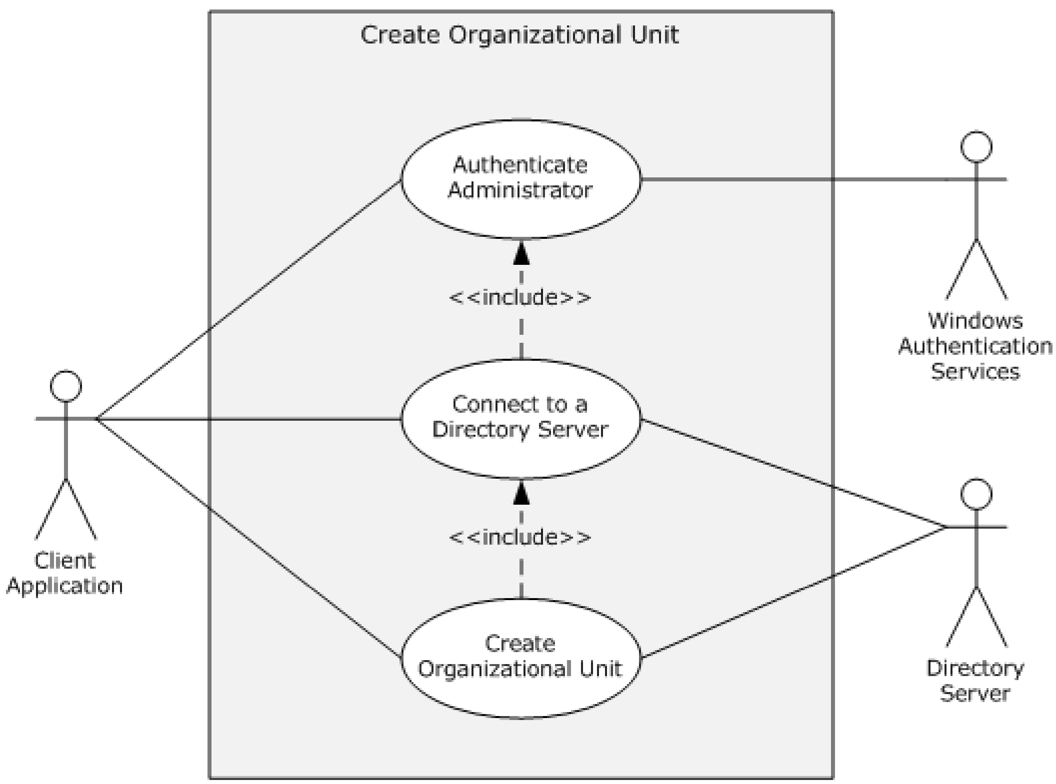

Figure 11: Use case diagram for creating an organizational unit

Primary Actor: The primary actor is the Client Application.

Actors

- Client Application

  The Client Application is the primary actor. It is the entity that prepares the connection to the Directory Server, submits the request to create an organizational unit, and relays the response to the administrator.

- Windows Authentication Services

  The Windows Authentication Services [MS-AUTHSOD] is the supporting actor that authenticates the administrator's identity. This is done so that access control decisions can be made by the Active Directory System.

- Directory Server

  The Directory Server is the supporting actor that receives the creation request and creates the organizational unit in the directory.

Stakeholders

- Administrator

  The administrator is the one who initiates operations such as create, search, modify, and delete on the application directory object. The administrator is primarily interested in whether the operations are successfully completed and in receiving an error message otherwise.

- Group Policy

  Group Policy allows managed configurations for users and computers.

  The Active Directory System guarantees that, if possible, the new organizational unit object is created, contains all the attribute values set by the Client Application, and is ready for Group Policy assignments.

- Directory

  The directory is the entity that will contain the object for the organizational unit.

Preconditions

- The system-wide preconditions described in section 2.6 must be satisfied. The Active Directory System must complete initialization, as described in section 2.6.

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

Main Success Scenario

1. **Trigger:** The administrator provides the name of the organizational unit as input to the Client Application with credentials and invokes the operation that creates the organizational unit.

2. The Client Application establishes a connection to the Directory Server. Windows Authentication Services authenticates the Client Application using the supplied credentials ([MS-AUTHSOD] section 2).

3. 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.

4. The Directory Server verifies that the credentials supplied through the Client Application have the necessary access-control rights to complete the operation ([MS-ADTS] section 5.1.3).

5. 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 also populated with attributes that are mandated by the server's processing rules and constraints ([MS-ADTS] sections 3.1.1.5.1 and 3.1.1.5.2).

6. The Directory Server sends a response to the Client Application indicating that the new organizational unit has been successfully created.

Postconditions: The organization unit is created and ready for use.

Extensions

- If the RDN value (that is, the name of the organizational unit to be created) supplied through the Client Application is not unique under the same parent container, as required by [MS-ADTS] section 3.1.1.5.2.2:

  1-5. Same as Main Success Scenario.

  6. The Directory Server sends a response to the Client Application indicating that the object name that was 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:

  1-5. Same as Main Success Scenario.

  6. The Directory Server sends a response to the Client Application indicating that the missing attribute is required in the request.

2.7.1.6 Cross-Domain Move - Client Application

In this use case, cross-domain movement of an object is performed between two domain controllers that are present in different domains.

Goal: To move an object from one domain to another domain.

Context of use: To perform cross-domain movement when an object is required to be moved from one domain to another domain. An administrator launches the Client Application in order to perform the action.

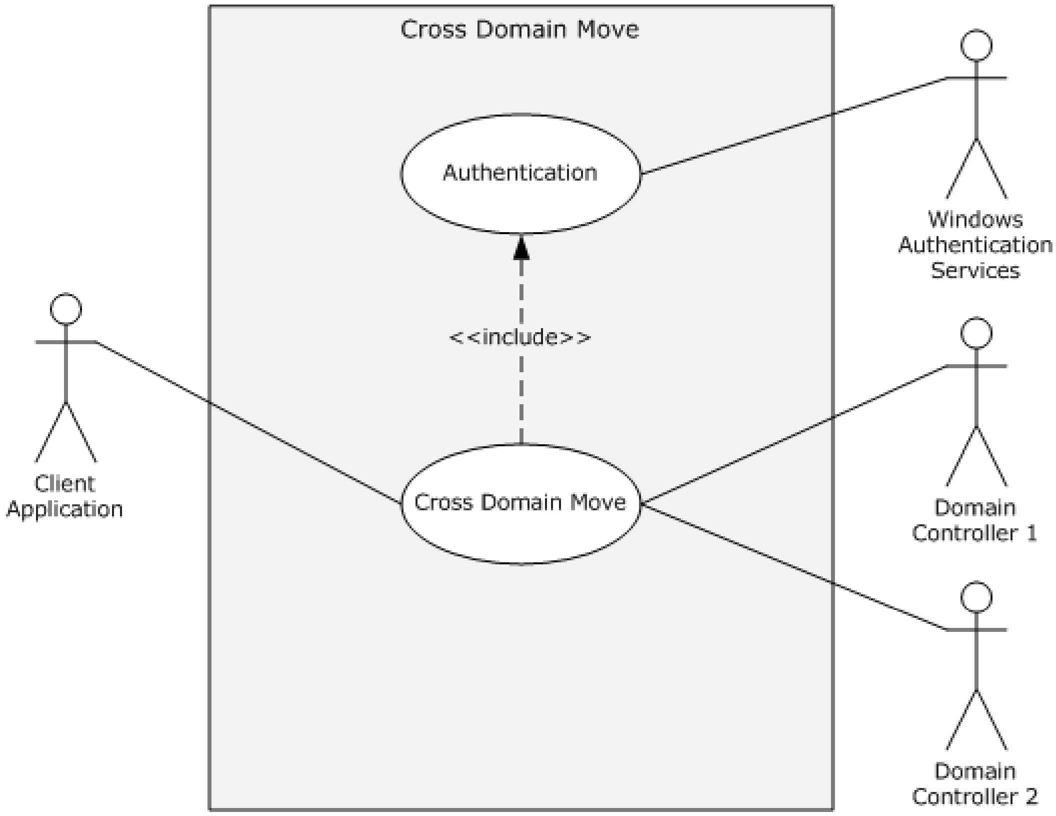

Figure 12: Use case diagram for performing a cross-domain move

Actors

- Client Application

  The Client Application is the primary actor that initiates the cross-domain movement of a particular object.

- Windows Authentication Services

  The Windows Authentication Services [MS-AUTHSOD] is the supporting actor that authenticates the administrator's identity. This is done so that access control decisions can be made by the Active Directory System.

- Domain Controller 1 (DC1)

  DC1 is the supporting actor that is a domain controller in a domain.

- Domain Controller 2 (DC2)

  DC2 is the supporting actor that is a domain controller in another domain.

Stakeholders

- Domain Administrators and Applications

Domain administrators and applications are the entities that move objects from one domain to another.

Preconditions

- The environment described in section 2.5 is in place and the system-wide preconditions described in section 2.6 are satisfied. The Active Directory System must complete initialization, as described in section 2.6.
- DC1 and DC2 are in different domains.
- The requester must have permissions to perform a cross-domain move operation as described in [MS-ADTS] section 3.1.1.5.4.2.1.

Main Success Scenario

1. **Trigger:** An administrator triggers a request on the Domain Client for movement of an object from DC1 to DC2.
2. The Client Application establishes a connection to DC1. Windows Authentication Services authenticates the Client Application using the supplied credentials ([MS-AUTHSOD] section 2).
3. The Domain Client sends a Modify DN request to DC1 for movement of the object as specified in [MS-ADTS] section 3.1.1.5.4.
4. DC1 sends an interdomain move request to DC2 as specified in [MS-ADTS] section 3.1.1.5.4.2.3.
5. DC2 adds a new object to its replica.
6. DC1 creates a proxy object and deletes the original object.

Postcondition

An object is moved from one domain to the other.

Extensions

None.

2.7.2 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 in the Object Management category (section 2.7.1).

Note These use cases are applicable only to Active Directory Domain Services (AD DS); they are not applicable to Active Directory Lightweight Directory services (AD LDS).

The following use case diagram illustrates the use cases of identity lifecycle management.

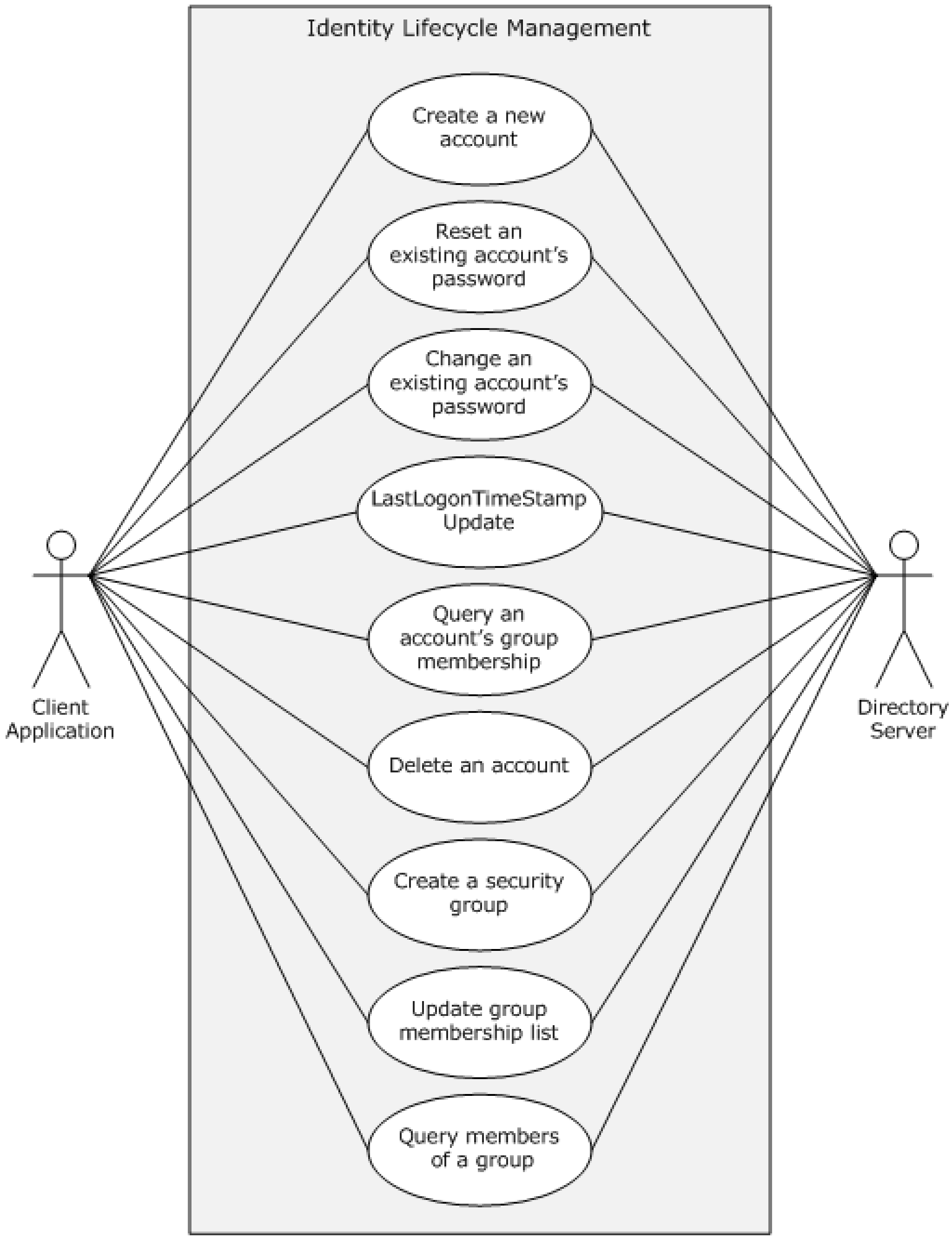

Figure 13: Use cases for identity lifecycle management

2.7.2.1 Create a New Account - Client Application

In this use case, an administrator wishes to create a new account in the directory to allow a user to access directory resources. The administrator launches the Client Application to create a new account. The Client Application establishes a connection to the Active Directory System.

Goal: Create a new account in the directory.

Context of Use: An administrator wants to create a new account in the directory.

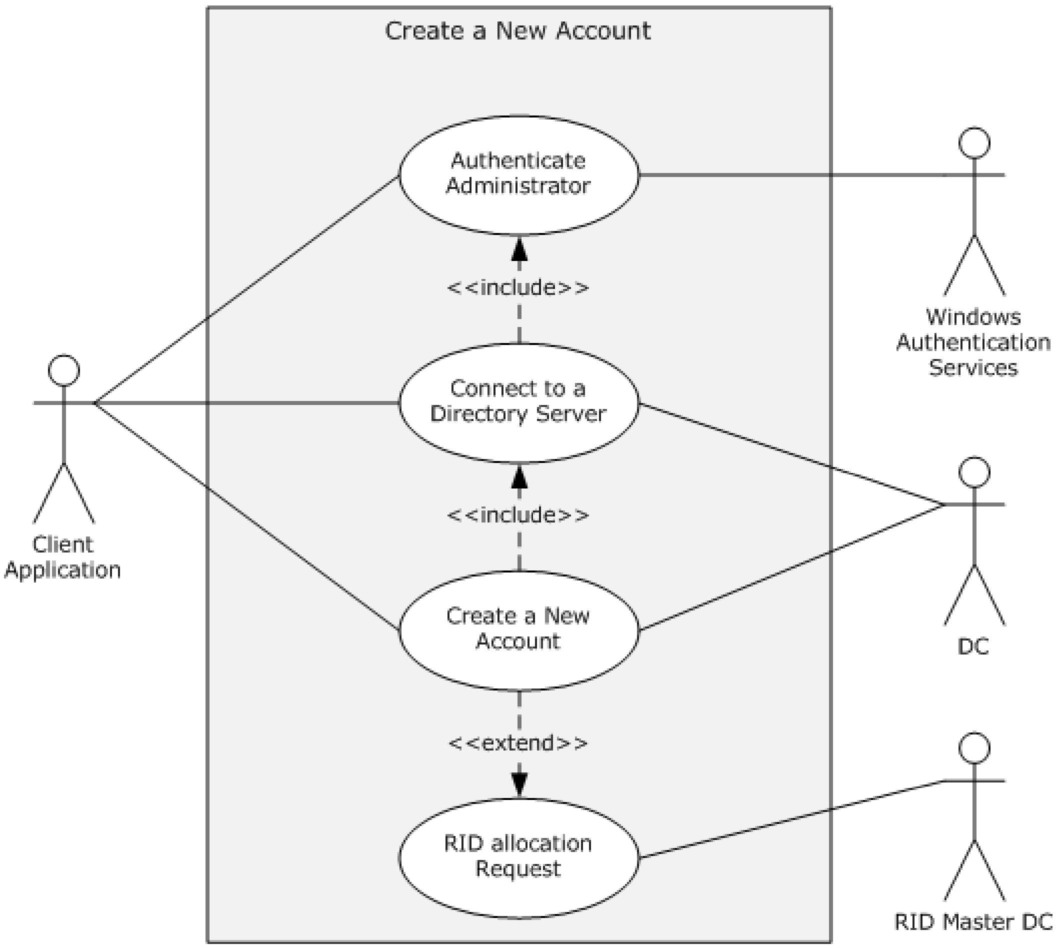

Figure 14: Use case diagram for creating a new account

Actors

- Client Application

  The Client Application is the primary actor. It is the entity that prepares the connection to the DC, submits the request to create the new account, and relays the response to the administrator.

- Windows Authentication Services

  The Windows Authentication Services [MS-AUTHSOD] is the supporting actor that authenticates the administrator's identity. This is done so that access-control decisions can be made by the Active Directory System.

- DC

  The DC is the supporting actor that receives the creation request and creates the new account.

- RID Master DC

The RID Master DC is a supporting actor. It is the domain controller that is the owner of the RID Master FSMO role for the domain.

Stakeholders

- Administrator

  The administrator is the one who initiates operations such as create, reset, change, query group members, create security group, modify group member list, and delete on the new account. The administrator is primarily interested in whether the operations are successfully completed and in receiving an error message otherwise.

- Directory

  The directory is the entity that contains the account being created.

Preconditions

- The system-wide preconditions described in section 2.6 must be satisfied. The Active Directory System must complete initialization, as described in section 2.6.
- 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.

Main Success Scenario

1. **Trigger:** The administrator launches the Client Application by providing the account name for the new account as input to the Client Application with credentials, indicating that creation of a new account is desired.
2. The Client Application establishes a connection to the DC. Windows Authentication Services authenticates the Client Application using the supplied credentials ([MS-AUTHSOD] section 2).
3. The Client Application sends a request to the DC asking it to create a new account and specifies the account name for the new account.
4. The DC verifies that the credentials supplied through the Client Application have the necessary access-control rights to complete the operation ([MS-ADTS] section 5.1.3).
5. The DC validates the constraints on the new account name, as described in [MS-SAMR] sections 3.1.1.6 and 3.1.1.8.4.
6. The DC 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] sections 3.1.1.5.1 and 3.1.1.5.2 and [MS-SAMR] sections 3.1.1.8 and 3.1.1.9).
7. The DC sends a response to the Client Application indicating that the new account has been successfully created.

Postconditions: The new account is created and ready for use.

Extensions

- If the credentials passed through the Client Application have insufficient access-control rights to set the password on the account:

  1-4. Same as Main Success Scenario.

5. The DC sends a response to the Client Application indicating that the Client Application has supplied credentials with insufficient access-control rights to set the password on the account.

- If the account name that is supplied through the Client Application does not satisfy the account name constraints that are outlined in [MS-SAMR] section 3.1.1.6:

  1-5. Same as Main Success Scenario.

  6. The DC sends a response to the Client Application indicating that the supplied account name does not meet the constraints.

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

  1-5. Same as Main Success Scenario.

  6. The DC sends a response to the Client Application indicating that the supplied account name is already in use by an existing account.

- If the DC has used all of the **relative identifiers (RIDs)** that were allocated to it by RID Master FSMO role owner:

  1-5. Same as Main Success Scenario. Then after step 5, the DC sends a RID allocation request to the RID Master DC according to the rules specified in [MS-DRSR] section 4.1.10.4.3 to obtain a new RID range.

  6-7. Same as main Success Scenario.

2.7.2.2 Reset an Existing Account's Password - Client Application

In this use case, a user has forgotten the password for his or her account and contacts an administrator. The administrator wants to reset the account's password to a known value so that 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.

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

Context of Use: When a user has forgotten his or her password and wants to reset it to a new value.

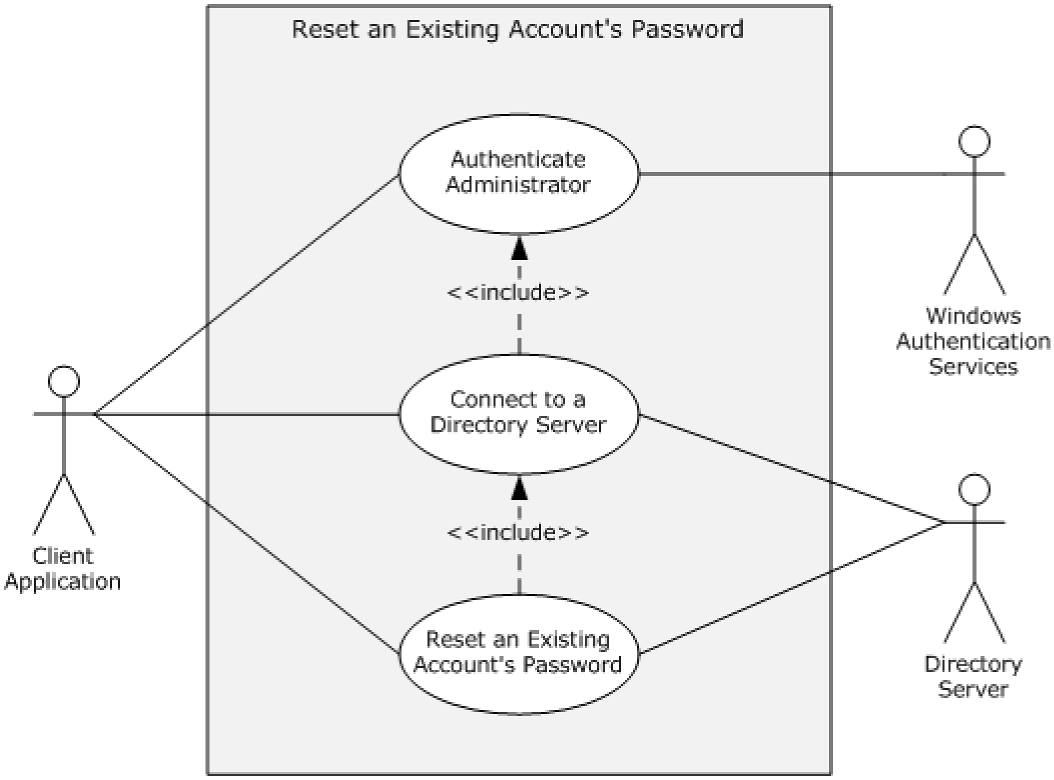

Figure 15: Use case diagram for resetting the password of an existing account

Actors

- Client Application

  The Client Application is the primary actor. It is the entity that prepares the connection to the Directory Server, submits the request to reset the password, and relays the response to the administrator.

- Windows Authentication Services

  The Windows Authentication Services [MS-AUTHSOD] is the supporting actor that authenticates the administrator's identity. This is done so that access control decisions can be made by the Active Directory System.

- Directory Server

  The Directory Server is the supporting actor that receives the password-reset request and performs the tasks associated with resetting a user's password in the directory.

Stakeholders

- Administrator

  The administrator is the one who initiates operations such as create, reset, change, query group members, create security group, modify group member list, and delete on an account. The administrator is primarily interested in whether the operations are successfully completed and in receiving an error message otherwise.

- User

  The user is the person who needs to access directory resources.

  For the user, the Active Directory System guarantees that, if possible, the password for the user's account is reset.

- Directory

  The directory is the entity that contains the user's existing account.

Preconditions

- The system-wide preconditions described in section 2.6 must be satisfied. The Active Directory System must complete initialization, as described in section 2.6.
- 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 on which the password change is being performed must exist.

Main Success Scenario:

1. **Trigger:** The administrator provides the account name of the existing account and the new password as input to the Client Application and invokes the operation that resets the password of an account.
2. The Client Application establishes a connection to the Directory Server. Windows Authentication Services authenticates the Client Application using the supplied credentials ([MS-AUTHSOD] section 2).
3. 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 the new password supplied by the administrator.
4. The Directory Server verifies that the credentials supplied through the Client Application have the necessary access-control rights to complete the operation. ([MS-ADTS] section 5.1.3).
5. The Directory Server verifies that the new password satisfies the password policy as outlined in [MS-SAMR] section 3.1.1.7.1.
6. 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 ([MS-ADTS] sections 3.1.1.5.1 and 3.1.1.5.3 and [MS-SAMR] section 3.1.1.8.7).
7. The Directory Server sends a response to the Client Application indicating that the password has been successfully updated.

Postconditions: The account's password is reset.

Extensions

- If the credentials passed through the Client Application have 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 to the Client Application indicating that the Client Application has supplied credentials with insufficient access-control rights to set the password on the account.

- If the password supplied through the Client Application does not satisfy the password constraints outlined in [MS-SAMR] section 3.1.1.7.1:

  1-5. Same as Main Success Scenario.

  6. The Directory Server sends a response to the client indicating that the supplied password does not meet the constraints.

2.7.2.3 Change an Existing Account's Password (PDC) - Client Application

In this use case, a user whose account is present in an Active Directory domain wishes to change his or her existing password to a new value. The user launches a Client Application to change the password on his or her account. The Client Application establishes a connection to the Active Directory System by connecting to a domain controller that is the primary domain controller (PDC) FSMO role owner for the domain.

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

Context of use: This use case is used when a client machine connects to the PDC for LDAP and domain services and the user wants to change the password of the user account.

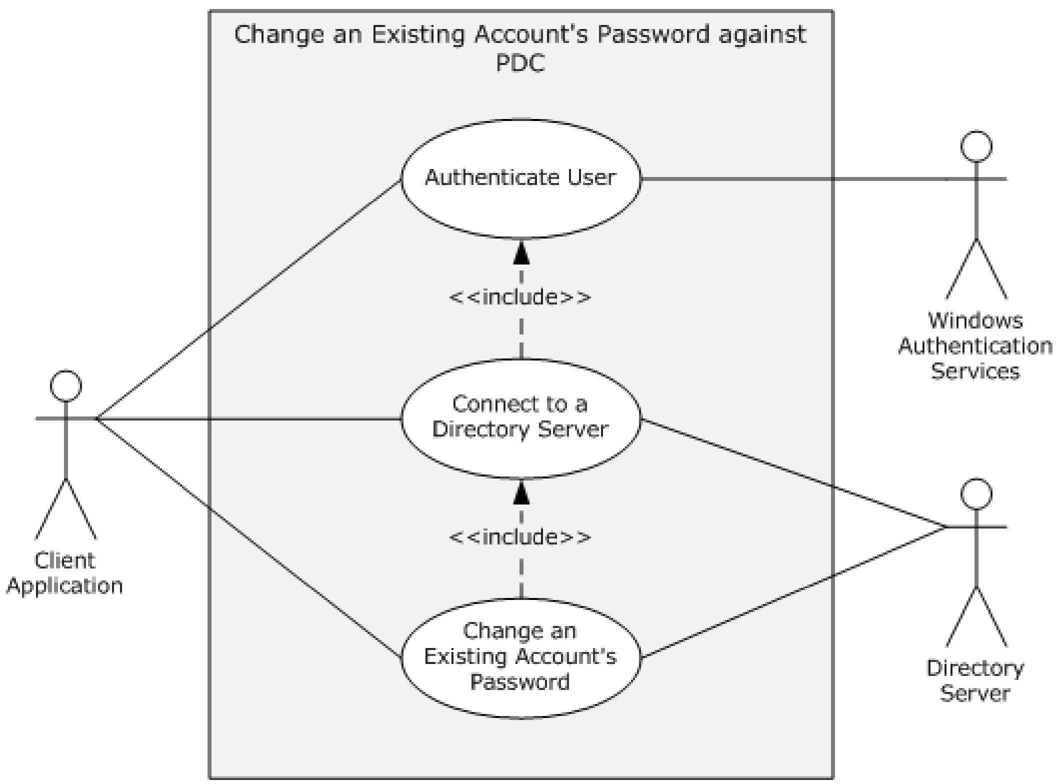

Figure 16: Use case diagram for changing the password of an existing account (PDC)

Actors

- Client Application

  The Client Application is the primary actor. It is the entity that prepares the connection to the Directory Server, submits the request to change the password, and relays the response to the user.

- Windows Authentication Services

  The Windows Authentication Services [MS-AUTHSOD] is the supporting actor that authenticates the user's identity. This is done so that access control decisions can be made by the Active Directory System.

- Directory Server

  The Directory Server is the supporting actor that receives the password-change request and performs the tasks associated with changing a user's password in the directory. The directory server is the owner of PDC FSMO role for the domain.

Stakeholders

- User

  The user is the person who initiates the password change on his or her existing account. The user is primarily interested in whether the password was successfully changed and in receiving an error message otherwise.

- Directory

  The directory is the entity that contains the user's existing account.

Preconditions

- The system-wide preconditions described in section 2.6 must be satisfied. The Active Directory System must complete initialization, as described in section 2.6.
- 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 on which the password change is being performed must exist.

Main Success Scenario

1. **Trigger:** The user provides the account name of the existing account, the existing password for the account, and the new value for the password to the Client Application and invokes the operation that changes the password of the account.
2. The Client Application establishes a connection to the Directory Server. Windows Authentication Services authenticates the Client Application using the supplied credentials, as described in [MS-AUTHSOD] section 2.
3. 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, the current password, and the new password supplied by the user.
4. The Directory Server verifies that the credentials supplied through the Client Application have the necessary access-control rights to complete the operation, as described in [MS-ADTS] section 5.1.3.

5. The Directory Server verifies that the current password supplied through the Client Application matches the account's password stored in the directory.

6. The Directory Server verifies that the new password satisfies the password policy, as described in [MS-SAMR] section 3.1.1.7.1.

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 ([MS-ADTS] sections 3.1.1.5.1 and 3.1.1.5.3 and [MS-SAMR] section 3.1.1.8.7).

8. The Directory Server sends a response to the Client Application indicating that the password has been successfully updated.

Postconditions: The account's password is changed.

Extensions:

- If the credentials passed through the Client Application have 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 to the Client Application indicating that it supplied credentials with 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 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.

  1-6. Same as Main Success Scenario.

  7. The Directory Server sends a response to the Client Application indicating that the supplied password does not meet the constraints.

2.7.2.4 Change an Existing Account's Password (DC) - Client Application

In this use case, a user whose account is present in an Active Directory domain wishes to change his or her existing password to a new value. The user launches a Client Application to change the password on the account. The Client Application establishes a connection to the Active Directory System by connecting to a DC that is not the owner of the PDC FSMO role for the domain. This use case highlights the communication between the DC and the PDC in the domain.

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

Context of use: The user wants to change the password on his or her user account.

Figure 17: Use case diagram for changing the password of an existing account (DC)

Actors

- Client Application

  The Client Application is the primary actor. It is the entity that prepares the connection to the Directory Server, submits the request to change the password, and relays the response to the user.

- Windows Authentication Services

The Windows Authentication Services [MS-AUTHSOD] is the supporting actor that authenticates the user's identity. This is done so that access control decisions can be made by the Active Directory System.

- DC

  A domain controller that is not the owner of the PDC FSMO role for the domain. This is the supporting actor that receives the password-change request, performs the tasks associated with changing a user's password in the directory, and sends a password update request to the PDC.

- PDC

  The primary domain controller of the domain. This is the supporting actor that receives the password update request from the DC and updates the user-password details in its directory database. The PDC is owner of PDC FSMO role for the domain.

Stakeholders

- User

  The user is the person who initiates the password change on his or her existing account. The user is primarily interested in whether the password was successfully changed and in receiving an error message otherwise.

- Directory

  The directory is the entity that contains the user's existing account.

Preconditions

- The system-wide preconditions described in section 2.6 must be satisfied. The Active Directory System must complete initialization, as described in section 2.6.
- The Client Application must have connectivity to a DC to which it can establish a connection (if it is not already connected) and send the request.
- The DC must have connectivity to the PDC to which it establishes a secure channel and sends the password update request.
- The account on which the password change is being performed must exist.

Main Success Scenario

1. **Trigger:** The user provides the account name of the existing account, the existing password for the account, and the new value for the password to the Client Application and invokes the operation that changes the password of the account.
2. The Client Application establishes a connection to the DC. Windows Authentication Services authenticates the Client Application using the supplied credentials, as described in [MS-AUTHSOD] section 2.
3. The Client Application sends a request to the DC asking it to change the password of the given account. This request includes the account name, the current password, and the new password supplied by the user.
4. An access check is performed on the DC to ensure that the user has access rights to complete the operation, as described in [MS-ADTS] section 5.1.3.

5. The DC verifies that the current password supplied through the Client Application matches the account's password stored in the directory.

6. The DC verifies that the new password satisfies the password policy, as described in [MS-SAMR] section 3.1.1.7.1.

7. The DC 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 ([MS-ADTS] sections 3.1.1.5.1 and 3.1.1.5.3 and [MS-SAMR] section 3.1.1.8.7).

8. The DC establishes a secure channel with the PDC according the processing rules and constraints specified in the [MS-NRPC] sections 3.1.1 and 3.1.4.3 and [MS-ADTS] section 6.1.6.9.2.

9. The DC sends a password update request to the PDC according the processing rules and constraints specified in [MS-SAMS] section 3.3.5.4.

10. The PDC sends a response to the DC indicating that the password has been successfully updated.

11. The DC sends a response to the Client Application indicating that the password has been successfully updated.

Postconditions: The account's password is changed at the DC and it is also updated for the PDC FSMO role owner of the domain.

2.7.2.5 Change User Account Password Against an RODC - Client Application

In this use case, a user whose account is present in an Active Directory domain wishes to change his or her existing password to a new value. The user launches a Client Application to change the password on his or her account. The Client Application establishes a connection to the Active Directory System, connecting to a read-only domain controller (RODC) to update its password.

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

Context of use: This use case is used when a client machine connects to an RODC for LDAP and domain services and the user wants to change the password of the user account.

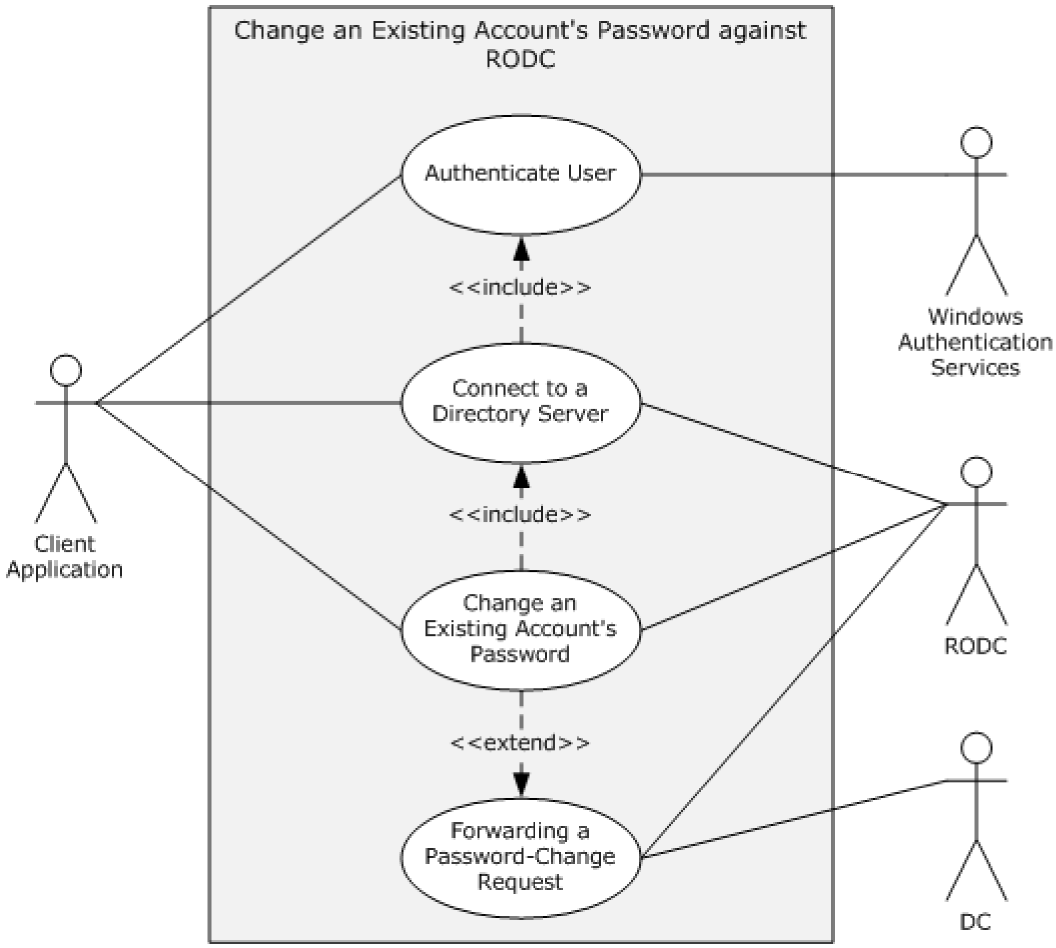

Figure 18: Use case diagram for changing the password of an existing account (RODC)

Actors

- Client Application

  The Client Application is the primary actor. It is the entity that prepares the connection to the Directory Server, submits the request to change the password, and relays the response to the user.

- Windows Authentication Services

  Windows Authentication Services [MS-AUTHSOD] is the supporting actor that authenticates the user's identity. This is done so that access control decisions can be made by the Active Directory System.

- RODC

  A read-only domain controller that contains a read-only replica of the NC that contains the user account. The RODC is the supporting actor that receives the password-change request and performs the tasks associated with changing a user's password in the directory. The RODC forwards the password update request to the DC.

- DC

  A domain controller that contains a writable replica of the NC that contains the user account. The DC is the supporting actor that receives the password update request from the RODC and updates the user password details in its directory database. The DC contains a writable replica of the domain NC in which the user account is present.

Stakeholders

- User

  The user is the person who initiates the password change on his or her existing account. The user is primarily interested in whether the password was successfully changed and in receiving an error message otherwise.

- Directory

  The directory is the entity that contains the user's existing account.

Preconditions

- The system-wide preconditions described in section 2.6 must be satisfied. The Active Directory System must complete initialization, as described in section 2.6.
- The Client Application must have connectivity to an RODC to which it can establish a connection (if it is not already connected) and send the request.
- The RODC must have connectivity to a DC to which it can establish a secure channel and send the password update request.
- The account on which the password change is being performed must exist.

Main Success Scenario

1. **Trigger:** The user provides the account name of the existing account, the existing password for the account, and the new value for the password to the Client Application and invokes the operation that changes the password of the account.
2. The Client Application establishes a connection to the RODC. Windows Authentication Services authenticates the Client Application using the supplied credentials, as described in [MS-AUTHSOD] section 2.
3. The Client Application sends a request to the RODC asking it to change the password of an existing account according to rules specified in one of the following references:
   1. [MS-NRPC] sections 3.5.4.4.6 and 3.5.4.4.5
   2. [MS-SAMR] sections 3.1.5.10.2 and 3.1.5.10.3
4. The RODC processes the request according the following rules.
   1. If the Client Application sends the request in step 3 according to the rules specified in [MS-NRPC], the RODC forwards the request to the DC according to the processing rules specified in [MS-SAMS] section 3.2.4.4.
   2. If the Client Application sends the request in step 3 according to the rules specified in [MS-SAMR], the RODC forwards the request to the DC according to the processing rules specified in [MS-SAMS] section 3.2.4.5.

5. The DC processes the request from the RODC according the rules that are specified in the appropriate section of [MS-SAMS], as indicated in step 4, and sends a response.

6. Upon receiving the success response from the DC, the RODC sends that response to the Client application.

Post conditions: The account's password is changed and it is updated in the writable NC replica of the DC.

2.7.2.6 User Login to Domain Services Using an RODC and Updating the User LastLogonTimeStamp - Client Application

In this use case, an user successfully logs into a domain using an RODC, after which the associated last-login time value is updated in the lastLogonTimeStamp attribute of the user account.

Goal: When the user successfully logs in, the lastLogonTimeStamp attribute of the user account is updated in the directory.

Context of use: This use case is used by an RODC to communicate the user's latest lastLogonTimeStamp to the DC whenever the user logs into the domain. The lastLogonTimeStamp attribute represents the time when the user successfully logged in to the domain. This use case is used in scenarios where the Client Application is connecting to an RODC for directory services.

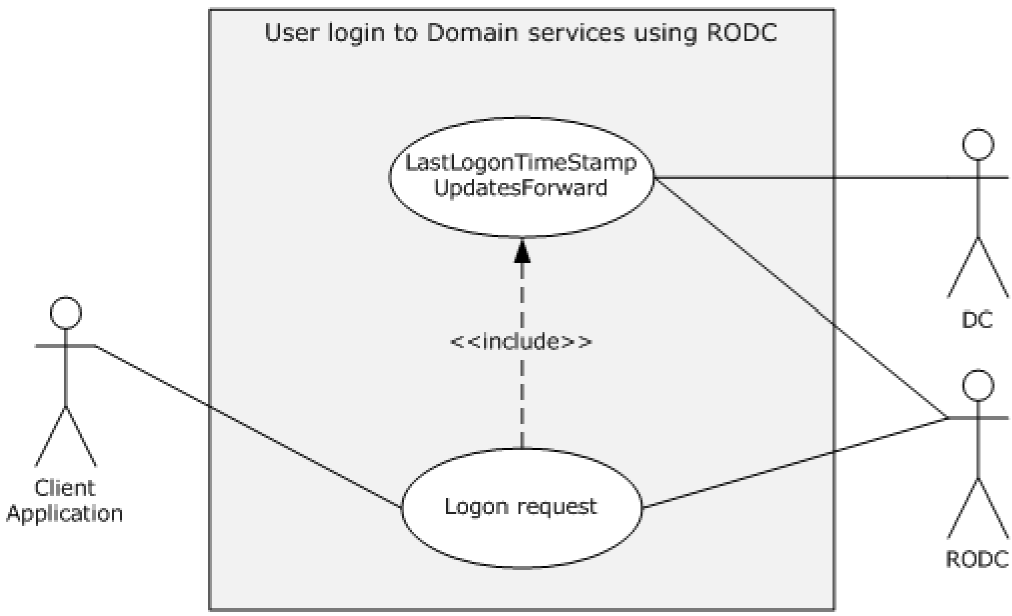

Figure 19: Use case diagram for last-login time value update using an RODC

Actors

- Client Application

  The Client Application is the primary actor. The user is trying to log in to the domain using the Client Application. It is the entity that initiates authentication operations to access a resource in the directory.

- RODC

The RODC is a read-only domain controller. It is the supporting actor to which the Client Application is connected in order to authenticate the user.

- DC

  The DC is a domain controller in the domain. It is the supporting actor that contains a writable replica of the naming context in which the user account is present.

Stakeholders

- User

  The user is the person whose account is being updated with a last-login time value.

- Directory

  The directory is the entity that contains user accounts and logon details of the user.

Preconditions

- The system-wide preconditions described in section 2.6 must be satisfied. The Active Directory System must complete initialization, as described in section 2.6.
- The Client Application must have connectivity to an RODC to which it can establish a connection (if it is not already connected) and send the request.
- The account that the user is using for the login must be present in the directory.
- The RODC must be configured to allow caching of the account credentials of a user.

Main Success Scenario

1. **Trigger:** The user is trying to log in to the client by entering his or her credentials.
2. The Client Application establishes a connection to the RODC. Windows Authentication Services present in the RODC authenticates the user using the supplied credentials ([MS-AUTHSOD] section 2).
3. The RODC verifies that the credentials supplied by the Client Application have the necessary access-control rights to complete the operation ([MS-ADTS] section 5.1.3).
4. If the user is successfully authenticated, the RODC sends a success response to the Client Application.
5. Upon successful verification of step3, the RODC sends a LogonTimeStampUpdatesForward request ([MS-SAMS] section 3.2.4.6) to the DC.
6. The DC updates the lastLogonTimeStamp attribute of the user account and sends a response.

Postconditions: The user account's lastLogonTimeStamp attribute is updated to reflect the user's last login time**.**

2.7.2.7 Query an Account's Group Membership - Client Application

In this use case, 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.

Goal: Retrieve an account's group membership.

Context of Use: An administrator wants to retrieve or use group membership of a directory object.

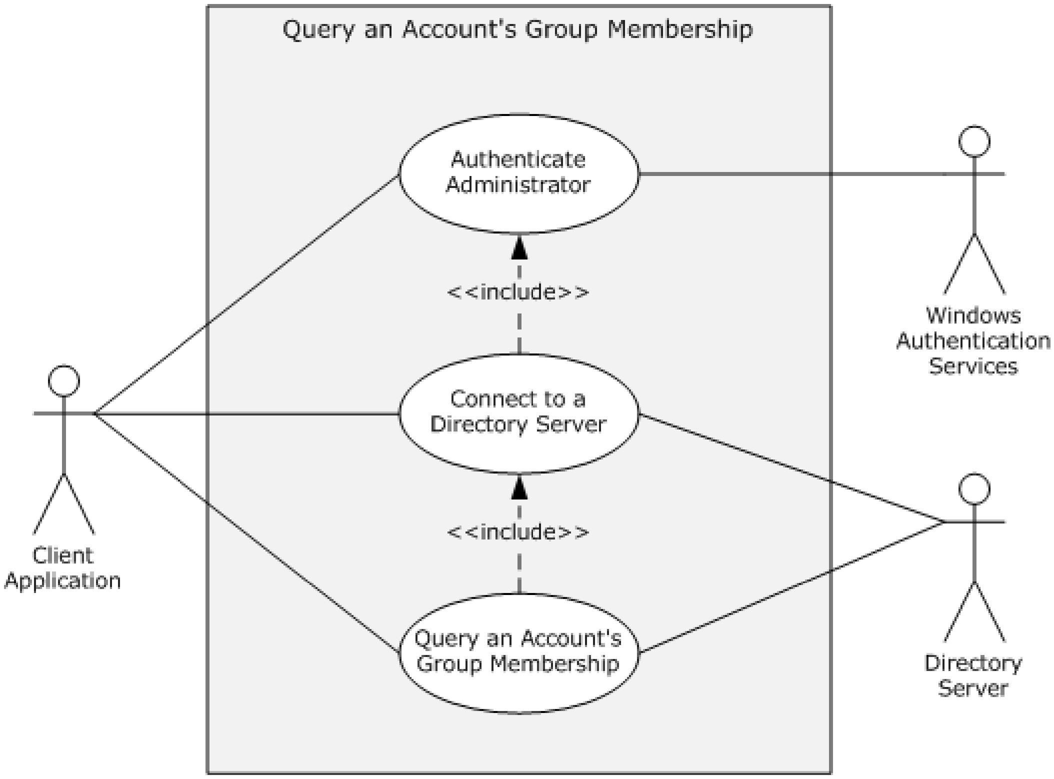

Figure 20: Use case diagram for querying the group membership of an account

Actors

- Client Application

  The Client Application is the primary actor. It is the entity that prepares the connection to the Directory Server, submits the request to obtain group membership, and relays the response to the administrator.

- Windows Authentication Services

  The Windows Authentication Services [MS-AUTHSOD] is the supporting actor that authenticates the administrator's identity. This is done so that access-control decisions can be made by the Active Directory System.

- Directory Server

  The Directory Server is the supporting actor that receives the request for group-membership information and gathers the information for the requestor.

Stakeholders

- Administrator

The administrator is the one who initiates operations such as create, reset, change, query group members, create security group, modify group member list, and delete on an account. The administrator is primarily interested in whether the operations are successfully completed and in receiving an error message otherwise.

- Directory

  The directory is the entity that contains and maintains group membership.

  In this operation, the directory is left unchanged.

Preconditions

- The system-wide preconditions described in section 2.6 must be satisfied. The Active Directory System must complete initialization, as described in section 2.6.
- 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 for which group membership is being requested must exist.

Main Success Scenario

1. **Trigger:** The administrator provides the account name of the account to query as input to the Client Application with credentials and invokes the operation that queries the group membership of an account.
2. The Client Application establishes a connection to the Directory Server. Windows Authentication Services authenticates the Client Application using the supplied credentials ([MS-AUTHSOD] section 2).
3. The Client Application sends a request to the Directory Server asking it to retrieve the group membership of the account.
4. The Directory Server verifies that the credentials supplied through the Client Application have the necessary access-control rights to complete the operation ([MS-ADTS] section 5.1.3).
5. The Directory Server sends a response to the Client Application that contains the group membership of the specified account.

Postconditions: Group membership information for the account is available to the Client Application**.**

Extensions

- If the credentials supplied through the Client Application have insufficient access-control rights to retrieve the group membership of the account:

  1-4. Same as Main Success Scenario.

  5. The Directory Server sends a response to the Client Application. Group membership information is not returned to the Client Application.

2.7.2.8 Delete an Account - Client Application

In this use case, 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.

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.

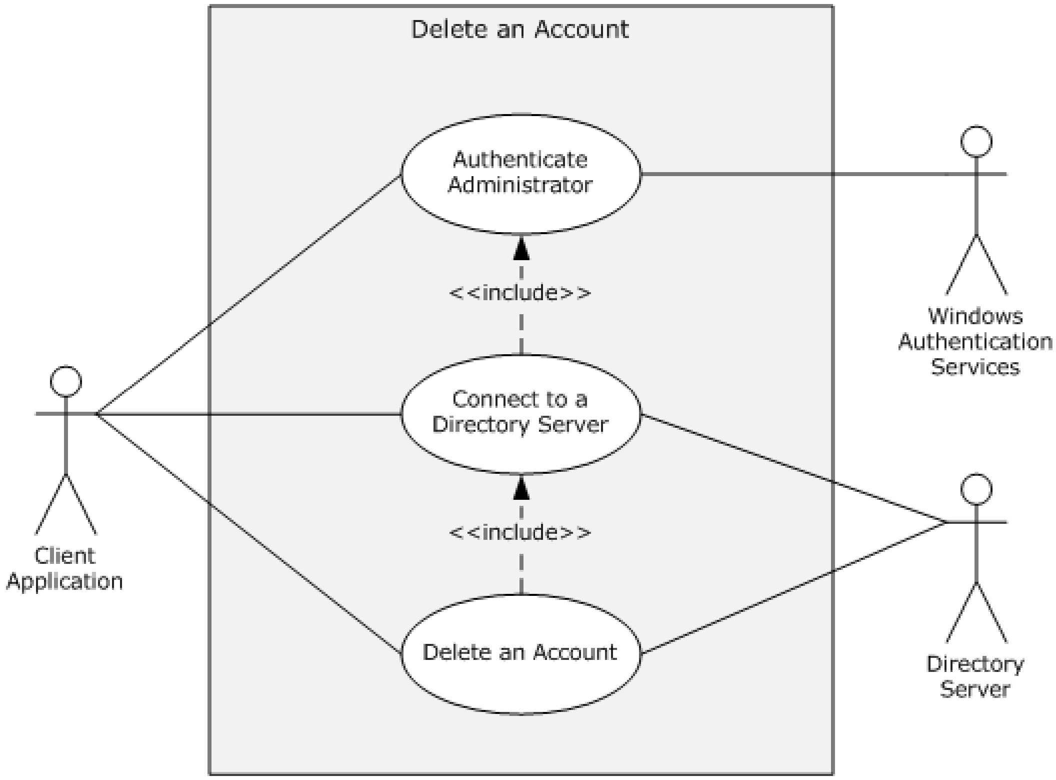

Figure 21: Use case diagram for deleting an account

Actors

- Client Application

  The Client Application is the primary actor. It is the entity that prepares the connection to the Directory Server, submits the request to delete an account, and relays the response to the administrator.

- Windows Authentication Services

  The Windows Authentication Services [MS-AUTHSOD] is the supporting actor that authenticates the administrator's identity. This is done so that access-control decisions can be made by the Active Directory System.

- Directory Server

  The Directory Server is the supporting actor that receives the deletion request and deletes the account from the directory.

Stakeholders

- Administrator

The administrator is the one who initiates operations such as create, reset, change, query group members, create security group, modify group member list, and delete on an account. The administrator is primarily interested in whether the operations are successfully completed and in receiving an error message otherwise.

- Directory

  The directory is the entity that contains the account being deleted.

Preconditions

- The system-wide preconditions described in section 2.6 must be satisfied. The Active Directory System must complete initialization, as described in section 2.6.
- 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 is being deleted must exist.

Main Success Scenario

1. **Trigger:** The administrator provides the account name of the account to be deleted as input to the Client Application with credentials and invokes the operation that deletes an account.
2. The Client Application establishes a connection to the Directory Server. Windows Authentication Services authenticates the Client Application using the supplied credentials ([MS-AUTHSOD] section 2).
3. The Client Application sends a request to the Directory Server asking it to delete the account.
4. The Directory Server verifies that the credentials supplied through the Client Application have the necessary access-control rights to complete the operation ([MS-ADTS] section 5.1.3).
5. The Directory Server deletes the object in the directory that represents the account with the account name supplied by the client. Additional processing tasks that are mandated by the server's processing rules and constraints might occur ([MS-ADTS] sections 3.1.1.5.1 and 3.1.1.5.3).
6. The Directory Server sends a response to the Client Application indicating that the account has been successfully deleted.

Postconditions: The account is no longer available.

Extensions

- If the credentials supplied through the Client Application have insufficient access-control rights to delete the account:

  1-4. Same as Main Success Scenario.

  5. The Directory Server sends a response to the Client Application indicating that the supplied credentials have insufficient access-control rights to delete the account.

2.7.2.9 Create a Security Group - Client Application

In this use case, 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.

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.

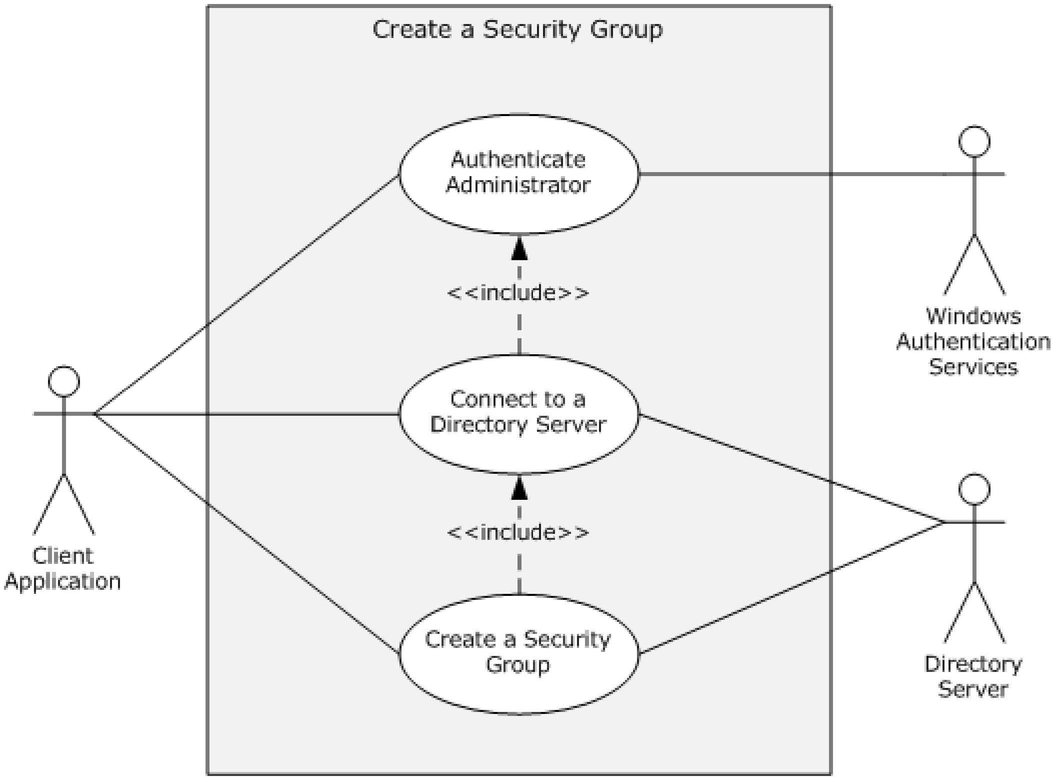

Figure 22: Use case diagram for creating a security group

Actors

- Client Application

  The Client Application is the primary actor. It is the entity that prepares the connection to the Directory Server, submits the request to create the security group, and relays the response to the administrator.

- Windows Authentication Services

  The Windows Authentication Services [MS-AUTHSOD] is the supporting actor that authenticates the administrator's identity. This is done so that access-control decisions can be made by the Active Directory System.

- Directory Server

  The Directory Server is the supporting actor that receives the creation request and creates the security group.

Stakeholders

- Administrator

The administrator is the one who initiates operations such as create, reset, change, query group members, create security group, modify group member list, and delete on an account. The administrator is primarily interested in whether the operations are successfully completed and in receiving an error message otherwise.

- Directory

  The directory is the entity that contains the security group being created.

Preconditions

- The system-wide preconditions described in section 2.6 must be satisfied. The Active Directory System must complete initialization, as described in section 2.6.
- 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.

Main Success Scenario

1. **Trigger:** The administrator provides the group name for the new security group as input to the Client Application with credentials and invokes the operation that creates a new security group.
2. The Client Application establishes a connection to the Directory Server. Windows Authentication Services authenticates the Client Application using the supplied credentials ([MS-AUTHSOD] section 2).
3. 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.
4. The Directory Server verifies that the credentials supplied through the Client Application have the necessary access-control rights to complete the operation ([MS-ADTS] section 5.1.3).
5. The Directory Server validates the constraints on the new group name, as described in [MS-SAMR] sections 3.1.1.6 and 3.1.1.8.4.
6. The Directory Server creates an object in the directory that represents 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 ([MS-ADTS] sections 3.1.1.5.1 and 3.1.1.5.2).
7. The Directory Server sends a response to the Client Application indicating that the new security group has been successfully created.

Postconditions: The new security group is created and ready for use.

Extensions

- If the credentials supplied through the Client Application have insufficient access-control rights to create the new security group:

  1-4. Same as Main Success Scenario.

  5. The Directory Server sends a response to the Client Application indicating that the supplied credentials have 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 described in [MS-SAMR] section 3.1.1.6:

1-5. Same as Main Success Scenario.

6. The Directory Server sends a response to the Client Application indicating that the specified group name does not meet the constraints.

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

1-5. Same as Main Success Scenario.

6. The Directory Server sends a response to the Client Application indicating that the specified group name is already in use by an existing group.

2.7.2.10 Modify Group Member List - Client Application

In this use case, an existing security group is used to control access to directory resources. An administrator wishes to modify the member list of that group so that a new account can access the controlled resources. The administrator launches a Client Application to modify the member list for an existing group. The Client Application establishes a connection to the Active Directory System.

Goal: Modify the member list of an existing group.

Context of Use: An administrator wishes to add or delete members to a security group.

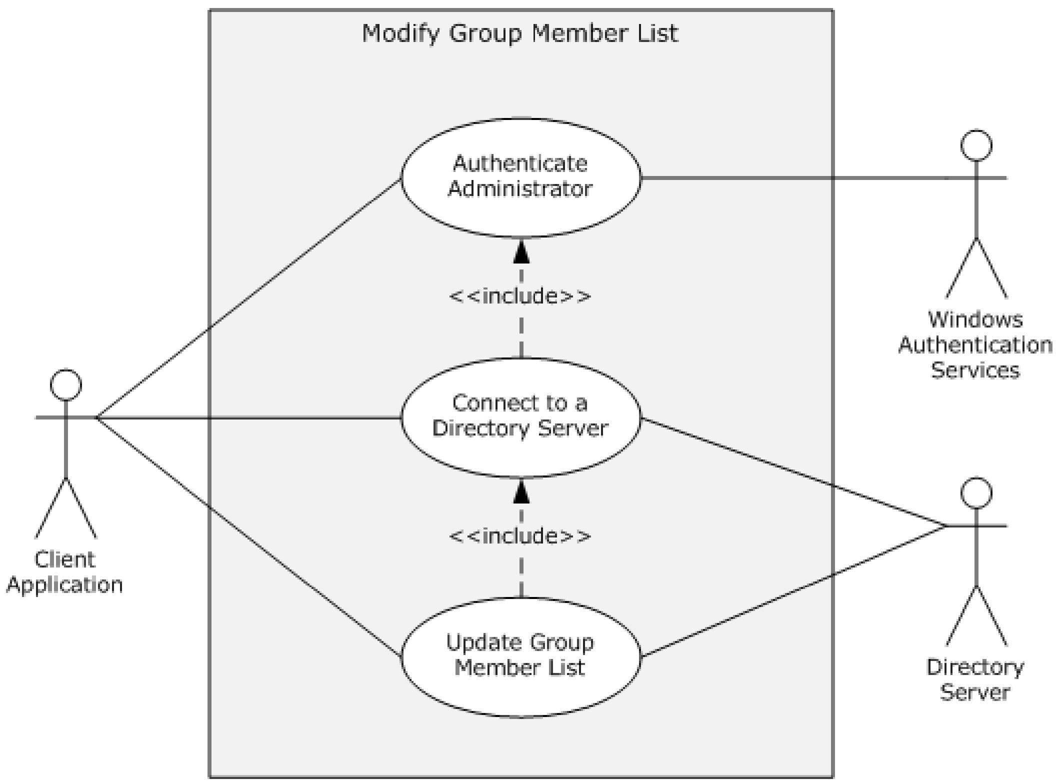

Figure 23: Use case diagram for modifying the member list of a group

Actors

- Client Application

The Client Application is the primary actor. It is the entity that prepares the connection to the Directory Server, submits the request to modify the member list of a group, and relays the response to the administrator.

- Windows Authentication Services

  The Windows Authentication Services [MS-AUTHSOD] is the supporting actor that authenticates the administrator's identity. This is done so that access-control decisions can be made by the Active Directory System.

- Directory Server

  The Directory Server is the supporting actor that receives the request and modifies the list.

Stakeholders

- Administrator

  The administrator is the one who initiates operations such as create, reset, change, query group members, create security group, modify group member list, and delete on an account. The administrator is primarily interested in whether the operations are successfully completed and in receiving an error message otherwise.

- Directory

  The directory is the entity that contains the list being modified.

Preconditions

- The system-wide preconditions described in section 2.6 must be satisfied. The Active Directory System must complete initialization, as described in section 2.6.
- 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 security group that is being modified must exist.

Main Success Scenario

1. **Trigger:** The administrator provides the group name for the group to be modified and the updates for the group's member list as input to the Client Application with credentials and invokes the operation that modifies the member list of a group.
2. The Client Application establishes a connection to the Directory Server. Windows Authentication Services authenticates the Client Application using the supplied credentials ([MS-AUTHSOD] section 2).
3. The Client Application sends a request to the Directory Server asking it to modify the member list of the specified group. The updates for the member list are included in the request.
4. The Directory Server verifies that the credentials supplied through the Client Application have the necessary access-control rights to complete the operation ([MS-ADTS] section 5.1.3).
5. The Directory Server verifies that the new member list satisfies the constraints described in [MS-SAMR] section 3.1.1.8.9.

6. The directory object that represents the group is modified with the new member list. Additional processing might occur, as described in [MS-ADTS] sections 3.1.1.5.1 and 3.1.1.5.3, and [MS-SAMR] section 3.1.1.8.9.
7. The Directory Server sends a response to the Client Application indicating that the member list has been modified.

Postconditions: The group's member list is modified.

Extensions

- If the credentials supplied through the Client Application have insufficient access-control rights to modify the member list of the group:

  1-4. Same as Main Success Scenario.

  5. The Directory Server sends a response to the Client Application indicating that the supplied credentials have insufficient access-control rights to modify the member list of the group.

- If the member list supplied through the Client Application does not satisfy the constraints ([MS-SAMR] section 3.1.1.8.9):

  1-5. Same as Main Success Scenario.

  6. The Directory Server sends a response to the Client Application indicating that the specified member list does not meet the constraints.

2.7.2.11 Query Members of a Group - Client Application

In this use case, 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.

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.

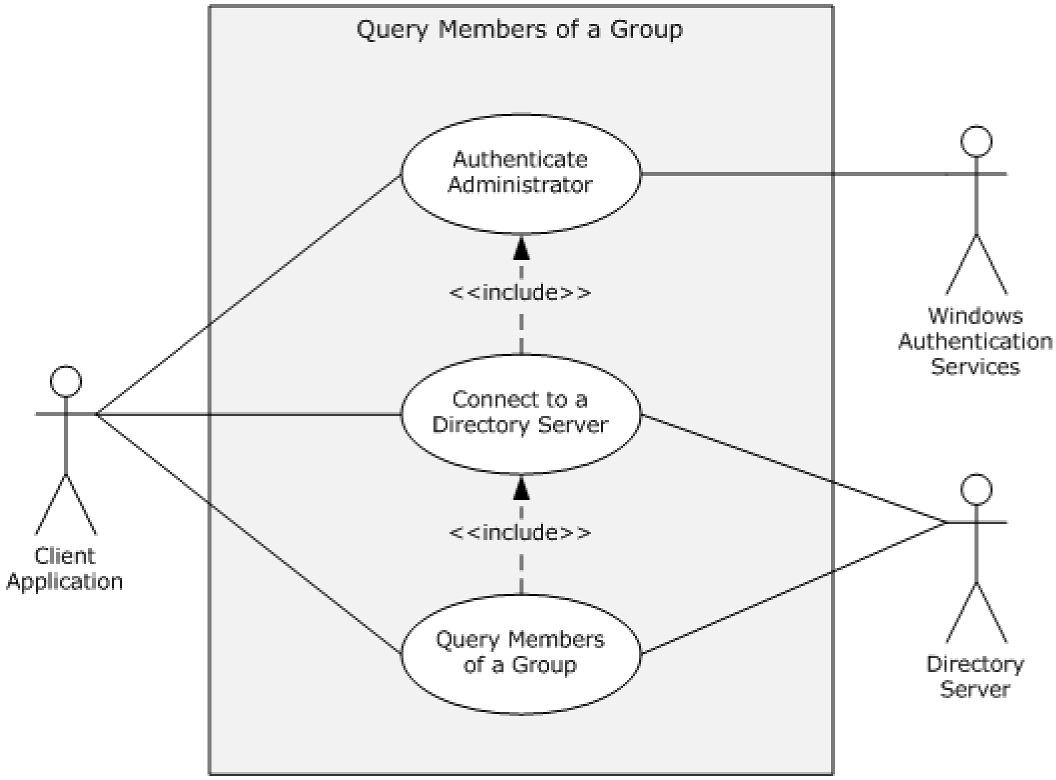

Figure 24: Use case diagram for querying the members of a group

Actors

- Client Application

  The Client Application is the primary actor. It is the entity that prepares the connection to the Directory Server, submits the request to retrieve a group's member list, and relays the response to the administrator.

- Windows Authentication Services

  The Windows Authentication Services [MS-AUTHSOD] is the supporting actor that authenticates the administrator's identity. This is done so that access-control decisions can be made by the Active Directory System.

- Directory Server

  The Directory Server is the supporting actor that receives the request for a group's member list and gathers the information for the requestor.

Stakeholders

- Administrator

  The administrator is the one who initiates operations such as create, reset, change, query group members, create security group, modify group member list, and delete on an account. The administrator is primarily interested in whether the operations are successfully completed and in receiving an error message otherwise.

- Directory

  The directory is the entity that contains and maintains group membership information.

  In this operation, the directory is left unchanged.

Preconditions

- The system-wide preconditions described in section 2.6 must be satisfied. The Active Directory System must complete initialization, as described in section 2.6.
- 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 group for which information is being sought must exist.

Main Success Scenario

1. **Trigger:** The administrator provides the group name of the group to be queried as input to the Client Application with credentials and invokes the operation that queries a group's member list.
2. The Client Application establishes a connection to the Directory Server. Windows Authentication Services authenticates the Client Application using the supplied credentials ([MS-AUTHSOD] section 2).
3. The Client Application sends a request to the Directory Server asking it to retrieve the member list of the group.
4. The Directory Server verifies that the credentials supplied through the Client Application have the necessary access-control rights to complete the operation ([MS-ADTS] section 5.1.3).
5. The Directory Server sends a response to the Client Application that contains the member list for the specified group.

Postconditions: The member list of the group is available to the Client Application.

Extensions

- If the credentials supplied through the Client Application have 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 to the Client Application. The member list of the group is not returned to the Client Application.

2.7.3 Schema Management

When the set of classes and attributes in the base Active Directory schema does 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 an existing class. Schema classes and attributes are objects stored in Active Directory. Adding a new class to the schema is equivalent to creating a new object of the classSchema class ([MS-ADTS] section 3.1.1.2.4.8); adding a new attribute to the schema is equivalent to creating a new object of the attributeSchema class ([MS-ADTS] section 3.1.1.2.3); adding an attribute to an existing class is done by modifying the corresponding classSchema object, which modifies the class definition to contain the attribute. After a new class or attribute is successfully added to the schema, users can create the objects of the newly defined or extended schema class.

The following use case diagram illustrates the use cases of schema management.

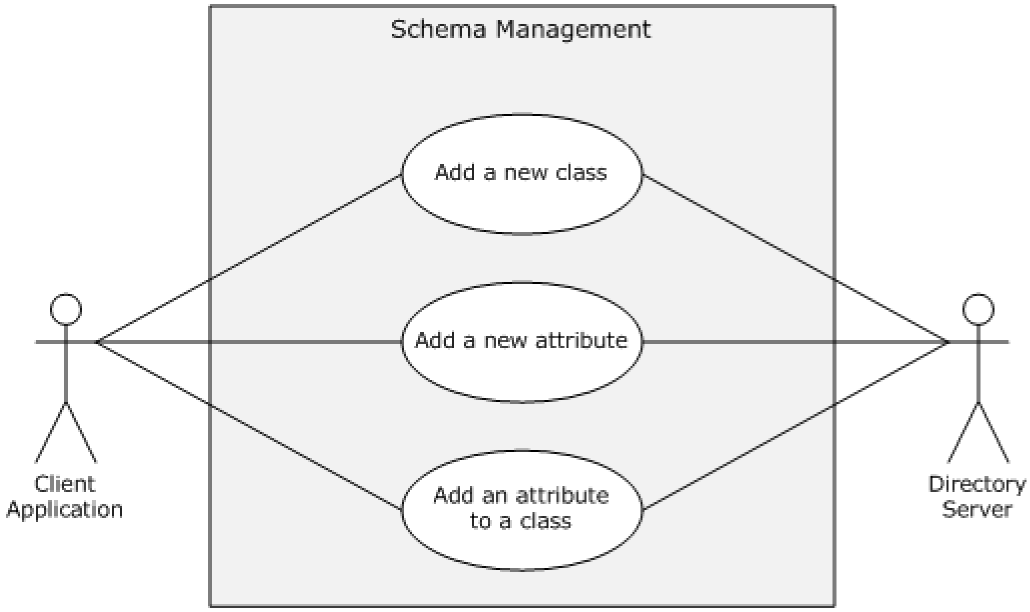

Figure 25: Use cases for schema management

2.7.3.1 Add a New Class to the Schema - Client Application

In this use case, the administrator realizes that the set of classes in the base Active Directory schema does not meet the needs of an application on the client. The administrator extends the schema by adding a new class to the schema; that is, by creating a new object of the classSchema class. After the new class is successfully added to the schema, the administrator can create objects of the newly defined class.

Goal: The Client Application 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.

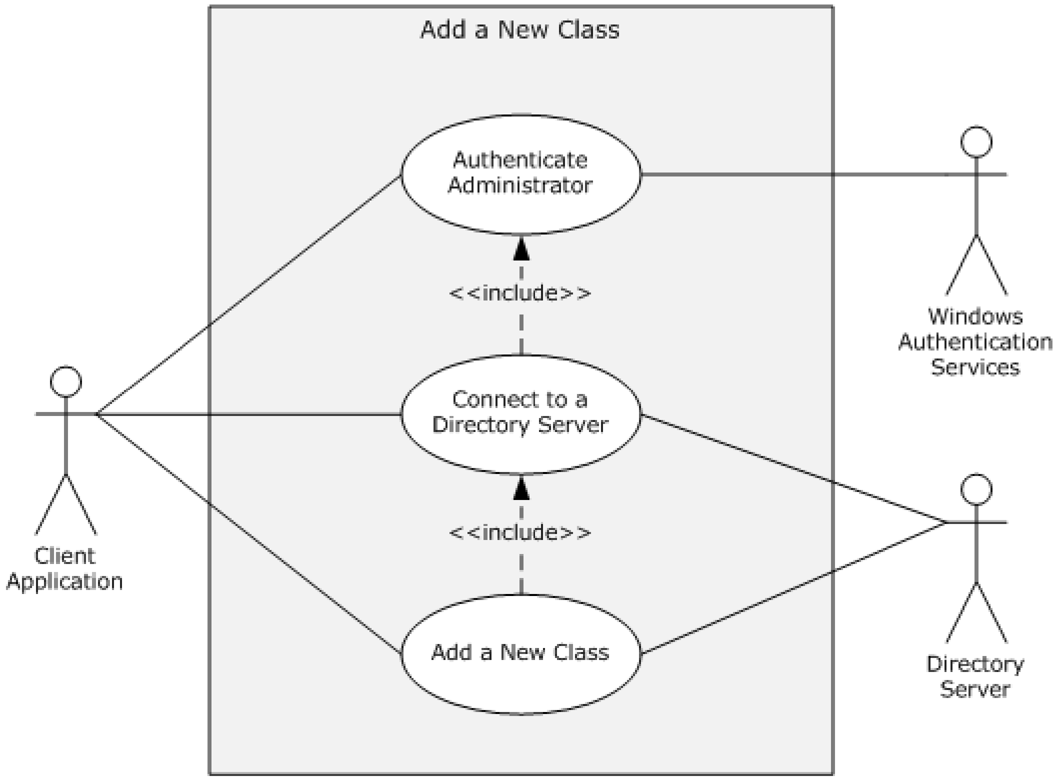

Figure 26: Use case diagram for adding a new class to the Active Directory schema

Actors

- Client Application

  The Client Application is the primary actor. It is the entity that prepares the connection to the Directory Server, submits the request to add a new class, and relays the response to the administrator.

- Windows Authentication Services

  The Windows Authentication Services [MS-AUTHSOD] is the supporting actor that authenticates the administrator's identity. This is done so that access-control decisions can be made by the Active Directory System.

- Directory Server

  The Directory Server is the supporting actor that receives the request and adds the new class.

Stakeholders

- Administrator

  The administrator is the one who initiates the addition of a new class to the schema. The administrator is primarily interested in whether the class was successfully added and in receiving an error message otherwise.

- Directory

The directory is the entity that contains the additional class.

Preconditions

- The system-wide preconditions described in section 2.6 must be satisfied. The Active Directory System must complete initialization, as described in section 2.6.
- 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.

Main Success Scenario

1. **Trigger:** The administrator provides the mandatory attributes ([MS-ADTS] section 3.1.1.2) for the new class as input to the Client Application with credentials and invokes the operation that adds a new class to the schema.
2. The Client Application establishes a connection to the Directory Server that owns the Schema Master **FSMO role** ([MS-ADTS] section 3.1.1.5.1.8). Windows Authentication Services authenticates the Client Application using the supplied credentials ([MS-AUTHSOD] section 2).
3. The Client Application sends a request to the 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.
4. The Directory Server verifies that the credentials supplied through the Client Application have the necessary access-control rights to complete the operation ([MS-ADTS] section 3.1.1.2.5).
5. The Directory Server verifies that it owns the Schema Master FSMO role ([MS-ADTS] section 3.1.1.2.5).
6. The Directory Server validates the constraints on the new class attributes (as outlined in [MS-ADTS] section 3.1.1.2.5).
7. The Directory Server creates an object in the directory that represents 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 ([MS-ADTS] sections 3.1.1.2.5, 3.1.1.5.1, and 3.1.1.5.2.)
8. The Directory Server sends a response to the Client Application indicating that the new class has been successfully added to the schema.

Postconditions: The new object of class classSchema is created and ready for use.

Extensions

- If the credentials supplied through the Client Application have insufficient access-control rights to add the new schema class:

  1-4. Same as Main Success Scenario.

  5. The Directory Server sends a response to the Client Application indicating that the supplied credentials have insufficient access-control rights to add the new class to the schema.

- 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):

  1-5. Same as Main Success Scenario.

6. The Directory Server sends a response to the Client Application with a referral to the Directory Server that does own the Schema Master FSMO role.

- If the class name supplied through the Client Application is not unique:

  1-6. Same as Main Success Scenario.

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

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

  1-6. Same as Main Success Scenario.

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

2.7.3.2 Add a New Attribute to the Schema - Client Application

In this use case, an administrator realizes that the set of attributes in the base Active Directory schema does not meet the needs of an application on the client. The administrator extends the schema by adding a new attribute to the schema; that is, by creating a new object of class attributeSchema. After the new attribute is successfully added to the schema, the administrator can then add the attribute to a class and hence create objects of that class with the new attribute.

Goal: The Client Application adds a new attribute to the schema of the Active Directory System.

Context of Use: When the set of attributes in the base Active Directory schema does not meet the needs of a Client Application.

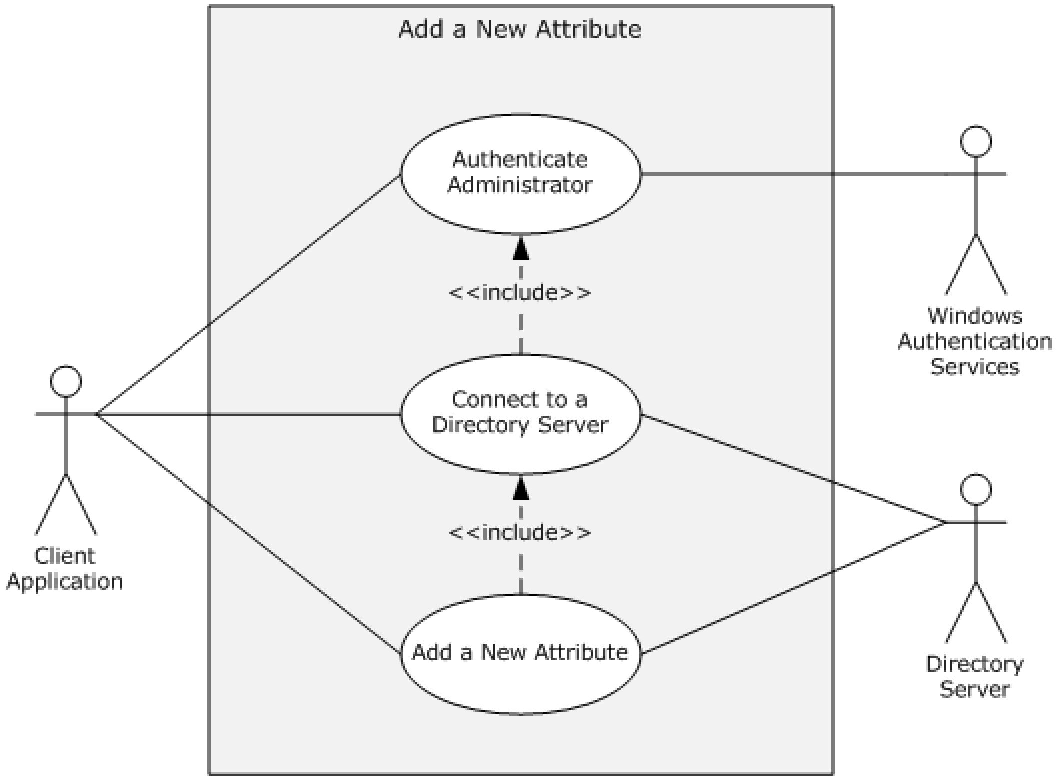

Figure 27: Use case diagram for adding a new attribute to the Active Directory schema

Actors

- Client Application

  The Client Application is the primary actor. It is the entity that prepares the connection to the Directory Server, submits the request to add a new attribute, and relays the response to the administrator.

- Windows Authentication Services

  The Windows Authentication Services [MS-AUTHSOD] is the supporting actor that authenticates the administrator's identity. This is done so that access-control decisions can be made by the Active Directory System.

- Directory Server

  The Directory Server is the supporting actor that receives the request and adds the new attribute.

Stakeholders

- Administrator

  The administrator is the one who initiates the addition of a new attribute to the schema. The administrator is primarily interested in whether the attribute was successfully added and in receiving an error message otherwise.

- Directory

  The directory is the entity that contains the additional attribute.

Preconditions

- The system-wide preconditions described in section 2.6 must be satisfied. The Active Directory System must complete initialization, as described in section 2.6.
- 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.

Main Success Scenario

1. **Trigger:** The administrator provides the mandatory attributes ([MS-ADTS] section 3.1.1.2) for the new object as input to the Client Application with credentials and invokes the operation that adds a new attribute to the schema.
2. The Client Application establishes a connection to the Directory Server that owns the Schema Master FSMO role ([MS-ADTS] section 3.1.1.5.1.8). Windows Authentication Services authenticates the Client Application by using the supplied credentials ([MS-AUTHSOD] section 2).
3. The administrator provides of class attributeSchema to the Client Application.
4. The Client Application sends a request to the 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 verifies that the credentials supplied through the Client Application have the necessary access-control rights to complete the operation ([MS-ADTS] section 3.1.1.2.5).
6. The Directory Server verifies that it owns the Schema Master FSMO role ([MS-ADTS] section 3.1.1.2.5).
7. The Directory Server validates the constraints on the new attributeSchema object attributes, as described in ([MS-ADTS] section 3.1.1.2.5).
8. The Directory Server creates an object of class attributeSchema in the directory that represents 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 ([MS-ADTS] sections 3.1.1.2.5, 3.1.1.5.1, and 3.1.1.5.2).
9. The Directory Server sends a response to the Client Application indicating that the new attribute has been successfully added to the schema.

Postconditions: The new object of class attributeSchema is created and ready for use.

Extensions

- If the credentials supplied through the Client Application have insufficient access-control rights to add the new attribute to the schema:

  1-5. Same as Main Success Scenario.

  6. The Directory Server sends a response to the Client Application indicating that the supplied credentials have insufficient access-control rights to add the new attribute to the schema.

- 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):

  1-6. Same as Main Success Scenario.

  7. The Directory Server sends a response to the Client Application with a referral to the Directory Server that does own the Schema Master FSMO role.

- If the attribute name supplied through the Client Application is not unique:

  1-7. Same as Main Success Scenario.

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

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

  1-7. Same as Main Success Scenario.

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

2.7.3.3 Add an Attribute to a Class - Client Application

In this use case, an existing class in the base Active Directory schema lacks an attribute needed by an application on the client. The administrator extends the schema by adding an attribute to the class. After the attribute is successfully added to the class, the administrator can create objects of the extended class with the newly added attribute.

Goal: The Client Application 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.

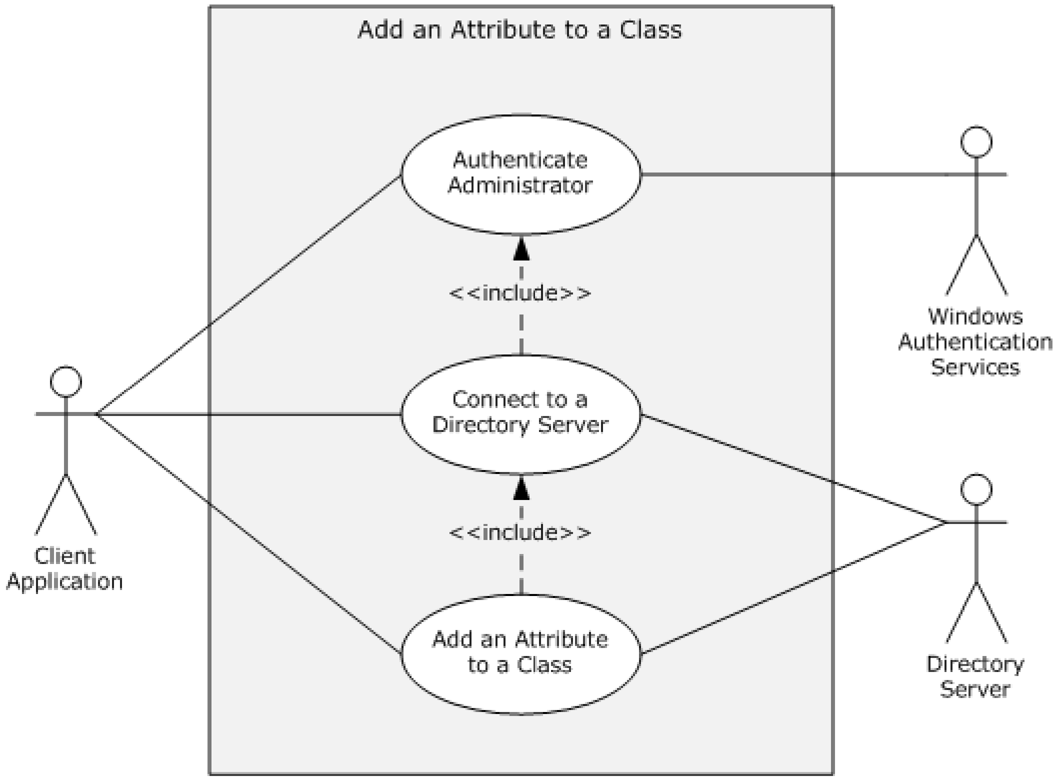

Figure 28: Use case diagram for adding an attribute to an existing class

Actors

- Client Application

  The Client Application is the primary actor. It is the entity that prepares the connection to the Directory Server, submits the request to add an attribute to a class, and relays the response to the administrator.

- Windows Authentication Services

  The Windows Authentication Services [MS-AUTHSOD] is the supporting actor that authenticates the administrator's identity. This is done so that access-control decisions can be made by the Active Directory System.

- Directory Server

  The Directory Server is the supporting actor that receives the request and adds the attribute to the class.

Stakeholders

- Administrator

  The administrator is the one who initiates the addition of an attribute to a class in the schema. The administrator is primarily interested in whether the attribute was successfully added to the class and in receiving an error message otherwise.

- Directory

  The directory is the entity that contains the attribute and the class.

Preconditions

- The system-wide preconditions described in section 2.6 must be satisfied. The Active Directory System must complete initialization, as described in section 2.6.
- 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.
- The attribute and the class must exist in the directory schema.

Main Success Scenario

1. **Trigger:** The administrator provides the attribute name and the class name as input to the Client Application with credentials and invokes the operation that adds an attribute to a class.
2. The Client Application establishes a connection to the Directory Server that owns the Schema Master FSMO role ([MS-ADTS] section 3.1.1.5.1.8). Windows Authentication Services authenticates the Client Application that is using the supplied credentials ([MS-AUTHSOD] section 2).
3. The Client Application sends a request to the Directory Server asking it to add an attribute to a class in the directory schema and specifies the attribute name and the class name for this operation.
4. The Directory Server verifies that the credentials supplied through the Client Application have the necessary access-control rights to complete the operation ([MS-ADTS] section 3.1.1.2.5).
5. The Directory Server verifies that it owns the Schema Master FSMO role ([MS-ADTS] section 3.1.1.2.5).
6. The Directory Server validates the constraints on the attribute and the class, as outlined in ([MS-ADTS] section 3.1.1.2.5).
7. The Directory Server adds the attribute to the class.
8. The Directory Server sends a response to the Client Application indicating that the attribute has been successfully added to the class.

Postconditions: The class has been updated with the specified attribute.

Extensions

- If the credentials supplied through the Client Application have insufficient access-control rights to modify the class:

  1-4. Same as Main Success Scenario.

  5. The Directory Server sends a response to the Client Application indicating that the supplied credentials have 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):

  1-5. Same as Main Success Scenario.

6. The Directory Server sends a response to the Client Application with a referral to the Directory Server that does own the Schema Master FSMO role.

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

  1-6. Same as Main Success Scenario.

  7. The Directory Server sends a response 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:

  1-6. Same as Main Success Scenario.

  7. The Directory Server sends a response to the Client Application indicating that the value to be added already exists.

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

  1-6. Same as Main Success Scenario.

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

2.7.4 Name Translation

The use case in this category represents name translation between a directory object's SID and human-readable names of the security principals in the access-control entries (ACEs) that secure the object. Name translation can be used in tasks such as determining how an object is secured in the directory, or modifying the way in which it is secured.

The following use case diagram illustrates the use case of name translation.

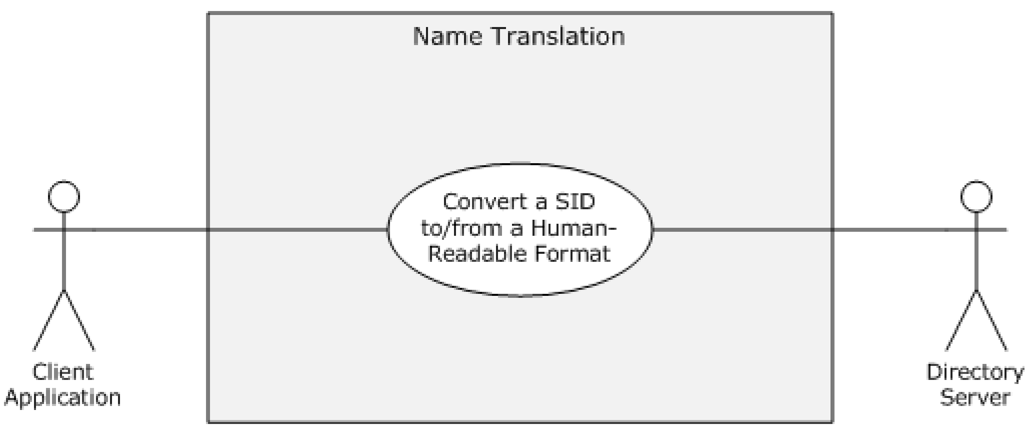

Figure 29: Use case for name translation

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

Note This use case is applicable only to Active Directory Domain Services (AD DS); it is not applicable to Active Directory Lightweight Directory services (AD LDS).

This use case describes the translation between the machine-readable and human-readable forms of names.

When an administrator wishes to investigate and maintain the security of a directory object, translation between the machine-readable and human-readable forms of names might be required. This translation allows the administrator, via a Client Application, to secure access to a directory object without having to understand machine-readable names. The Client Application displays the human-readable names of the security principals in the ACEs that secure the object, which have been translated from SIDs. The administrator specifies human-readable names of security principals when securing the object, which are translated to SIDs.

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.

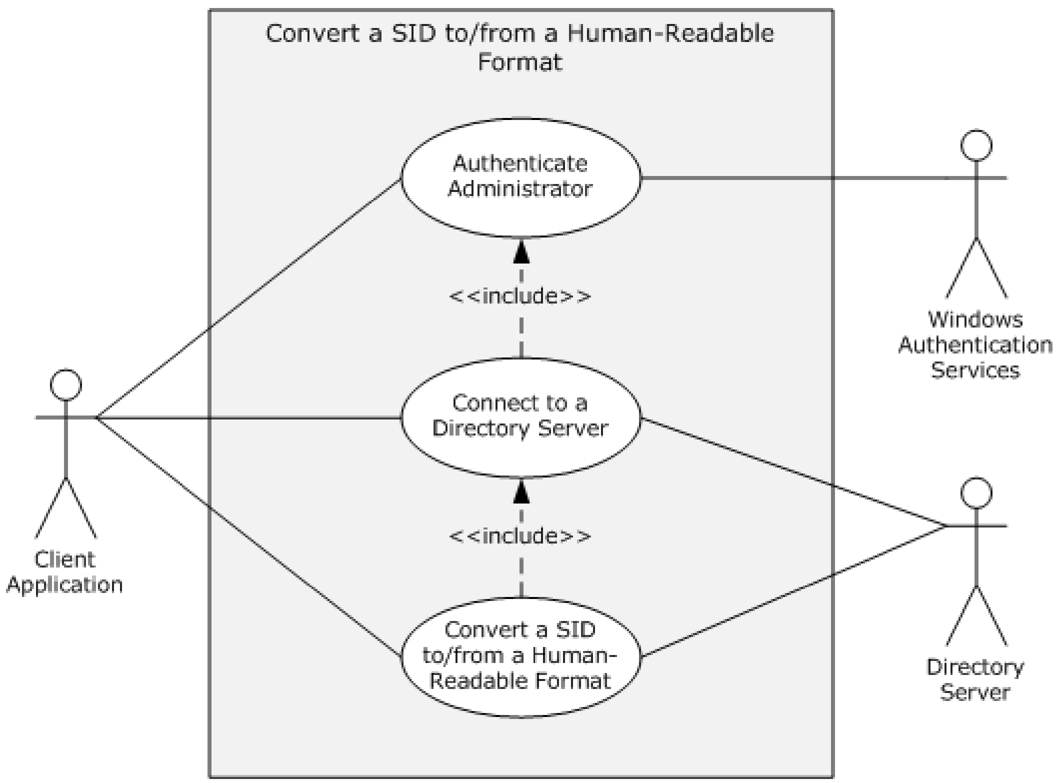

Figure 30: Use case diagram for converting a SID to or from a human-readable format

Actors

- Client Application

  The Client Application is the primary actor. It is the entity that prepares the connection to the Directory Server, submits the request for name translation, and returns the result to the administrator.

- Windows Authentication Services

The Windows Authentication Services [MS-AUTHSOD] is the supporting actor that authenticates the administrator's identity. This is done so that access-control decisions can be made by the Active Directory System.

- Directory Server

  The Directory Server is the supporting actor that receives the translation request and performs the actual translation.

Stakeholders

- Administrator

  The administrator is the one who performs security actions that trigger the need for a name translation. The administrator is primarily interested in reading and giving human-readable names and in not having to understand machine-readable names.

- Directory

  The directory is the entity that contains the objects being considered by the administrator.

Preconditions

- The system-wide preconditions described in section 2.6 must be satisfied. The Active Directory System must complete initialization, as described in section 2.6.
- The application must have network connectivity to a directory server that meets the requirements described in section 2.5, to which it can establish a connection (if it is not already connected) and send the request.

Main Success Scenario

1. **Trigger:** Following an action by the administrator, the Client Application needs to display human-readable names that correspond to the SIDs in the **access control lists (ACLs)** that secure the object with which the administrator is interacting. Alternatively, the administrator provides a human-readable name to the Client Application with credentials to set an ACL on an object, and the Client Application needs to retrieve the SID of the object corresponding to that name in order to do so.
2. The Client Application establishes a connection to the Directory Server. Windows Authentication Services authenticates the Client Application using the supplied credentials ([MS-AUTHSOD] section 2).
3. The Client Application sends a request to the Directory Server to perform name translation between the SID and the human-readable name of directory objects of interest to the administrator.
4. The Directory Server identifies the directory objects for which name translation needs to be performed.
5. From the set of directory objects so identified, the Directory Server obtains their names in the requested name format.
6. The Directory Server sends a response to the client containing the names in the requested format.

Postconditions: The translated information and is available to the Client Application.

Extensions

- The SID or the name supplied through the Client Application is misformatted, as defined in [MS-DTYP] section 2.4.2.3 and [MS-LSAT] section 3.1.4.5, respectively:

  1-4. Same as Main Success Scenario.

  5. The Directory Server sends a response to the Client Application indicating that an invalid parameter was used in the request, as indicated in [MS-LSAT] sections 3.1.4.5 and 3.1.4.9.

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

  1-4. Same as Main Success Scenario.

  5. The Directory Server sends a response 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:

  1-4. Same as Main Success Scenario.

  5. The Directory Server sends a response 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:

  1-4. Same as Main Success Scenario.

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

2.7.5 Directory Replication

The use cases in this category represent replication of directory data that is maintained by the Active Directory System. Domain and forest data must be replicated among disparate physical storage devices to maintain the integrity of the abstract logical structure of the directory. Replication can happen at various levels within a directory and under varying circumstances. Consider the following examples.

- If copies of domain data, or replicas, exist on multiple servers within the domain, that is, if there are multiple domain controllers (DCs) within the domain, new objects and modifications to existing objects are replicated to all of the domain's DCs using normal replication cycles.
- A forest typically consists of multiple domains, each of which is maintained by one or more domain controllers. Some of the data stored in the directory is considered to be forest-wide data, which must be replicated to all of the domain controllers in the forest.
- Some of the values that are stored in the directory, such as user password and account status, are of such a nature that propagation of changes in these values is time-critical. Replication of

these values must happen outside of normal replication cycles to allow for rapid propagation of the changes throughout the directory.

The following use case diagram illustrates the use cases of Directory Replication that are described in this document.

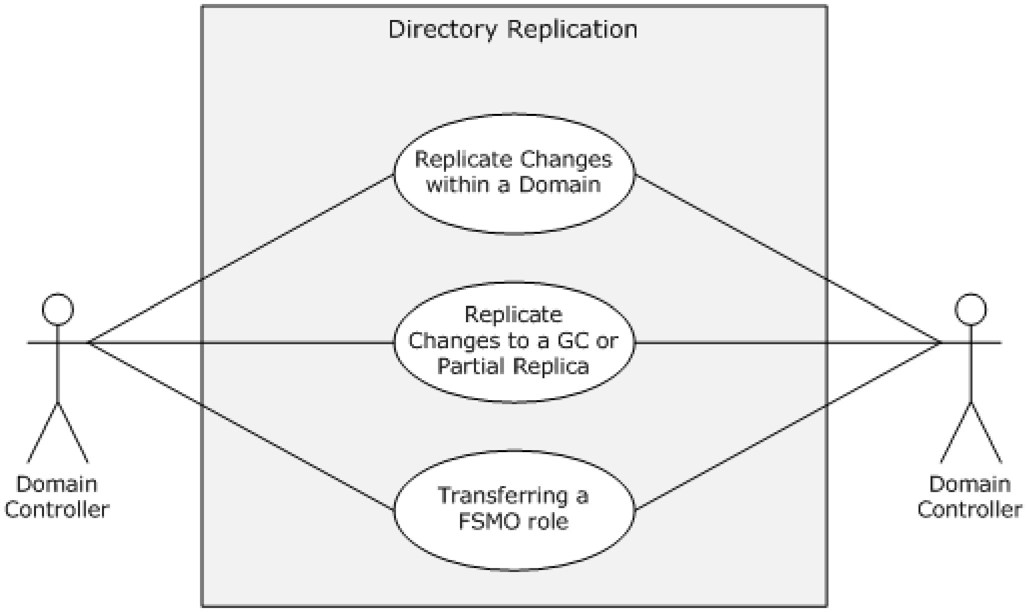

Figure 31: Use cases for directory replication

2.7.5.1 Replicate Changes Within a Domain - Domain Controller

In this use case, a domain is maintained by three domain controllers; that is, copies of the domain data, or replicas, exist on three domain controllers within the domain: Domain Controller 1 (DC1), Domain Controller 2 (DC2), and Domain Controller 3 (DC3). Changes to the data have been implemented on DC1; that is, DC1 has originating updates, and those changes need to be replicated to DC2 and DC3 using a normal, intra-site replication cycle. This replication tropology between DC1, DC2, and DC3 is formed by the Knowledge Consistency Checker (KCC) based on administrator-assigned costs as described in section 1 and [MS-ADTS] section 6.2.

Goal: Replicate originating updates that occurred on one domain controller to the other domain controllers in the domain.

Context of use: Changes need to be replicated within a domain to keep data consistent among domain controllers that store same directory partitions. This scenario works in a replication topology built by the KCC as mentioned in section 1 and [MS-ADTS] section 6.2.

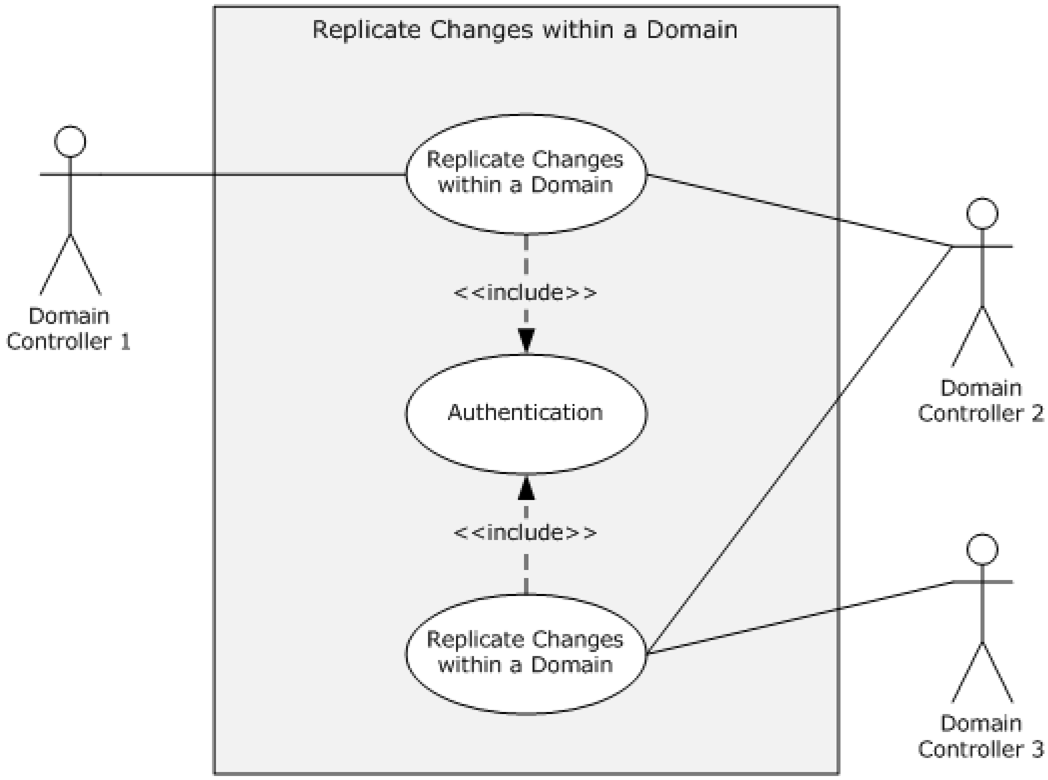

Figure 32: Use case diagram for replicating changes in domain data

Actors

- Domain Controller 1 (DC1)

  DC1 is the primary actor and has originating updates to its domain data.

- Domain Controller 2 (DC2)

  DC2 is a supporting actor that has not yet received the originating updates applied to DC1. DC2 is a replication partner of DC1.

- Domain Controller 3 (DC3)

  DC3 is a supporting actor that has not yet received the originating updates applied to DC1. DC3 is a replication partner of DC2.

Stakeholders

- Domain Administrators and Applications

  Domain administrators and applications are the entities that change attribute values in domain data.

- Domain Users

  Domain users are the people who depend on the information that is stored in the directory.

For all of these entities, the Active Directory System propagates all changes to domain data to all domain controllers in the domain so that a consistent view of the directory is maintained for all clients, regardless of which domain controller they communicate with.

Preconditions

- The environment described in section 2.5 is in place and the system-wide preconditions described in section 2.6 are satisfied. The Active Directory System must complete initialization, as described in section 2.6.
- The Knowledge Consistency Checker has created the replication topology of the domain; that is, the physical connections among the domain controllers of the domain that are used for replication traffic.

Main Success Scenario

Note This Main Success Scenario has no dependency on replication requests between DC1 to DC2 and DC2 to DC3. The order in this scenario has been chosen to show how the changes are replicated from DC1 to DC2 and DC3. This scenario covers scheduled replication, which is described in section 1 and [MS-ADTS] section 3.1.1.1.14.

1. **Trigger:** A domain administrator changes an attribute's value for an object in the domain data, the change manifesting as an originating update on DC1.
2. DC2 identifies its replication partner as DC1.
3. DC1 and DC2 perform **mutual authentication** using Kerberos.
4. DC2 sends a request to DC1 periodically to obtain the new value.
5. DC2 applies the change to its replica.
6. DC3 then identifies its replication partner as DC2.
7. DC2 and DC3 perform mutual authentication using Kerberos.
8. DC3 sends a request to DC2 periodically to obtain the new value.
9. DC3 applies the change to its replica.

Postconditions: The change to the attribute's value is replicated throughout the domain.

Extensions: None

2.7.5.2 Replicate Changes to a GC or a Partial Replica by Using RPC - Domain Controller

In this use case, replication of changes is performed periodically between a domain controller that is a full replica and a GC or a partial replica in another domain by using RPC.

Goal: Replicate changes to a GC or partial replica.

Context of use: Changes need to be replicated between two domain controllers, where one is a full replica and the other is a GC or a partial replica in another domain. This scenario works in a replication topology built by the KCC as mentioned in section 1 and [MS-ADTS] section 6.2.

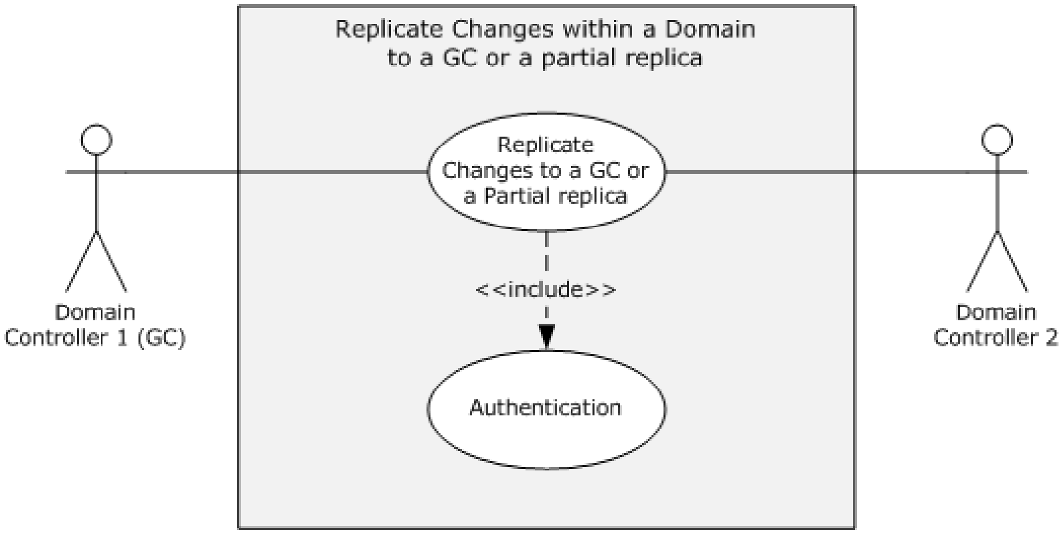

Figure 33: Use case diagram for replicating changes in a full replica to a GC or a partial replica

Actors

- Domain Controller 1 (GC)

  Domain Controller 1, or DC1, is the primary actor that has not yet received the originating updates applied to DC2. DC1 is a GC or partial replica and a replication partner of DC2.

- Domain Controller 2 (DC2)

  DC2 is the supporting actor that has originating updates to its domain data.

Stakeholders

- Domain Administrators and Applications

  Domain administrators and applications are the entities that change attribute values in domain data.

Preconditions

- The environment described in section 2.5 is in place and the system-wide preconditions described in section 2.6 are satisfied. The Active Directory System must complete initialization, as described in section 2.6.
- The Knowledge Consistency Checker (KCC) has created the replication topology of the domain; that is, the physical connections among the domain controllers of the domain that are used for replication traffic.
- DC1 and DC2 are in different domains.
- DC1 has a partial replica of the domain in which DC2 resides.

Main Success Scenario

1. **Trigger:** A domain administrator changes an attribute's value for an object in the domain data, the change manifesting as an originating update on DC2.
2. Depending on the replication interval calculated by the KCC ([MS-ADTS] section 6.2), DC1 sends a request to DC2 to obtain new values of only the attributes of the objects that are present in the DC1 partial replica.
3. DC2 responds to DC1 with the new values.
4. DC1 applies the changes to its replica.

Postcondition: Replication changes to the attributes present in partial replica are updated to the new values present in full replica.

Extensions: None.

Variation: Replicate changes to a GC or a partial replica by using SMTP

All the details in the preceding scenario are the same except that there are additional preconditions as follows:

- DC1 and DC2 are in different sites ([MS-SRPL] section 1.3).
- The domain controllers require the ability to send and receive SMTP messages using any SMTP mail transfer agent (as specified in [RFC2821]).
- There are additional conditions that the configurations of the domain controllers must meet before the DRS Protocol Extensions for SMTP can be used to replicate state between the domain controllers, as described in [MS-SRPL] sections 1.5 and 3.1.3.

2.7.5.3 Transferring a FSMO Role - Domain Controller

This use case describes the transfer of a FSMO role of one domain controller to another domain controller.

Goal: Transfer a FSMO role owned by one domain controller to another domain controller.

Context of use: Used to transfer a FSMO role when a domain controller is being demoted or a domain administrator is initiating the transfer of FSMO roles.

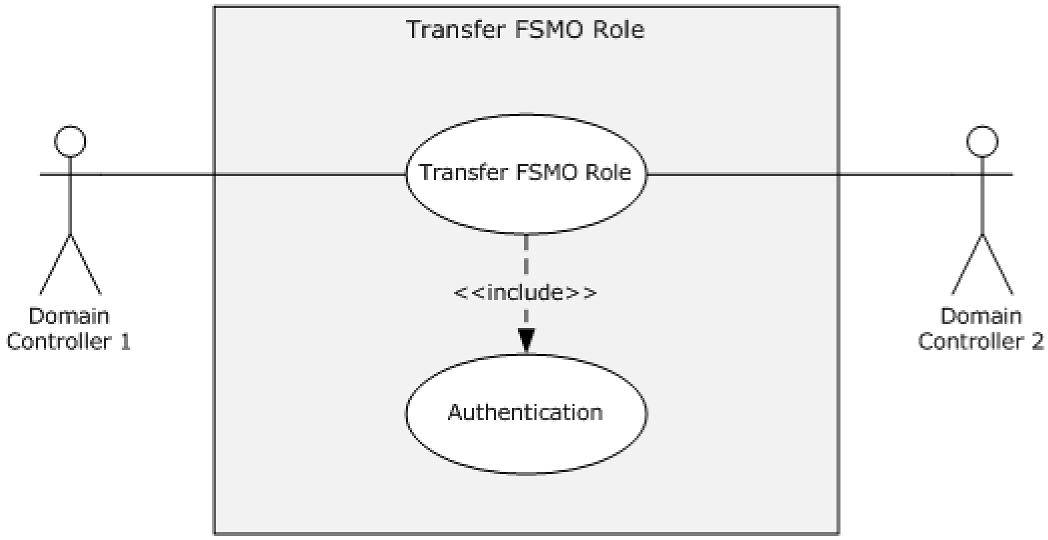

Figure 34: Use case diagram for transferring a FSMO role

Actors

- Domain Controller 1 (DC1)

  DC1 is the primary actor that is shutting down or being demoted by an administrator.

- Domain Controller 2 (DC2)

  DC2 is the supporting actor to which the FSMO role is being transferred.

Stakeholders

- Domain Administrators and Applications

  Domain administrators and applications are the entities that trigger transfer of FSMO roles.

Preconditions

- The environment described in section 2.5 is in place and the system-wide preconditions described in section 2.6 are satisfied. The Active Directory System must complete initialization, as described in section 2.6.

- DC1 is the owner of the FSMO role being transferred to DC2.

Main Success Scenario

1. **Trigger:** Transfer of the FSMO role is triggered when DC1 is demoted or upon domain administrator initiation.

2. DC1 identifies the new FSMO role owner to be DC2 either arbitrarily in the case of demotion or by administrator input.

3. DC1 and DC2 perform **mutual authentication** using Kerberos.

4. DC1 sends a rootDSE modify message ([MS-ADTS] section 3.1.1.3.3) to DC2, requesting that DC2 assume ownership of the FSMO role.

5. DC2 determines the current owner of the FSMO role, which in this case is DC1, and invokes an extended operation request ([MS-DRSR] section 4.1.10.4.3) against DC1 to request transfer of the FSMO role ownership.

6. DC1 transfers its FSMO role to DC2 through the transfer request.

Postconditions: The FSMO role is transferred to another domain controller.

Extensions: None.

2.7.6 Trust Management

These use cases cover trust creation between domains or forests, along with trust validation. A domain trust is created by creating a trusted domain object (TDO) as described in [MS-ADTS] section 6.1.6. These TDOs are managed and created as specified in [MS-LSAD] section 3.1.4.7. Domain trusts are validated or managed by the NRPC protocol as specified in [MS-NRPC] section 3.5.4.7.

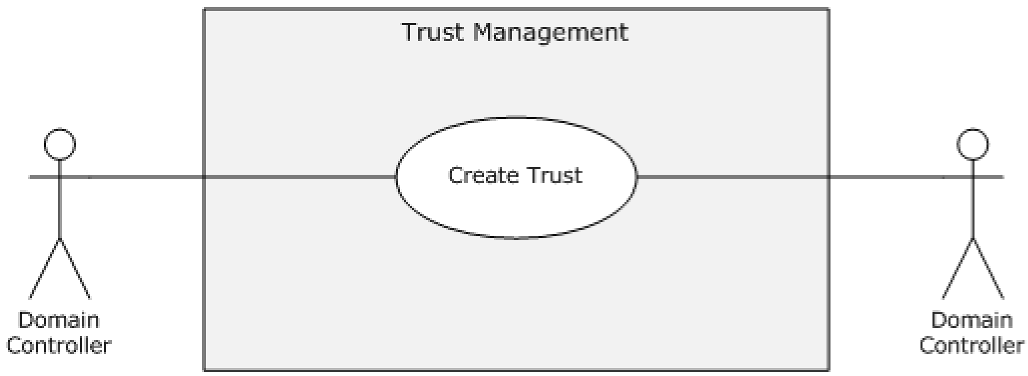

Figure 35: Use cases for trust management

2.7.6.1 Create a Trust - Domain Controller

In this use case, a trust is created between two domains or forests.

Goal: Create a trust between domains or forests for allowing access of resources between them.

Context of use: To create a trust between domains or forests in order to access resources between them.

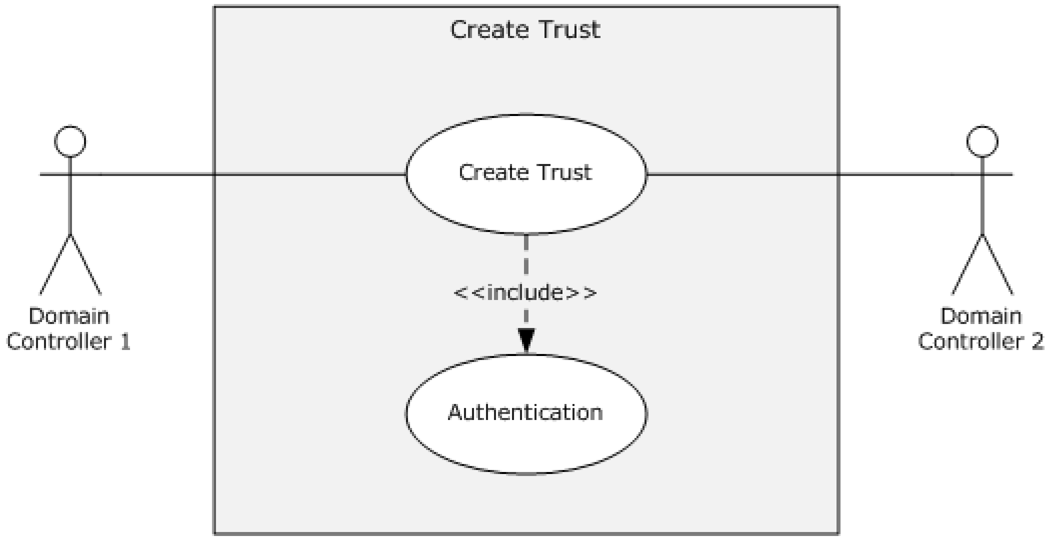

Figure 36: Use case diagram for creating a trust between domains or forests

Actors

- Domain Controller 1 (DC1)

  DC1 is the primary actor that is in some domain.

- Domain Controller 2 (DC2)

  DC2 is the supporting actor that is in a different domain. It can be in the same forest or in a different forest.

Stakeholders

- Users

  Users are people or other entities belonging to one domain who need to access directory resources in the other domain.

Preconditions

- The environment described in section 2.5 is in place and the system-wide preconditions described in section 2.6 are satisfied. The Active Directory System must complete initialization, as described in section 2.6.
- DC1 and DC2 are in different domains but can be in the same forest or different forests.

Main Success Scenario

1. **Trigger:** An application triggers trust creation while creating a trust between domain controllers that belong to different domains.
2. DC1 and DC2 perform **mutual authentication** using Kerberos.
3. DC1 sends a request for creating a trusted domain object (TDO) to DC2.
4. The TDO is created in DC2.

Postconditions: The trust is created between the two domains.

Extensions: None.

2.7.7 Domain Services

The use cases in this category pertain to the interaction between domain clients and those servers that service requests from the domain client to participate in domain activities. The domain activities that are relevant to this discussion include locating a domain controller, joining a domain, and unjoining from a domain.

The following use case diagram illustrates the use cases pertaining to domain-join activities.

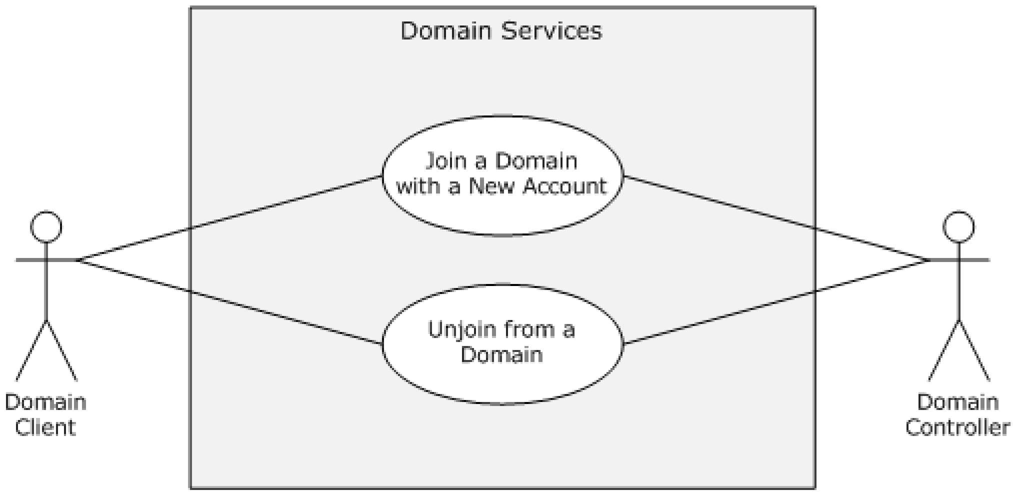

Figure 37: Use cases for domain services

2.7.7.1 Join a Domain with a New Account - Domain Client

This use case covers the general case of joining a domain with a new account. A new account can be created in the domain using either SAMR or LDAP. See sections 3.1.2 and 3.1.3 for details.

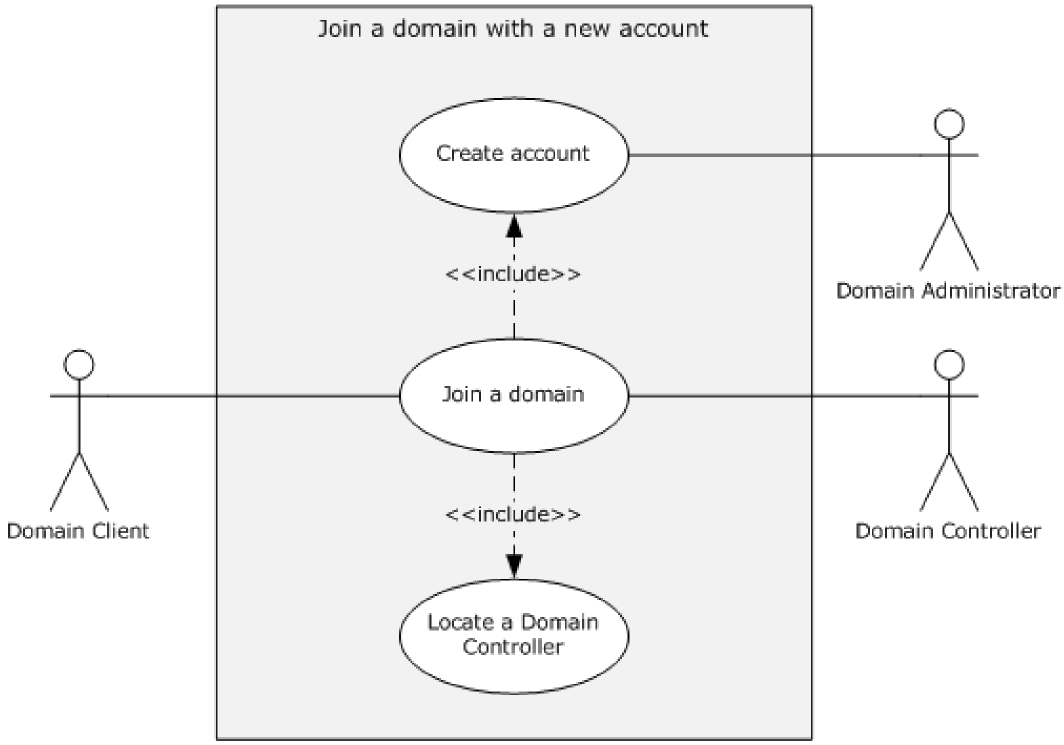

Figure 38: Join a domain by creating a new account

Goal: Join a Domain Client to a domain by creating a new account for the Domain Client in the domain.

Context of Use: This task is invoked by the domain-client administrator in order to enable the Domain Client to access the services and resources in a domain, as well as to provide domain members access to the Domain Client.

Actors

- Domain Client

  The Domain Client is the primary actor. It is the entity that locates and connects to the Domain Controller and is joined to the domain.

- Domain Controller

  The Domain Controller is the supporting actor that advertises its capabilities, responds to domain-join inquiries, and ultimately joins the Domain Client to the domain.

- Domain Administrator

  The Domain Administrator is the supporting actor that enables the Domain Client, using the credentials of the Domain Administrator, to open a secure connection to the Domain Controller.

Stakeholders

- End User

The end user is the person who wants to join a Domain Client to a domain so that he or she can access resources within the domain.

The primary interest of the end user is in whether the Domain Client was joined to the domain.

- Client Administrator

  The client administrator is the person who initiates the domain-join process on the Domain Client.

  The client administrator is primarily interested in whether the Domain Client was successfully joined to the domain and in receiving an error otherwise.

Preconditions: The credentials of an administrator of the domain who can create machine accounts in the domain are available to the client administrator.

Main Success Scenario

1. **Trigger**: This use case is triggered by the client administrator to join the client computer to a domain.
2. The Domain Client locates a Domain Controller using the Locate a Domain Controller use case (see section 2.7.7.3.1).
3. The Domain Client opens a secure connection to the Domain Controller using the Domain Administrator's supplied credentials.
4. The Domain Client retrieves domain information
5. The Domain Client uses the Domain Administrator's Credentials to set up an account for itself in the domain.
6. The Domain Client determines the trusted domains.
7. The Domain Client updates the client account in the domain.
8. The Domain Client updates the local client state.
9. The Domain Client reinitializes local protocols.

Postconditions: The Domain Client is joined to the domain.

Extensions: None.

Variation - Join a Domain with a new account created via LDAP

All details are identical to those of the main success scenario except the steps 3-5, which are replaced with the following steps:

3. The Domain Client connects to the LDAP server on the Domain Controller and performs a bind to establish a secure LDAP connection using the Domain Administrator's credentials.

4. The Domain Client retrieves domain information.

5. The Domain Client uses LDAP to create an account in the domain for itself.

2.7.7.2 Unjoin from the Domain - Domain Client

In this use case, a client administrator wants to unjoin a Domain Client from the domain that it is currently part of, usually to repurpose or decommission the client computer.

Goal: Unjoin a Domain Client from the domain.

Context of Use: This task is invoked by the domain-client administrator in order to enable the Domain Client to unjoin from the domain so that its resources are not accessed by the members of the domain.

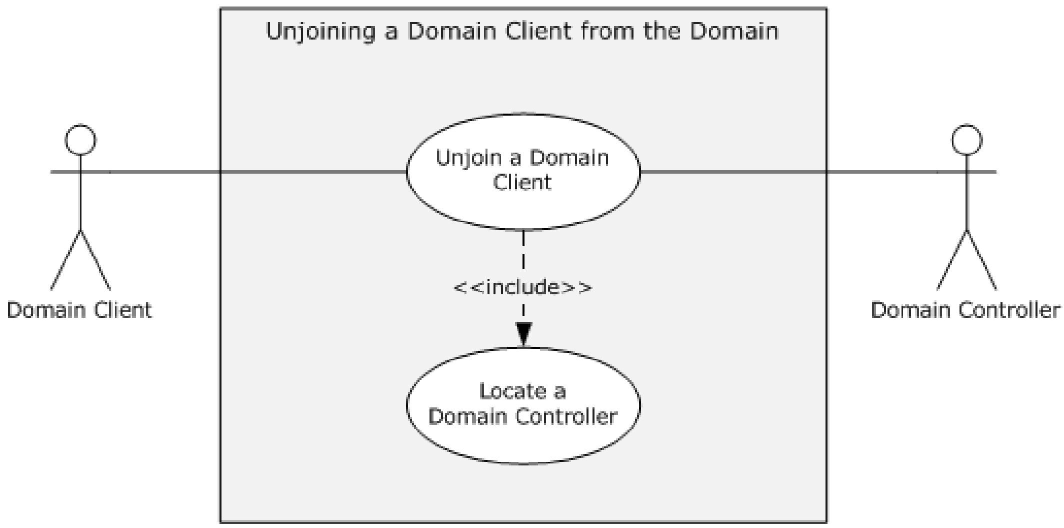

Figure 39: Use case diagram for unjoining a Domain Client from the domain

Actors

- Domain Client

  The Domain Client is the primary actor. It is the entity that locates and connects to the Domain Controller and unjoins from the domain. If the unjoin task fails, the local state of the Domain Client is unchanged.

- Domain Controller

  The Domain Controller is the supporting actor that advertises its capabilities, responds to domain-unjoin inquiries, and services the request from the Domain Client to disable the domain account.

Stakeholders

- Client Administrator

  The client administrator is the person who initiates the domain-unjoin process on the Domain Client.

  The primary interest of the client administrator is that the state of the computer object on the Domain Controller is updated to reflect that the Domain Client has unjoined from the domain.

- Client Computer

The client computer is the computer on which the Domain Client is running before the domain-unjoin task is initiated.

The primary interest of the client computer is that the machine local state of the Domain Client is updated to reflect that the client has unjoined from the domain.

Preconditions

- The client computer must have successfully completed the domain join task, as described in preceding sections, and must still be part of the domain.

Main Success Scenario

1. **Trigger**: This use case is triggered by the client administrator to unjoin the client computer from the domain.
2. The Domain Client locates a Domain Controller using the Locate a Domain Controller use case (see section 2.7.7.3.1).
3. The Domain Client establishes an SMB/CIFS connection to the Domain Controller using the credentials of the domain administrator ([MS-SMB2], [MS-SMB], or [MS-CIFS]).
4. The Domain Client disables the machine account on the Domain Controller using the SAMR protocol [MS-SAMR].
5. The (former) Domain Client updates its local state.
6. The (former) Domain Client closes connections.
7. The (former) Domain Client reinitializes local protocols.

Postconditions: The Domain Client is no longer joined to the domain.

Extensions: None.

2.7.7.3 Supporting Use Cases

2.7.7.3.1 Locate a Domain Controller - Domain Client

This use case describes the task of locating a domain controller. When an application on the client needs to access resources in a domain, locating a domain controller is the first step in the process.

This use case is important for other use cases and examples in which the Domain Client is not yet joined to the domain and the Netlogon Remote Protocol is therefore not initialized. For more information, see the preceding domain-services use cases and section 3.1.

Goal: Locate a domain controller in order to perform domain-oriented actions.

Context of Use: Any Domain Client that requires access to directory resources needs to authenticate itself to the directory. For Authentication, Domain Clients need to be connected to one of the Domain Controllers that it is reachable. To locate a reachable Domain Controller, Domain Clients perform this use case.

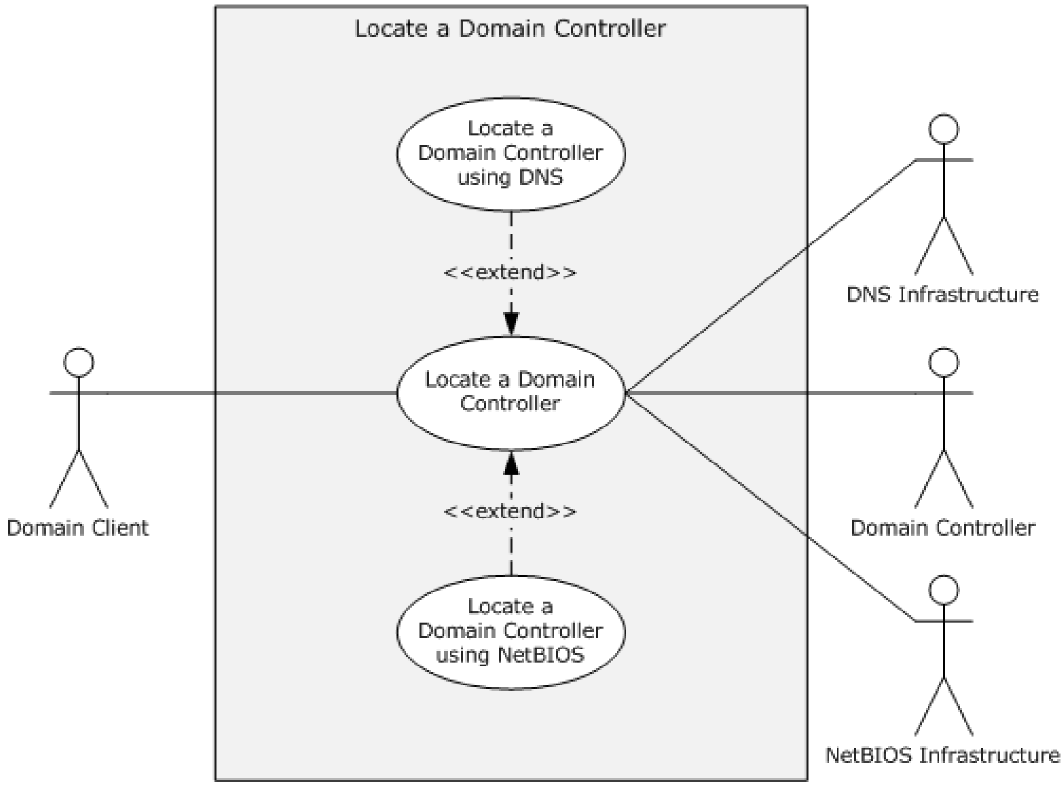

Figure 40: Use case diagram for locating a domain controller

Actors

- Domain Client

  The Domain Client is the primary actor. It is the entity that locates and queries the Domain Controller and that will eventually be joined to the domain.

- Domain Controller

  The Domain Controller is the supporting actor that registers its capabilities, responds to inquiries about those capabilities, and ultimately joins the Domain Client to the domain.

- DNS Infrastructure

  The DNS Infrastructure is a supporting actor that maintains information about Domain Controllers and sends that information to the Domain Client.

- NetBIOS Infrastructure

  The NetBIOS Infrastructure is a supporting actor that maintains information about Domain Controllers and sends that information to the Domain Client.

Stakeholders

- End User

The end user is the person who wants to join a Domain Client to a domain so that he or she can access resources in the domain.

The primary interest of the end user is whether a Domain Controller can be located so that the Domain Client can be joined to the domain and in receiving an error otherwise. If a Domain Controller cannot be located, the local state of the Domain Client is left unchanged.

- Applications

  Applications enable the end user to access resources within the domain. Applications can also use domain resources autonomously.

  The primary interest of an application is that a Domain Controller that meets the required capabilities is located and that information about the Domain Controller is provided to the Domain Client so that the Domain Client can be joined to the domain. By having the Domain Client joined to the domain, the application can use domain resources to perform the tasks that were initiated by the end user, by the application itself, or by other applications on the Domain Client.

Preconditions

Locating a domain controller relies on at least one of the infrastructures, NetBIOS or DNS, being available to discover domain controllers that can satisfy the requested capabilities.

Main Success Scenario

In this scenario, the FQDN(2) of the domain in which the Domain Controller is to be located is available to the Domain Client.

1. **Trigger**: This use case is triggered by operations such as joining a computer to a domain in order to locate a Domain Controller.
2. The Domain Client uses the FQDN(2) to query the DNS Infrastructure for the SRV records of certain types of Domain Controllers, as described in [MS-ADTS] section 6.3.6.1.
3. The DNS Infrastructure provides one or more SRV records of Domain Controllers that are of the specified type to the Domain Client.
4. The Domain Client resolves the names of the Domain Controllers using the DNS Infrastructure to obtain the IP addresses. It then contacts the Domain Controllers via an LDAP Ping ([MS-ADTS] section 6.3.3) to determine "liveness" and to confirm that the requested capabilities are present.
5. At least one Domain Controller that satisfies the Domain Client's requirements responds to the Domain Client's ping.
6. A Domain Controller is chosen for use in other tasks such as joining a domain.

Postconditions: A Domain Controller is located.

Extensions: None.

Variation - Locate a Domain Controller using NetBIOS

In this scenario, only the NetBIOS domain name of the domain is available. Domain Controllers in the domain have created a mailslot with a registered NetBIOS group name, as described in [MS-MAIL] section 3.1.4.1 and [MS-ADTS] section 6.3.5.

1. The Domain Client queries the NetBIOS Infrastructure for NetBIOS group names that contain a list of Domain Controllers.

2. The NetBIOS Infrastructure provides the NetBIOS group names.

3. Using the NetBIOS group names, the Domain Client contacts the candidate Domain Controllers via a MAILSLOT ping ([MS-ADTS] section 6.3.5) sent to a NetBIOS group name ([MS-MAIL] section 3.1.4.1) that is registered by Domain Controllers ([MS-ADTS] section 6.3.5).

4. At least one Domain Controller that satisfies the client's requirements responds to the Domain Client's ping.

5. A Domain Controller is chosen for use in other tasks such as joining a domain.

2.8 Versioning, Capability Negotiation, and Extensibility

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 provided in the table later in this section. Information about which versions of AD DS and AD LDS support Web Services protocols is given in the following product behavior note.<5>. 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 that is listed in the table.

Modes and Protocols Supported

Mode	Protocols supported	Protocols of which a subset is supported	Schemas implemented
AD DS (without Web Services)	DSSP LDAP LSAD LSAT SAMR	DRSR: 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 DS (with Web Services)	ADCAP DSSP LDAP LSAD LSAT SAMR WS-Enumeration WS-Transfer *Protocol Extensions* IMDA WSDS WSPELD	DRSR: 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	LDAP	DRSR: All methods of the drsuapi interface are	[MS-ADLS]

Mode	Protocols supported	Protocols of which a subset is supported	Schemas implemented
(without Web Services)		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.	
AD LDS (with Web Services)	LDAP WS-Enumeration WS-Transfer *Protocol Extensions* IMDA WSDS WSPELD	ADCAP: All methods of the AccountManagement port type are supported. The following methods of the TopologyManagement port type are supported: MoveADOperationMasterRole ChangeOptionalFeature DRSR: 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]

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 Services-enabled versions of both AD DS and AD LDS.

2.9 Error Handling

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
- Failures while joining or unjoining a domain

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

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 2.4, and in the failure scenarios described in the sections that follow.

Various kinds of errors might occur, impacting the system. More precisely, an error condition might 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.

2.9.1 Transient Unavailability of Durable Storage

As described in section 2.6, 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 can permit a request that requires retrieving state to succeed 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 that is 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 of the system's protocols should be in the same state as immediately prior to the storage becoming unavailable.

2.9.2 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 nonrepairable 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 can 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.

Because 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.

2.9.3 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 detects 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 should proceed in a manner similar to the case where the durable storage has become permanently unavailable. Essentially, data that cannot be trusted is treated in the same manner as no data at all.

2.9.4 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 outstanding requests. A functional networking system is also vital to the ability of directory servers to replicate directory data among themselves. This situation could cause a temporary loss of system coherency if segments of the network that are part of the same domain cannot communicate with each other but clients and domain controllers within the segments are still able to communicate.

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 to it, and directory servers will not be able to respond to replication requests. As such, clients and directory servers should always enforce a time-out on any requests that they send so that 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 and among directory servers, individual directory servers should resume processing requests in accord with the behavior described in the Technical Documents. The loss of networking does not in itself cause any change in the state of the abstract data of the system's protocols.

2.9.5 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. 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 that are dependent on the location algorithms of [MS-ADTS] section 6.3 will once again be able to locate directory servers.

2.9.6 Failures while Joining or Unjoining a Domain

Several of the examples in this document describe domain-join tasks that are completed through a series of actions that affect the necessary state changes such that the client is joined to the domain (see section 3.1). These changes include those that are local to the client and those that occur in the domain (that is, those that create or modify a computer account object on a domain controller (DC)). In general, failure of any one particular action must cause failure of the associated task. Exceptions to this principle are specified where necessary (for example, failed updates to the SNTP protocol [MS-SNTP] during join or unjoin processing are ignored).

Although unlikely, the domain-join tasks described in this document might fail when making local (client) state changes. Such failures may occur due to reasons such as resource starvation. The tasks do not attempt to remedy these failure conditions; the only recourse is for the task caller to re-execute the task.

When communicating with a remote machine such as a domain controller, some obvious potential failure conditions include lack of network connectivity, or insufficient security privileges to create or modify a computer account object. The domain-join tasks described in this document do not attempt to remedy these failure conditions; the only recourse is for the task caller to re-execute the task. When a task is re-executed, no assumptions should be made about the state of a computer-account object in the domain.

All domain-join tasks described in this document make reasonable efforts in the face of failure to restore local client state to the original starting state. If those efforts fail, administrator intervention (outside the scope of the task) might be necessary. Similarly, if a task successfully creates or modifies a computer account object in the domain but then fails in a later step, the task will make reasonable efforts to either disable or delete the computer account object. Failure to disable or delete the computer account object in that case might require domain administrator intervention (outside the scope of the task) to make the changes manually.

2.10 Coherency Requirements

Coherency requirements exist for shared state between the protocols that comprise the Active Directory System. 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 3.1.1.5.1.3), 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 those protocols (subject to access-control restrictions).

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.

2.11 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 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 not have access. If they can modify configuration information stored in the directory by other systems, such as the Group Policy system [MS-GPOD], they can exert control over the behavior of those systems. The directory can also contain sensitive information to which read access should be restricted.

In addition, interacting with a domain controller brings with it certain constraints that affect the security of a distributed system and domain as a whole. Since the domain controller serves as the root of trust for the entire domain, and potentially additional domains based on trust relationships, great attention must be paid to the correctness of the implementation of all the member protocols. Any security flaw in the implementation could compromise the entire domain. When beginning any interaction with the domain controller, the domain client is in a tenuous state. It needs to establish a connection and authenticate in a way that prevents attackers from manipulating the messages exchanged between the client and the domain controller.

It is crucial, therefore, 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 that is 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 also at the level of individual attributes on a directory object. The system also provides a means for securing the network traffic in the system against eavesdropping and tampering.

The following sections describe in more detail the security mechanisms and conditions in the Active Directory System.

2.11.1 Security Elements

Directory objects are protected by **security descriptors** that contain access control lists (ACLs) 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. 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 described in [MS-ADTS], 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:

- 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:

- DRSR
- SAMR
- SAMS
- LSAD
- LSAT
- DSSP

When performing an access check, the identity of the requestor, represented as a SID, is used to compare the permissions required to perform a given operation to 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 described in the corresponding protocol document, except that for WS-Transfer, WS-Enumeration, and ADCAP, it is instead described in [MS-ADDM] section 2.1.

The protocols provide mechanisms to digitally sign requests and responses to protect them from tampering while being transferred over the network and to encrypt the traffic to prevent eavesdropping. For more information, see section 2.11.2.

2.11.2 Communications Security

The Active Directory System relies on messages that are passed across the network between the client and the directory service, and from one directory service server to another. The system does not require this network to be fully trusted and allows for the possibility that a hostile party might be able to intercept such messages while they are being transferred. 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 gain information that the attacker is not intended to have.
- 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, and between directory service servers. Transport-level security protects the entire transport, effectively creating a protected "tunnel" between machines through which the messages are sent, protecting the confidentially and integrity of the messages sent through 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 (SSL)**/**Transport Layer Security (TLS)**-protected connection.	[MS-ADTS] section 5.1.2.2
	Message-level Protection is provided by signing and/or encryption using SASL.	[MS-ADTS] section 5.1.2.1, Using **SASL**
DRSR	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] sections 2.2.3.1 and 2.2.4.1
DSSP	None	
SAMR	Transport-level When using RPC over the SMB transport, protection is provided by the SMB transport.	[MS-SAMR] section 2.1

Protocol	Mechanisms	Reference
	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
LSAD	Transport-level Protection is provided by the SMB transport over which the RPC requests are sent.	[MS-LSAD] section 2.1
LSAT	Transport-level When using RPC over the SMB transport, protection is provided by the SMB transport.	[MS-LSAT] sections 2.1 and 3.1.4
	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 and 3.1.4
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
WS-Enumeration	Same as WS-Transfer, above.	[MS-ADDM] section 2.1
ADCAP	Same as WS-Transfer, above.	[MS-ADDM] section 2.1

In addition to these mechanisms for protecting desirable traffic between machines, 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 2.11.1) 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 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
DRSR	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
DSSP	None	
SAMR	Uses IDL "[range]" attributes to limit the size of requests that will	[MS-SAMR]

Protocol	Mechanisms	Reference
	be accepted by the directory service.	section 6
	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
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 and 7
	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
	Configures the RPC runtime to enforce Maximum Server Input Data Size.	[MS-LSAD] section 2.2.1
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 6 and 7
	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
WS-Transfer	*	
WS-Enumeration	*	
ADCAP	*	

* Implementations can provide mechanisms to limit the operations that can be performed or the size of the response.<6>

2.11.3 System Configuration Security

The configuration data and parameters for the Active Directory System are stored in the directory service itself. The configuration data is retrieved and manipulated using the same protocols that are used to manipulate any other data that is stored in the directory. In particular, much of it is stored in the form of directory objects that are accessible via the LDAP protocol. As such, this configuration data is protected by the access checks that are 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 2.11.2. 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 of the attacker's choosing (using the client's privileges).

2.11.4 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 2.11.1, 2.11.2, and 2.11.3 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 **service principal names (SPNs)** defined in [MS-DRSR] sections 2.2.3.2, 2.2.3.3, 2.2.4.2, and 2.2.4.3 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 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.

2.11.5 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 time-outs so that even if the directory service does not respond in a timely fashion 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.

- Use the SPNs to perform mutual authentication to ensure that it is communicating with the intended directory service and not an impostor.

2.12 Additional Considerations

None.

3 Examples

The following sections provide examples for the most common activities in the Active Directory System. These examples are divided into two major categories: domain-join examples and directory examples.

Note Windows clients might generate extra messages for their internal processing. These messages are not required to achieve the goals of the examples and are therefore not described in this document.

3.1 Domain-Join Examples

This section contains a set of examples that illustrate the activity of the Active Directory system when a client computer is joined to a domain. The following types of examples are covered in this section:

- Locating a domain controller
- Joining a domain
- Unjoining from a domain

A key aspect of these examples is that multiple protocols, such as LDAP and SAMR, can be used to affect the same state change such as provisioning a user. After that state change is completed, 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, a user that was created by using the SAMR protocol can have his or her password changed by using the LDAP protocol.

3.1.1 Example 1: Locate a Domain Controller

This example shows the pattern for locating a domain controller based on the domain name provided--both flat, NetBIOS names as well as the fully qualified DNS names preferred for AD-style domains. This example is useful in order to locate a domain controller for the purpose of joining a domain or performing other domain-related operations in a domain environment.

This example builds on the use case covered in section 2.7.7.3.1, Locate a Domain Controller - Domain Client.

Prerequisites

The general requirements set forth in section 2.4, Assumptions and Preconditions.

The Active Directory System must meet all preconditions described in section 2.7.7.3.1.

Initial System State

The location of a domain controller (DC) is not known.

Final System State

A DC that meets the required capabilities is located, and information about the DC is provided to the caller. If this goal cannot be achieved, an error is returned indicating that a DC could not be located.

Upon failure, the local state of the **client computer** is unchanged.

Sequence of Events

This task can be performed by using one or both of two possible approaches: DNS-based location or NetBIOS-based location. If the **fully qualified domain name (FQDN) (2)** of the domain is available, DNS-based location is performed. If only the NetBIOS name of the domain is available, or if the DNS-based location is unsuccessful, NetBIOS-based location is performed.

Locate a Domain Controller Using the DNS Infrastructure

The following sequence diagram depicts the message flow associated with this example.

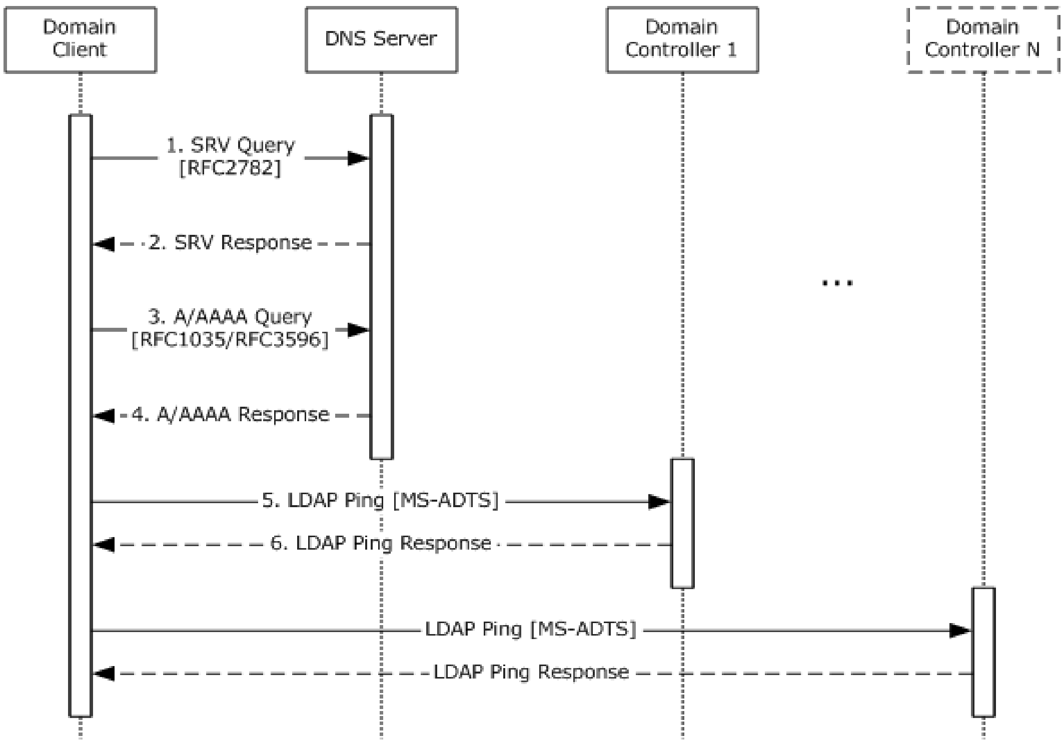

Figure 41: Message flow for domain location based on a DNS domain name

1. The FQDN of the domain in which the domain controller is to be located is available. The domain client makes use of the FQDN of the domain along with site information from the client and sends an SRV query to the DNS server to get a list of IP addresses of candidate domain controllers.

2. If no candidate domain controllers can be identified using SRV queries, the client should fall back to NetBIOS-based location (described later in this section).

   Active Directory servers offer the LDAP service over the TCP protocol; for example (see [MS-ADTS] section 6.3.6), clients can find an LDAP server by querying DNS for a record of the form:<7>

   ```
   _ldap._tcp.DnsDomainName
   ```

3. After the list of candidate domain controllers is obtained using the DNS infrastructure, the domain client selects a candidate domain controller based on weighted random order ([RFC2052]). The client sends A/AAAA DNS queries to resolve the SRV record to an IP address.

4. The domain client receives the IP address that was resolved from the SRV record in an A/AAAA DNS response.

5. Once the IP address is known, the client sends an LDAP "Ping" to the candidate domain controller to determine whether the domain controller is in fact handling requests and whether its capabilities satisfy the client's requirements.

6. Upon receipt of a successful LDAP Ping response, the domain client validates that the capabilities returned by the domain controller satisfy the requested capabilities. If the capabilities returned by the domain controller are incompatible with the requirements specified by the client, the client selects another candidate domain controller from the list of domain controller SRV records returned in step 2 and repeats steps 3 through 6.

   If all the responses in the SRV records have been checked and each SRV record points to a server that is either not available or does not match the requirements, then the DC location operation fails. In this case, the client should fall back to NetBIOS-based DC location (described later in this section). If a domain controller still cannot be found, an error is returned indicating that a domain controller could not be located.

   If the domain in which the domain controller is to be located is the same as the client computer's domain, the site name abstract data is updated with the client site name information returned as part of the LDAP Ping response by the domain controller.

Locate a Domain Controller Using the NetBIOS Infrastructure

The following sequence diagram depicts the message flow associated with this example.

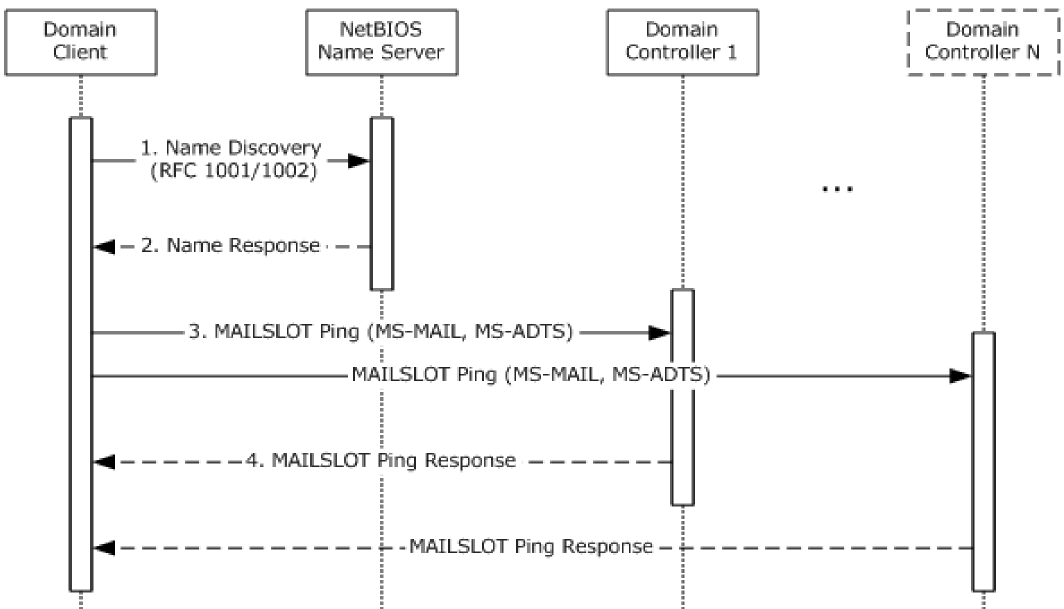

Figure 42: Message flow for domain location based on a NetBIOS domain name

1. When the FQDN of the domain is not available, the domain client determines the NetBIOS name of the domain based on the client's abstract data. If the NetBIOS name cannot be determined, an error is returned indicating that a domain controller could not be located. The domain client queries the NetBIOS name server for NetBIOS group names that contain the list of DCs.

2. The NetBIOS name server responds with the NetBIOS group names.

3. Using the NetBIOS group names that are registered by domain controllers along with their capabilities, the domain client sends a MAILSLOT Ping to candidate domain controllers using the Remote Mailslot Protocol. The ping response is used to determine availability and to confirm that the domain controller supports all the specified requirements.

4. Upon receipt of a successful MAILSLOT Ping response, the domain client validates that the capabilities returned by the domain controller satisfy the requested capabilities. If no domain controllers respond or if none match the required capabilities, the client returns an error indicating that a domain controller could not be located.

3.1.2 Example 2: Joining a Domain by Creating an Account via SAMR

This example describes the process of joining a client computer to a domain by creating an account via the SAMR protocol. This is a secure method for joining a domain, which involves establishing the account on the domain controller and creating a new, random password for the domain client and the account in the domain. This example is useful in order to join a domain or perform other domain-related operations using the SAMR protocol in a domain environment.

This example builds on the use case covered in section 2.7.7.1, Join a Domain with a New Account-Domain Client.

Prerequisites

The general requirements set forth in section 2.6, Assumptions and Preconditions.

The Active Directory System must meet all preconditions described in section 2.7.7.1.

Initial System State

The client is in a workgroup.

Final System State

Upon successful completion, the client state is updated indicating that the client is now joined to the domain.

Sequence of Events

The following sequence diagram depicts the message flow associated with this example.

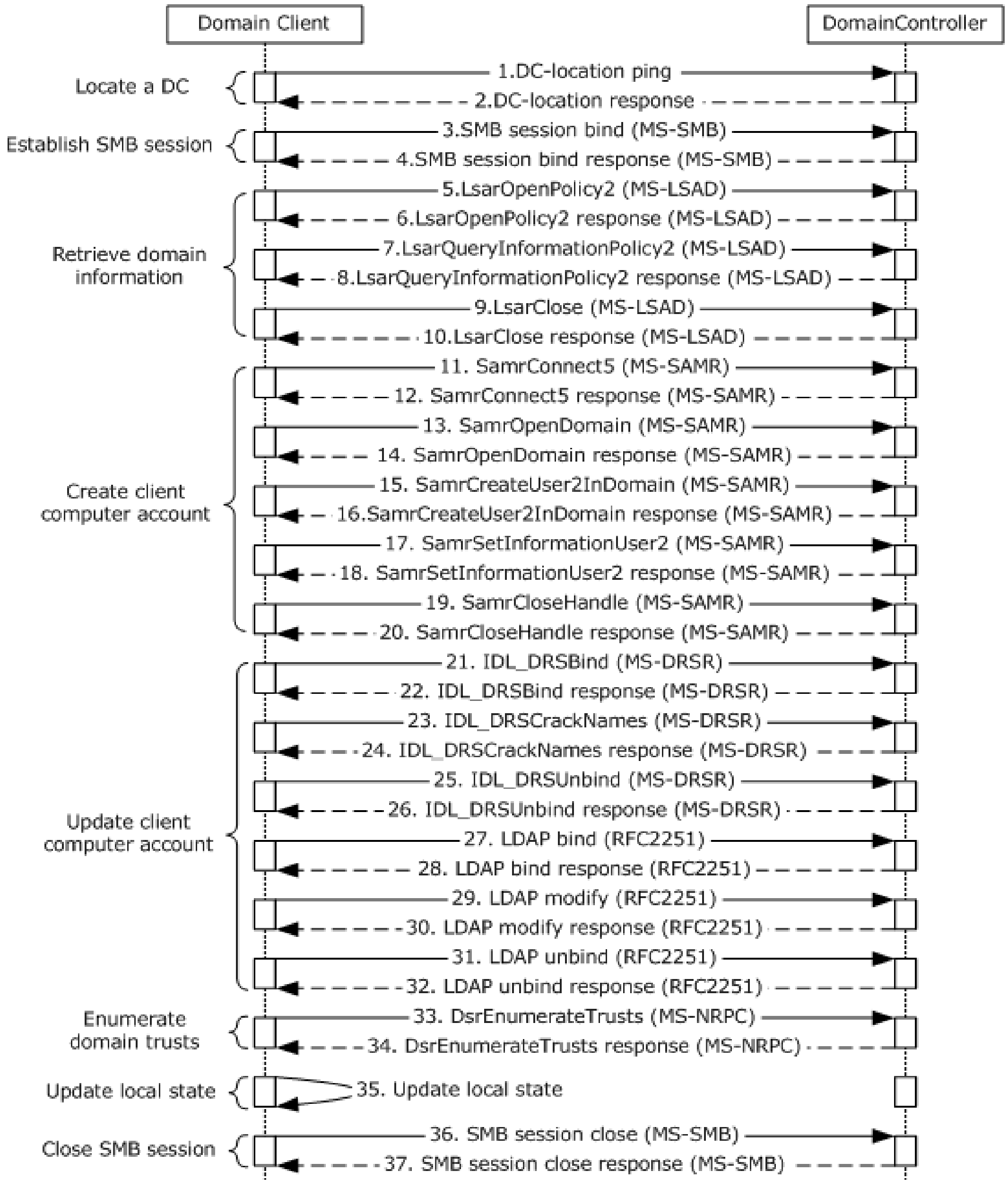

Figure 43: Message flow for joining a domain by creating a new account via SAMR

1. If the domain controller was not located earlier, the client locates a domain controller as specified in section 3.1.1.
2. If the client receives a successful response from the domain controller, the DC is located; if the response is not successful, the task fails.
3. To establish an SMB/CIFS session to the domain controller, the domain client sends an SMB session bind request using anonymous user credentials to the domain controller ([MS-CIFS] section 3.2.4.2).

4. When a successful SMB session bind response is received from the domain controller, the SMB/CIFS session is established between the domain client and the DC.

5. Using the SMB/CIFS session established in the preceding step as transport, the domain client makes an LsarOpenPolicy2 RPC call for obtaining policy handle. This handle is used for subsequent calls to retrieve domain information from the domain controller ([MS-LSAD] section 3.1.4.4.1).

6. Upon receiving a successful response from the domain controller, the domain client obtains a policy handle.

7. Using the policy handle obtained from the preceding step, the domain client sends an LsarQueryInformationPolicy2 request to the domain controller to retrieve domain information ([MS-LSAD] section 3.1.4.4.3).

8. Upon receiving a successful response from the domain controller, the domain client retrieves domain information and updates its abstract data.

9. The domain client makes an LsarClose RPC call to close the policy handle ([MS-LSAD] section 3.1.4.9.4.) obtained in step 6.

10. Upon a successful response from the domain controller, the policy handle is closed. Any failure during the preceding call sequence will cause the task to fail.

11. Using the SMB connection established previously, the domain client makes a SamrConnect5 request ([MS-SAMR] section 1.7.2) to the domain controller in order to connect to the SAM RPC server on the DC.

12. Upon a successful response from the domain controller, the domain client receives a handle to the server object from the DC.

13. Using the server handle obtained in the preceding step, the domain client sends a SamrOpenDomain request ([MS-SAMR] section 3.1.5.1.5) to the domain controller to obtain a handle for a **domain object**.

14. Upon a successful response, the domain controller returns a handle for a domain object.

15. The domain client sends a SamrCreateUser2InDomain request ([MS-SAMR] section 3.1.5.4.4) to the domain controller to create a new user account.

16. Upon a successful response from the domain controller, a new user of the domain client computer is created on the domain controller. If this fails because the account already exists, the domain client attempts to obtain a handle to the existing account ([MS-SAMR] sections 3.1.5.1.9 and 3.1.5.11.2).

17. The domain client sends a SamrSetInformationUser2 request ([MS-SAMR] section 3.1.5.6.4) to the domain controller to set the password of the newly created user.

18. Upon a successful response from the domain controller, the password is set.

19. After the new user account has been created, the domain client sends a SamrCloseHandle request to the domain controller to close the handles that were opened previously ([MS-SAMR] section 3.1.5.13.1).

20. Upon successful completion of the preceding call sequence, the domain client has successfully created or updated the client account in the domain.

21. The domain client sends an IDL_DRSBind request, which creates a context handle that is necessary in order to call any other methods in the interface ([MS-DRSR] section 4.1.3).

22. Upon a successful response from the domain controller, the domain client obtains a context handle.

23. The domain client sends an IDL_DRSCrackNames request ([MS-DRSR] section 4.1.4) to the domain controller to obtain the distinguished name (DN).

24. Upon a successful response from the domain controller, the domain client updates the DN of the client account.

25. The domain client sends an IDL_DRSUnbind request, which destroys the context handle that was previously created by the IDL_DRSBind method ([MS-DRSR] section 4.1.25).

26. Upon a successful response from the domain controller, the context handle that was created previously is destroyed.

27. The domain client connects to the LDAP service as described in the LDAP specification [RFC2251] and performs an LDAP bind to authenticate the connection using the process that is described in [MS-ADTS] section 5.1.1.

28. Upon a successful LDAP bind response, the client account is validated successfully.

29. Using the LDAP connection created in the previous step, the domain client sends an LDAP modify request to the domain controller. This request is used to update the client account with values for the server principal name and host name attributes.

30. Upon a successful response from the domain controller, the domain client updates the attributes in the client account.

31. The domain client sends an LDAP unbind request to unbind from the LDAP service, as specified in the LDAP specification [RFC2251].

32. Upon a successful response from the domain controller, the domain client unbinds from the LDAP service. This operation is performed even if errors were encountered in the preceding call sequence. Any errors in this operation are ignored.

33. To retrieve the list of domain trusts from the domain controller, the domain client sends a DsrEnumerateTrusts request ([MS-NRPC] section 3.5.4.6.1).

34. Upon a successful response from the domain controller, the domain client retrieves the list of trusts from the DC.

35. The domain client updates its local state variables.

36. After updating its local state, the domain client sends an SMB session close request ([MS-CIFS] section 3.4.4.8) to the domain controller to close the SMB/CIFS session that was established previously.

37. Upon a successful response from the domain controller, the SMB/CIFS session is closed.

3.1.3 Example 3: Joining a Domain by Creating an Account via LDAP

This example describes the process of joining a client computer to a domain by creating an account via Lightweight Directory Access Protocol (LDAP). This is a secure way of joining the domain. This task shares many of the actions in establishing the relationship between the client and the domain controller (DC) with the example in section 3.1.2. As a variation of that example, the creation or modification of the machine account in the domain can be accomplished via LDAP instead of the SAM RPC interface. This can be useful for alternate domain client implementations that do not wish to include the SAM RPC interface client. This example also highlights again why the domain controller server implementation must enforce that the multiple interfaces to the same underlying **account database** keep the objects synchronized across the different protocol interfaces. This example is useful in order to join a client to a domain or perform other domain-related operations using the LDAP protocol in a domain environment.

This example builds on the variation in the use case covered in section 2.7.7.1.

Prerequisites

The general requirements set forth in section 2.6.

The Active Directory System must meet all preconditions described in section 2.7.7.1.

Initial System State

The client is in workgroup.

Final System State

Upon successful completion, the client state is updated indicating that the client is now joined to the domain.

Sequence of Events

The following sequence diagram depicts the message flow associated with this example.

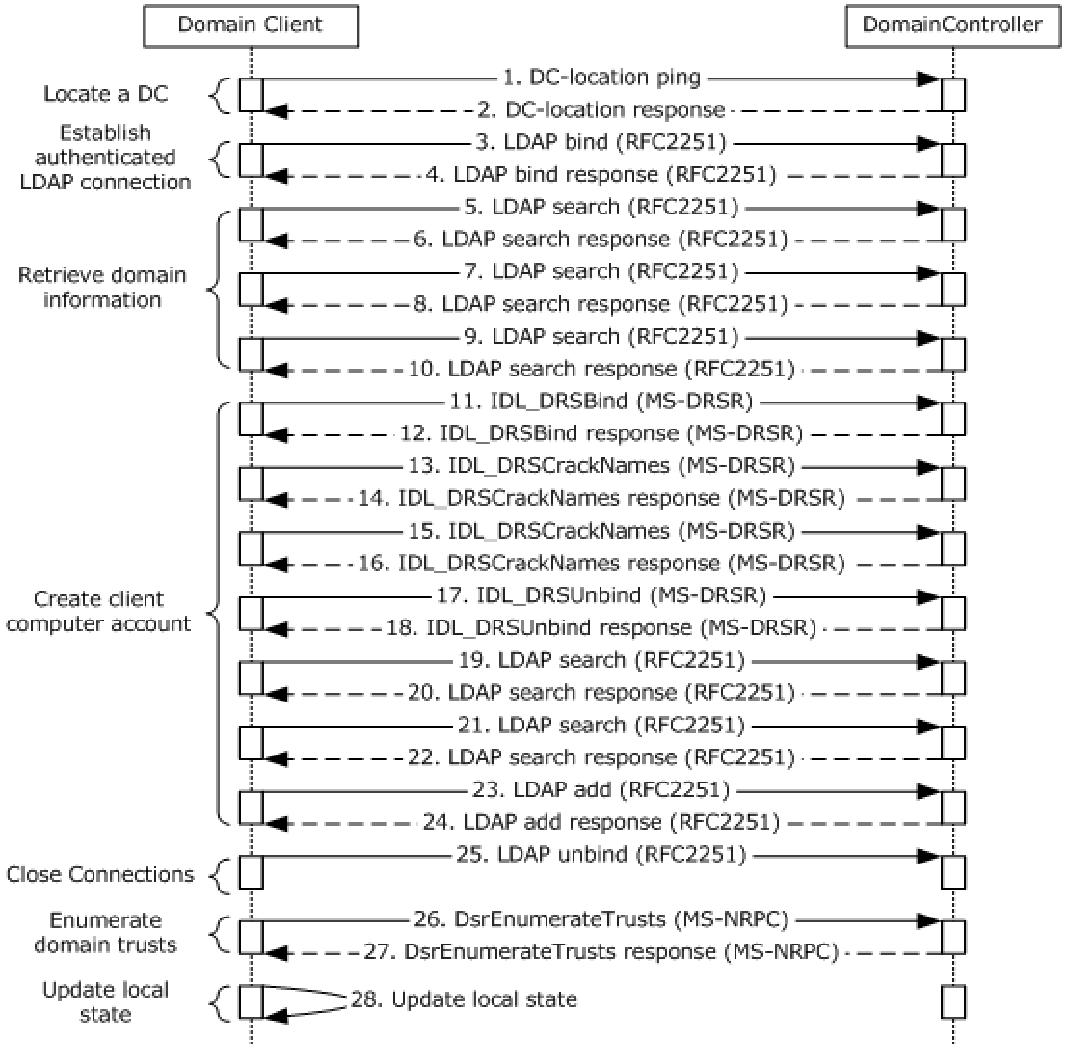

Figure 44: Message flow for joining a domain by creating a new account via LDAP

1. If the domain controller was not located earlier, the client locates a domain controller as specified in section 3.1.1.

2. If the client receives a successful response from the domain controller, the DC is located; if the response is not successful, the task fails.

3. The domain client computer sends an LDAP bind request ([RFC2251] section 4.2) to the domain controller with the credentials of the administrator.

4. The domain controller uses one of the methods defined elsewhere ([MS-AUTHSOD] section 2) to verify the credentials. Depending on the negotiated authentication method, this might involve additional domain client and server interactions not directly relevant to this discussion. After verification, the domain controller sends an LDAP bind response ([RFC2251] section 4.2.3) to the domain client.

5. The domain client computer sends an LDAP search request to the domain controller ([RFC2251] section 4.5.1) with baseObject set to the NULL domain name.

6. Upon a successful response from the domain controller, the domain client updates the local domain name and the forest name in its abstract data.
7. The domain client computer sends an LDAP search request to the domain controller ([RFC2251] section 4.5.1) with baseObject set to the domain name obtained from step 5.
8. Upon a successful response from the domain controller, the domain client updates the local domain GUID and domain SID in its abstract data.
9. The domain client computer sends an LDAP search request to the domain controller ([RFC2251] section 4.5.1) with baseObject set to the concatenation of "CN=Partitions," with the value of configurationNamingContext obtained in step 5.
10. Upon a successful response from the domain controller, the domain client updates the NetBIOS name in its abstract data.
11. The domain client sends an IDL_DRSBind request ([MS-DRSR] section 4.1.3) to create a context handle that is necessary to call any other DRS messages.
12. The domain controller sends and IDL_DRSBind response to the domain client.
13. The domain client sends an IDL_DRSCrackNames request ([MS-DRSR] section 4.1.4) to the domain controller to get the distinguished name (DN).
14. Upon a successful response from domain controller, if the DN of the organizational unit (OU) used to create the computer account was specified and the account exists under the OU, the domain client updates the DN of the client account.
15. If the DN of the OU used to create the computer account was specified and the account does not exist under the OU, the domain client sends an LDAP compare request ([RFC2251] section 4.10) to the domain controller; otherwise, the domain client sends an IDL_DRSCrackNames request ([MS-DRSR] section 4.1.4).
16. Upon a successful response from the domain controller, the domain client updates the DN of the client account.
17. The domain client sends an IDL_DRSUnbind request ([MS-DRSR] section 4.1.25) to destroy the context handle created previously.
18. Upon a successful response from the domain controller, the previously created context handle is destroyed.
19. If the DN of the OU used to create the computer account was not specified and the account does not exist under the OU, the domain client sends an LDAP search request ([RFC2251] section 4.5.1) to the domain controller.
20. Upon a successful response from domain controller, the domain client parses the returned value for the DN of the computer container DN (see [MS-ADTS] section 6.1.1.4 for details) and obtains the DN of the preferred OU. If this search fails, the task will fail.
21. The domain client sends LDAP search request ([RFC2251] section 4.5.1) to the domain controller with baseObject set to the preferred OU obtained from previous request.
22. Upon a successful response from the domain controller, the domain client updates the DN of the client account.
23. The domain client sends an LDAP add request ([RFC2251] section 4.7) to the domain controller to add the new account into the directory with the updated DN from the previous step.

24. Upon a successful response from domain controller, the new account is added in the directory.

25. The domain client sends an LDAP unbind message ([RFC2251] section 4.3) to the domain controller to remove all the connections that were built in the previous steps.

26. To retrieve the list of domain trusts from the domain controller, the domain client sends a DsrEnumerateTrusts request ([MS-NRPC] section 3.5.4.6.1).

27. Upon a successful response from the domain controller, the domain client retrieves the list of trusts from the DC.

28. The domain client updates its local state variables.

3.1.4 Example 4: Unjoining a Domain Member

This example describes the process of unjoining a client computer from a domain. To unjoin from a domain, a client administrator locates a domain controller (DC) and then performs actions against the DC. This example is applicable to a client computer that is part of a domain and needs to unjoin from the domain.

This example builds on the use case covered in section 2.7.7.2, Unjoin from a Domain - Domain Client.

Prerequisites

The general requirements set forth in section 2.6, Assumptions and Preconditions.

The Active Directory System must meet all preconditions described in section 2.7.7.2.

Initial System State

The client is joined to a domain.

Final System State

Upon successful completion of this task, the client's state variables are updated and client is unjoined from the domain.

Sequence of Events

The following sequence diagram depicts the message flow associated with this example.

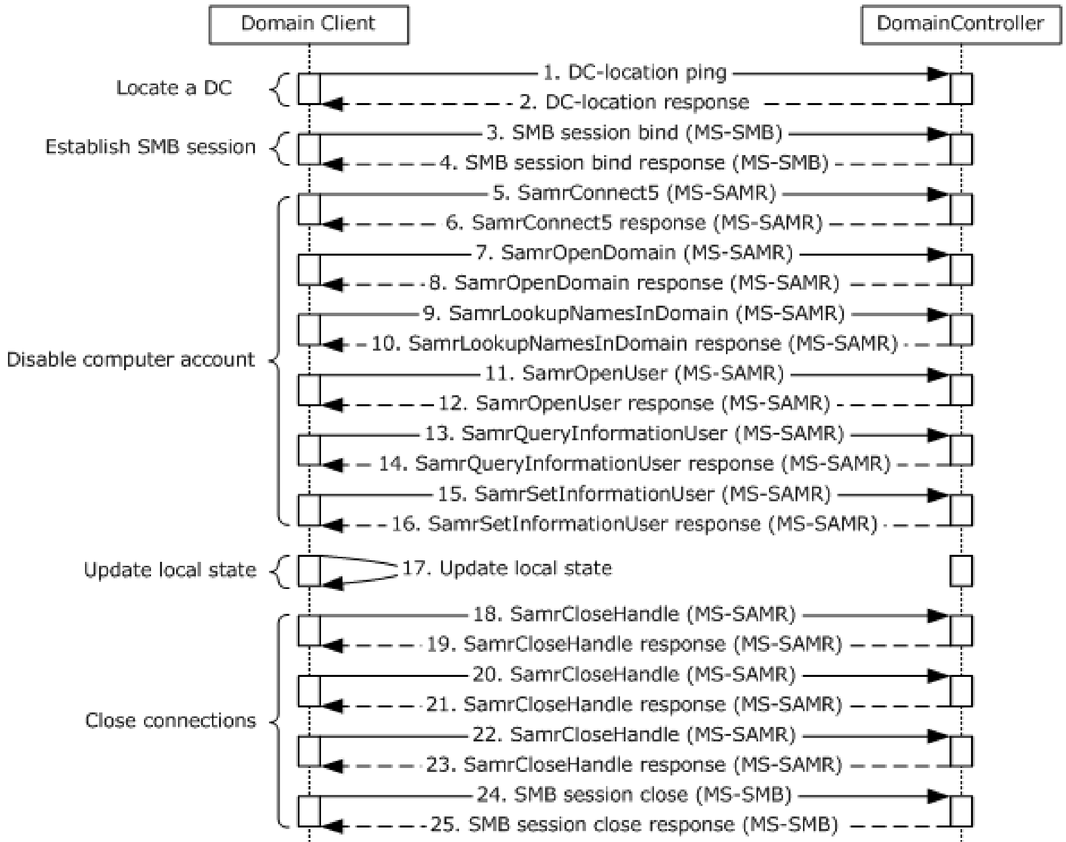

Figure 45: Message flow for unjoining from a domain

1. If the domain controller was not located earlier, the client locates a domain controller as specified in section 3.1.1.

2. If the client receives a successful response from the domain controller, the DC is located; if the response is not successful, the task fails.

   Note The initial exchange (to locate a DC) is representative only of the traffic between the client and the selected domain controller. This traffic might not even be present, depending on whether previous results from the Locate a Domain Controller task (section 3.1.1) have been cached. Also, additional exchanges that might occur to other domain controllers are not represented.

3. To establish an SMB/SMB2/CIFS session to the domain controller, the domain client sends an SMB session bind request using anonymous user credentials to the domain controller ([MS-CIFS] section 3.2.4.2).

4. Upon a successful response from domain controller, the SMB/SMB2/CIFS session is established between the domain client and the DC.

5. Using the SMB connection established in the previous step, the domain client sends a SamrConnect5 request ([MS-SAMR] section 3.1.5.1.1) to the domain controller in order to connect to the SAM RPC server on the DC.

6. Upon a successful response from domain controller, the domain client receives a handle to the server object from the DC.

7. Using the server handle obtained in the preceding step, the domain client sends a SamrOpenDomain request ([MS-SAMR] section 3.1.5.1.5) to the domain controller to obtain a handle for a domain object.

8. Upon a successful response, the domain controller returns a handle for a domain object.

9. In order to determine the relative ID (RID) of the account, the domain client sends a SamrLookUpNamesInDomain request ([MS-SAMR] section 3.1.5.11.2) to the domain controller.

10. Upon a successful response, the domain controller returns the RID of the existing domain-client account.

11. In order to obtain a handle to modify user account information, the domain client sends a SamrOpenUser request ([MS-SAMR] section 3.1.5.1.9) to the domain controller.

12. Upon a successful response, the domain controller returns a handle to a user account.

13. The domain client sends a SamrQueryInformationUser request ([MS-SAMR] section 3.1.5.5.6) to the domain controller in order to obtain attributes from the user object.

14. Upon a successful response, the domain controller returns attributes of the user object.

15. In order to disable the user account in the directory, the domain client sends a SamrSetInformationUser request ([MS-SAMR] section 3.1.5.6.5) to the domain controller.

16. Upon a successful response from the domain controller, the domain client disables the user account in the directory.

17. The domain client updates its local state variables.

18-23. After the user account is disabled, the domain client sends SamrCloseHandle requests to the domain controller to close the handles opened earlier ([MS-SAMR] section 3.1.5.13.1). The client receives responses from the server for all the close-handle requests. Upon successful completion of the preceding call sequence, the domain client has successfully created or updated the client account in the domain.

24. The domain client sends an SMB session close request ([MS-CIFS] section 3.4.4.8) to the domain controller to close the SMB/SMB2/CIFS session that was established earlier.

25. Upon a successful response from the domain controller, the SMB/SMB2/CIFS session is closed.

3.2 Directory Examples

This section contains a set of examples that illustrate common uses of the Active Directory System. The following examples are given:

- 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 examples is that multiple protocols, such as LDAP and SAMR, can be used to affect the same state change, such as provisioning a user. Once that state change is complete, 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 that was created using the SAMR protocol can have his or her password changed using the LDAP protocol, or be queried using the Web Services protocols.

3.2.1 Example 1: Provision a User Account Using the LDAP Protocol

In this example, an administrator provisions a user account using the LDAP protocol. To perform this task, the 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, sets its user properties using the LDAP protocol, and sets the user account's password using the Kerberos protocol.

This example applies only to AD DS.

This example uses the LDAP and Kerberos protocols and uses concepts described in [MS-SAMR] (though it does not use the protocol defined in [MS-SAMR]).

This example covers the use cases in sections 2.7.2.1, Create a New Account - Client Application, and 2.7.2.2, Reset an Existing Account's Password - Client Application.

Prerequisites

The general requirements set forth in section 2.6, Assumptions and Preconditions.

The Active Directory System must meet all preconditions described in section 2.7.2.1.

Initial System State

None.

Final System State

The new user object has been provisioned in the directory with the specified attributes.

Sequence of Events

The following sequence diagram depicts the message flow associated with this example.

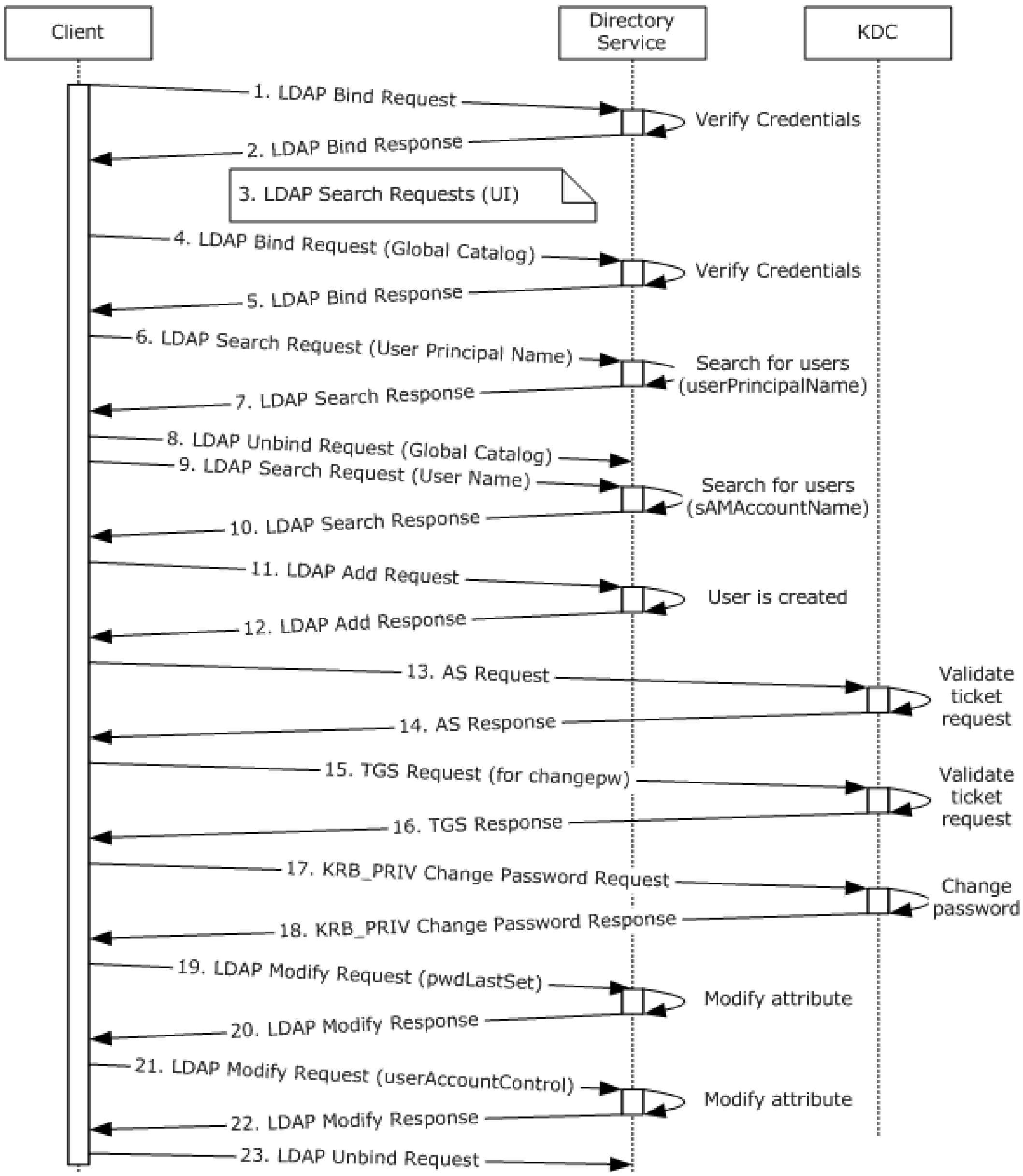

Figure 46: Message flow for provisioning a user account using the LDAP protocol

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

1. The client application starts and an LDAP bind request ([RFC2251] section 4.2) is sent to the directory server with the credentials of the administrator.

2. The directory server uses one of the methods defined elsewhere ([MS-AUTHSOD] section 2) to verify the credentials. Depending on the negotiated authentication method, this might involve additional client and server interactions that are not directly relevant to this discussion. After

verification, the directory server sends an LDAP bind response ([RFC2251] section 4.2.3) to the client.

3. At this point the client sends LDAP search requests ([RFC2251] section 4.5.1) to populate data in the client application's user interface. This step is necessary only for user-interface display purposes specific to the example.

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

5. The global catalog verifies the credentials ([MS-AUTHSOD] section 2) and sends an LDAP bind response ([RFC2251] section 4.2.3).

6. An LDAP search request ([RFC2251] section 4.5.1) is sent to the global catalog, starting at the root of the forest and querying the entire forest, to look for security principals that have the same **user principal name (UPN)** as the requested user principal name of the new account. The purpose of this search is 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) listing any accounts that have the specified user principal name.

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

9. The client application sends an LDAP search request ([RFC2251] section 4.5.1) to the directory, starting at the root of the domain and querying the entire domain, to look for security principals that have the same name (stored in the sAMAccountName attribute) as the one requested for the new account. The purpose of this search is 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) listing any accounts that have the specified user name.

    For the purposes of this example, it is assumed that the LDAP search responses for the user principal name and the user name do not contain any accounts; that is, there were no matches. With this assumption, the client application has now verified that the chosen user principal name and user name are not currently in use. The client application continues with the add operation.

11. An LDAP add request ([RFC2251] section 4.7) 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) and performs validation as described in [MS-ADTS] sections 3.1.1.5.1 and 3.1.1.5.2. It then sends an LDAP add response indicating success.

    Now that the user has been created, the client application 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) to the Authentication Service (AS) requesting a **ticket-granting ticket (TGT)** that it will use later for authentication.

14. The service validates the credentials in the AS request and sends an AS response ([RFC4120] section 3.1) with a TGT.

15. The client sends a Kerberos TGS request ([RFC4120] section 3.3) to the **ticket-granting service (TGS)** using the TGT and requests 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) with the service ticket.

17. Using the service ticket, the client sends a **KRB_PRIV** change-password request ([RFC4120] section 3.5 and [RFC3244] section 2) 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 and [RFC3244] section 2) 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.

    At this point the Kerberos operations are complete. 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 the next logon, the client sends an LDAP modify request ([RFC2251] section 4.6) 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).

    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 2.2.15) and may contain additional bits depending on administrator-provided values.

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

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

    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) to the server. The LDAP connection to the directory server is closed.

3.2.2 Example 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 to accomplish this task 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 example applies only to AD DS.

This example differs from the previous example (section 3.2.1) in that it uses the SAMR protocol rather than LDAP to create the user, and also in that it only provides a minimal set of attributes for the newly created account.

This example covers the use case in section 2.7.2.1, Create a New Account - Client Application.

Prerequisites

The general requirements set forth in section 2.6, Assumptions and Preconditions.

The Active Directory System must meet all preconditions described in sections 2.7.1.1 and 2.7.2.1.

Initial System State

None.

Final System State

The new user object has been created in the directory.

Sequence of Events

The following sequence diagram depicts the message flow that is associated with this example.

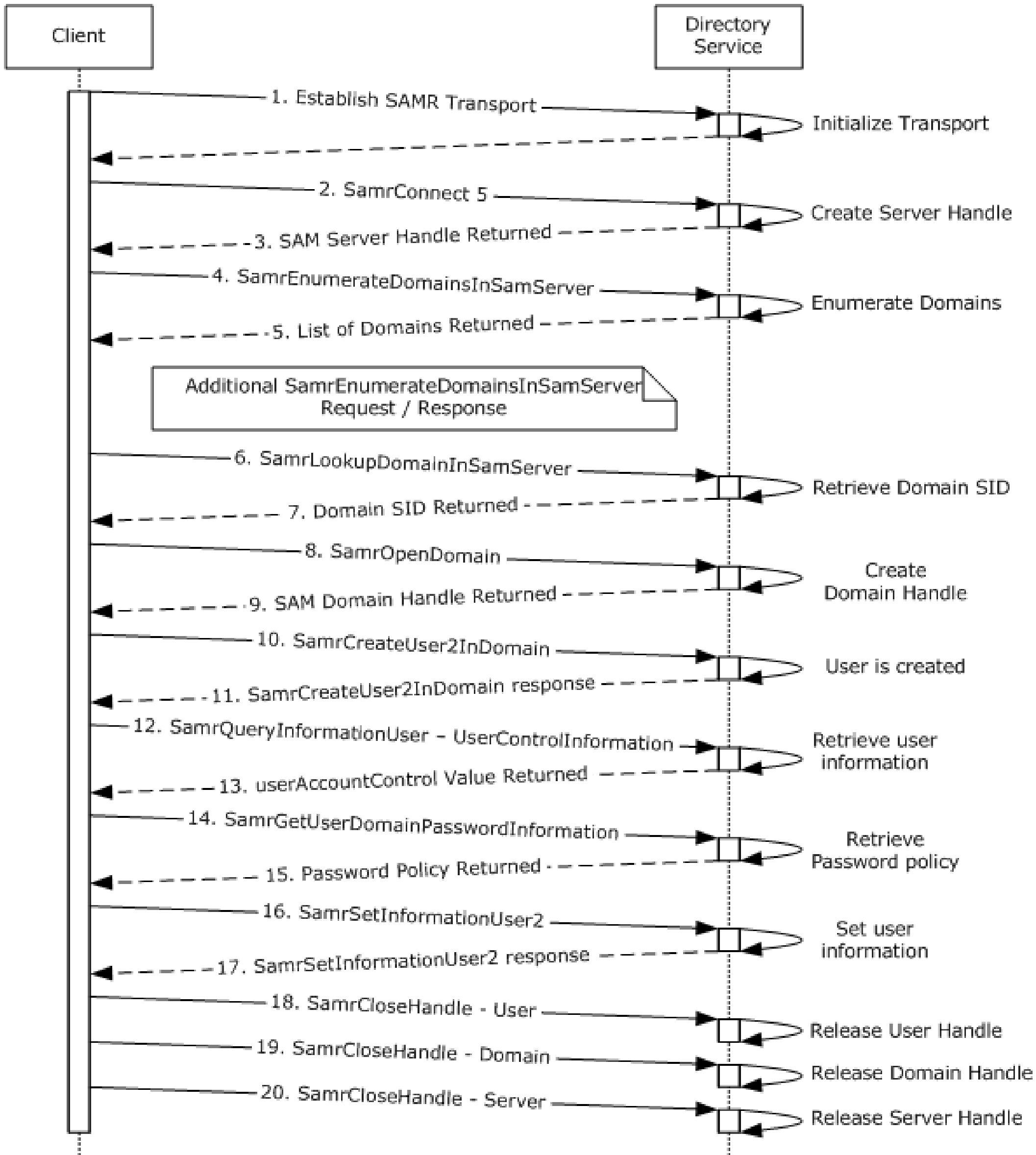

Figure 47: Message flow for provisioning a user account using the SAMR protocol

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.
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) with the DesiredAccess parameter set to MAXIMUM_ALLOWED ([MS-SAMR] section 2.2.1.1) to the server, requesting a server handle.
3. The server processes the request ([MS-SAMR] section 3.1.5.1.1) 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) using the server handle it previously obtained in step 3.

5. The server processes the request ([MS-SAMR] section 3.1.5.2.1) 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) 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) using the server handle from step 3. In this request, it specifies one of the names 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) 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) using the server handle it previously obtained in step 3 and the domain SID it obtained in step 7, with the DesiredAccess parameter set to MAXIMUM_ALLOWED ([MS-SAMR] section 2.2.1.1).

9. The server processes this request ([MS-SAMR] section 3.1.5.1.5) 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) with the DesiredAccess parameter set to MAXIMUM_ALLOWED ([MS-SAMR] section 2.2.1.1) to create the new user. The request includes the domain handle received earlier and the user name of the new user to create, and has the AccountType parameter set to USER_NORMAL_ACCOUNT ([MS-SAMR] section 2.2.1.12). 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 that the account can be updated in subsequent steps).

11. The server processes the request ([MS-SAMR] section 3.1.5.4.4), and creates the new user. It creates a user directory object with sAMAccountName set 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).

12. The client application sends a SamrQueryInformationUser request ([MS-SAMR] section 3.1.5.5.6) using the user handle it obtained in the previous step, requesting that UserControlInformation be returned. This is done 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), and returns the value of the userAccountControl attribute (in accordance with the behavior described in [MS-SAMR] sections 3.1.5.4.4 and 3.1.5.14.11) 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) 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) 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), 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) based on administrator-provided values.

17. The server processes the request ([MS-SAMR] section 3.1.5.6.4), 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) 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, the domain handle, and then the server handle).

As the final step, the transports created for communication are closed.

3.2.3 Example 3: Provision a User Account Using the SAMR Protocol Including the Need for a RID Allocation Request

This example demonstrates the communication between a DC in a domain and the RID Master FSMO role owner of the domain in order to obtain a RID range allocation. The RID Master FSMO role owner is responsible for allocating RID ranges that are used by DCs to create new security principals in the directory service.

To perform this task, an administrator runs a client application using the SAMR protocol from a client computer. The client application targets a directory server in the Active Directory System to create a user. However, the DC against which the user is to be created has exhausted its allocated RID range. The DC communicates with the RID Master FSMO role owner to obtain a new RID range.

This example applies only to AD DS.

This example covers the use case in section 2.7.2.1, Create a New Account - Client Application, specifically the extension wherein the DC that is serving the request has exhausted its allocated RID range.

Prerequisites

The general requirements set forth in section 2.6, Assumptions and Preconditions.

The Active Directory System must meet all preconditions described in sections 2.7.1.1 and 2.7.2.1.

Initial System State

The DC serving the request does not have any RIDs left from its allocated range of RIDs.

Final System State

The new user object has been created in the directory.

Sequence of Events

The following sequence diagram depicts the message flow that is associated with this example.

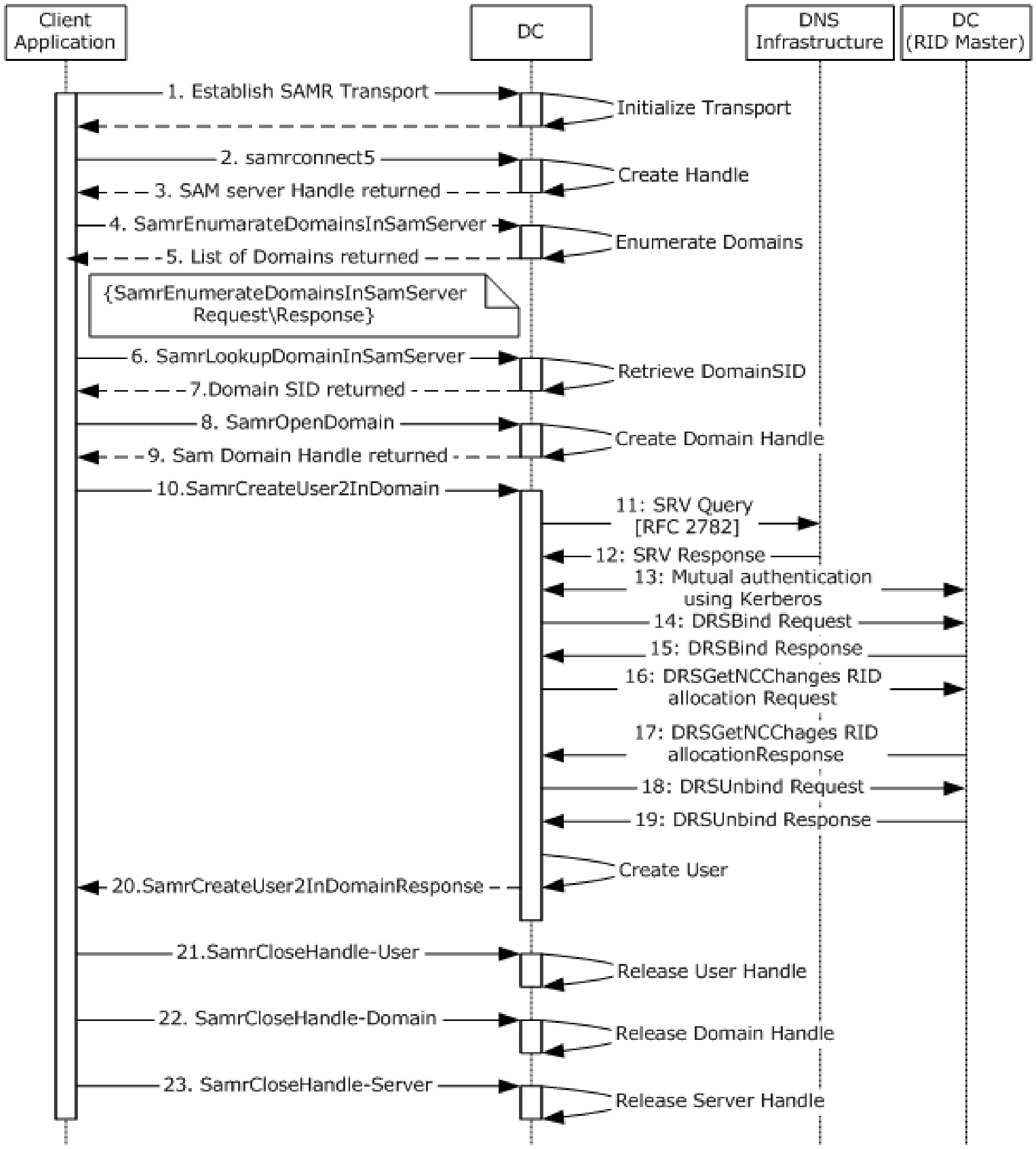

Figure 48: Message flow for provisioning a user account using the SAMR protocol

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.

2. The next step is to open a SAMR handle to the DC (directory server). The client application sends a SamrConnect5 request ([MS-SAMR] section 3.1.5.1.1) with the DesiredAccess parameter set to MAXIMUM_ALLOWED ([MS-SAMR] section 2.2.1.1) to the server, requesting a server handle.

3. The DC processes the request ([MS-SAMR] section 3.1.5.1.1) and sends a response with the server handle to be used in subsequent 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) using the server handle it obtained in step 3.

5. The DC processes the request ([MS-SAMR] section 3.1.5.2.1) 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) 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) using the server handle from step 3. In this request it specifies one of the names that was returned in step 5, which corresponds to the domain object.

7. The DC processes the request ([MS-SAMR] section 3.1.5.11.1) 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) using the server handle it obtained in step 3 and the domain SID it obtained in step 7 with the DesiredAccess parameter set to MAXIMUM_ALLOWED ([MS-SAMR] section 2.2.1.1).

9. The DC processes this request ([MS-SAMR] section 3.1.5.1.5) 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) with the DesiredAccess parameter set to MAXIMUM_ALLOWED ([MS-SAMR] section 2.2.1.1) to create the new user. The request includes the domain handle received in the previous step and the user name of the new user to create, and has the AccountType parameter set to USER_NORMAL_ACCOUNT ([MS-SAMR] section 2.2.1.12). 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 that the account can be updated in subsequent steps).

11-13 When the server receives the request and determines that it has exhausted its allocated RIDs, it will attempt to connect to the owner of the RID Master FSMO role, or RID master. The server obtains the appropriate IP address from DNS and mutually authenticates with the RID master.

14. The DC sends an IDL_DRSBind request, as specified in [MS-DRSR] section 4.1.3, to the RID master.

15. The RID master returns a DRS handle to the DC.

16. The DC sends an IDL_DRSGetNCChanges request, with ulExtendedOp set to EXOP_FSMO_REQ_RID_ALLOC, according to the rules specified in [MS-DRSR] section 4.1.10.4.3.

17. The RID master processes this request according the processing rules specified in [MS-DRSR] section 4.1.10.5 and returns a range of RIDs.

18-19. The DC cleans up the connection by sending an IDL_DRSUnbind request.

20. After having obtained a new allocation of RIDs, the server processes the user-creation request ([MS-SAMR] section 3.1.5.4.4), and creates the new user. It creates a user directory object with the sAMAccountName attribute set 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 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).

21-23. 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) with SamHandle set to the handle the client is attempting to close. The client application closes the handles in the reverse order in which they were created (namely, the user handle, the domain handle, and then the server handle).

As the final step, the transports created for communication are closed.

3.2.4 Example 4: Change a User Account's Password

In this example, 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 example applies only to AD DS.

This example uses the SAMR protocol.

This example covers the use case in section 2.7.2.3, Change an Existing Account's Password (PDC) - Client Application.

Prerequisites

The general requirements set forth in section 2.6, Assumptions and Preconditions.

The Active Directory System must meet all preconditions described in section 2.7.2.3.

Initial System State

None.

Final System State

The user account's password in the directory has been changed to the new value.

Sequence of Events

The following sequence diagram depicts the message flow associated with this example.

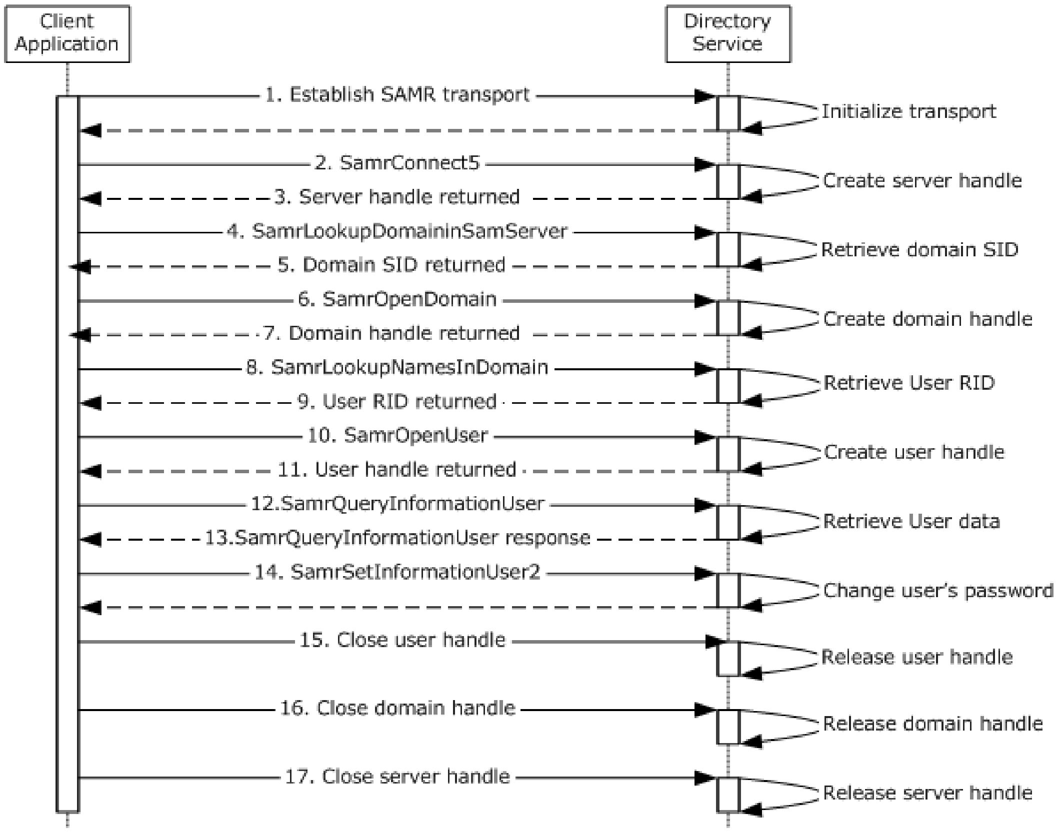

Figure 49: Message flow for changing a user account's password

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 described in [MS-SAMR] section 2.1.

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) to the server with the desired value set in the *DesiredAccess* parameter ([MS-SAMR] section 2.2.1.1). This message requests a server handle.

3. The server processes the request ([MS-SAMR] section 3.1.5.1.1) and sends a response with the server handle to be used in 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 and 5.

4. The client application sends a SamrLookupDomainInSamServer request ([MS-SAMR] section 3.1.5.11.1) using the server handle from step 3. 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) 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) using the server handle it previously obtained in step 3 and the domain SID it obtained in step 5, with the desired value set in the *DesiredAccess* parameter ([MS-SAMR] section 2.2.1.1).

7. The server processes this request ([MS-SAMR] section 3.1.5.1.5) and returns a response with a domain handle.

   The client application must now obtain 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). The request includes the domain handle and the sAMAccountName attribute.

9. The server processes the request ([MS-SAMR] section 3.1.5.11.2) 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). The request includes the domain handle and the RID of the user, and has the desired value set in the *DesiredAccess* parameter ([MS-SAMR] section 2.2.1.1).

11. The server processes the request ([MS-SAMR] section 3.1.5.1.9) and returns a response with a user handle.

12. The client application sends a SamrQueryInformationUser request ([MS-SAMR] section 3.1.5.5.6) to the server to query all user information.

13. The server processes the request and returns a response with all user information.

14. Now that the client application has a handle to the user, it calls SamrSetInformationUser2 to change the user's password. The server processes the request, returns STATUS_SUCESS, and changes the password of the user. Refer to [MS-SAMR] section 3.1.5.6.4.

15-17. 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) with SamHandle set to the handle that the client application is attempting to close. The client application closes the handles in the reverse order in which they were created (that is, the user handle, the domain handle, and then the server handle).

3.2.5 Example 5: Change a User Account's Password Against a Non-PDC DC

In this example, 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 DC in the Active Directory System. The client application changes the account's password using the SAMR protocol.

In this example the client connects to a DC that is not a PDC in order to change the account password. This example demonstrates the communication between a DC and a PDC when a user sends a password change request.

This example applies only to AD DS.

This example uses the SAMR, SAMS, and NRPC protocols.

This example covers the use case in section 2.7.2.4, Change an Existing Account's Password (DC) - Client Application.

Prerequisites

The general requirements set forth in section 2.6, Assumptions and Preconditions.

The Active Directory System must meet all preconditions described in section 2.7.2.4.

Initial System State

None.

Final System State

The user account's password in the DC has been changed to the new value, and it is also updated at the PDC.

Sequence of Events

The following sequence diagram depicts the message flow associated with this example.

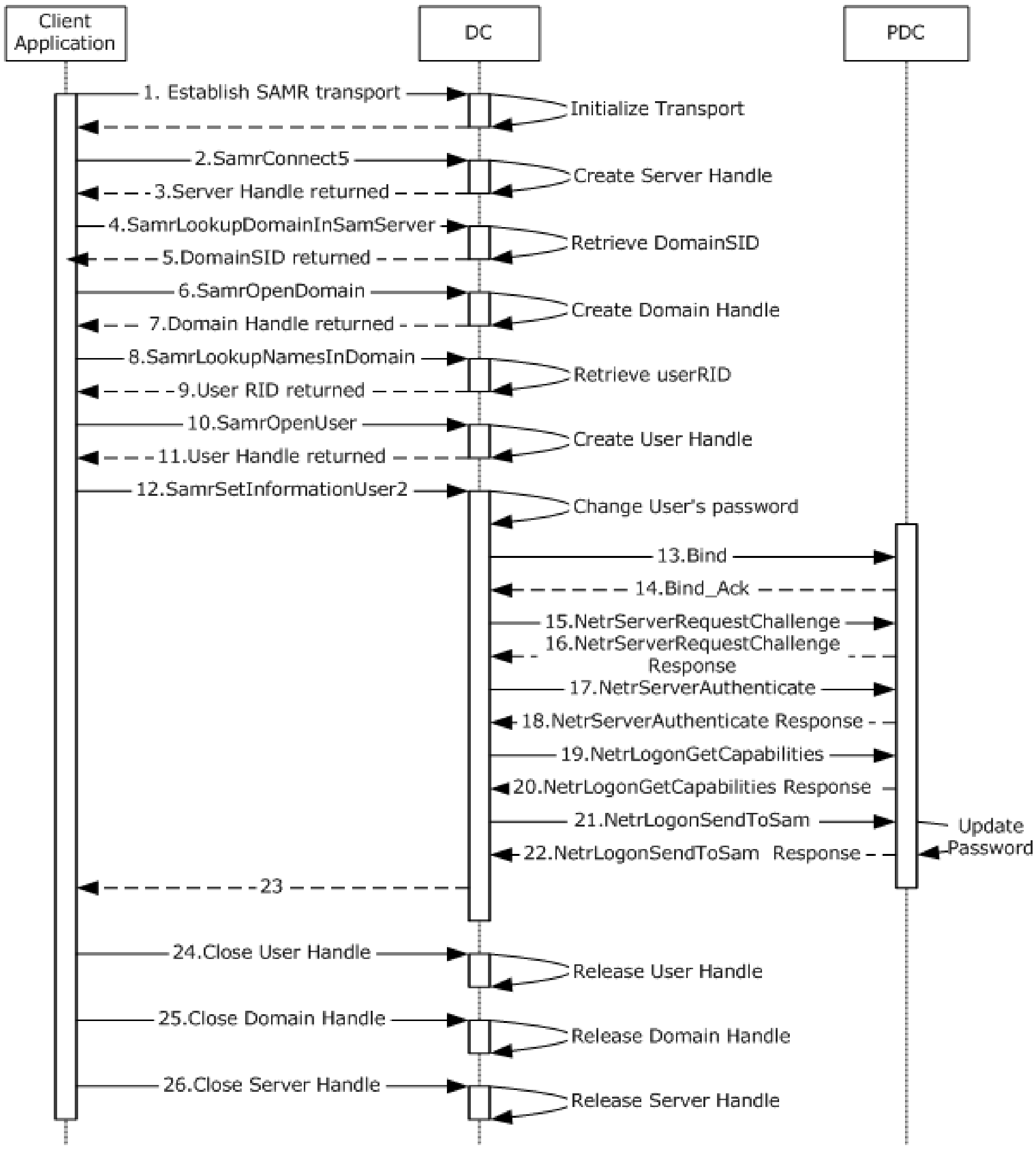

Figure 50: Message flow for changing a user account's password

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 DC using a supported transport, as described in [MS-SAMR] section 2.1.

2. The next step is to open a SAMR handle to the DC's SAMR server. The client application sends a SamrConnect5 request ([MS-SAMR] section 3.1.5.1.1) to the DC with the desired value set in the *DesiredAccess* parameter ([MS-SAMR] section 2.2.1.1). This message requests a server handle.

3. The DC processes the request ([MS-SAMR] section 3.1.5.1.1) and sends a response with the server handle to be used in subsequent calls.

Before continuing by opening a SAMR handle to the domain, the client application must first know the SID of the domain. This is determined using steps 4 and 5.

4. The client application sends a SamrLookupDomainInSamServer request ([MS-SAMR] section 3.1.5.11.1) using the server handle from step 3. In this request it specifies the domain name for the account.

5. The DC processes the request ([MS-SAMR] section 3.1.5.11.1) 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) using the server handle it obtained in step 3 and the domain SID it obtained in step 5 with the desired value set in the *DesiredAccess* parameter ([MS-SAMR] section 2.2.1.1).

7. The DC processes this request ([MS-SAMR] section 3.1.5.1.5) and returns a response with a domain handle.

   The client application must now obtain the 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). The request includes the domain handle and the sAMAccountName attribute.

9. The DC processes the request ([MS-SAMR] section 3.1.5.11.2) 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). The request includes the domain handle and the RID of the user, and has the desired value set in the *DesiredAccess* parameter ([MS-SAMR] section 2.2.1.1).

11. The server processes the request ([MS-SAMR] section 3.1.5.1.9) and returns a response with a user handle.

12. Now that the client application has a handle to the user, it calls SamrSetInformationUser2 to change the user's password. The DC changes the password of the user. Refer to [MS-SAMR] section 3.1.5.6.4.

13-20. The DC establishes a secure channel with the PDC as specified in [MS-NRPC] section 3.1.4.1.

21. The DC sends a password update request to the PDC as specified in [MS-SAMS] section 3.2.4.4 using the NetrLogonSendToSam method defined in [MS-NRPC] section 3.5.4.8.4 and according to the rules in [MS-NRPC] section 3.4.5.6.4.

22. The PDC processes the request according to the rules specified in [MS-SAMS] section 3.3.5.2 and [MS-NRPC] section 3.5.4.8.4 and returns a response.

23. The DC returns a response to the client application.

24-25. 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) with SamHandle set to the handle that the client application is attempting to close. The client application closes the handles in the reverse order in which they were created (that is, the user handle, the domain handle, and then the server handle).

3.2.6 Example 6: Update the User's lastLogOnTimeStamp Against an RODC When the User Binds to an LDAP Server.

In this example, the user's lastLogonTimeStamp attribute is updated when the user authenticates successfully to the LDAP server using an LDAP bind request .

This example applies only to AD DS.

This example uses the SAMS protocol.

This example covers the use case in section 2.7.2.6, User Login to Domain Services Using an RODC and Updating the User LastLogonTimeStamp - Client Application.

The lastLogonTimeStamp attribute is updated upon successful authentication of the user either by using interactive logon or Network logon to the directory system. This example shows user login to the directory using Network logon to the LDAP server.

Prerequisites

The general requirements set forth in section 2.6, Assumptions and Preconditions.

The Active Directory System must meet all preconditions described in section 2.7.2.6.

Initial System State

None.

Final System State

The user's lastLogonTimeStamp attribute is updated.

Sequence of Events

The following sequence diagram depicts the message flow associated with this example.

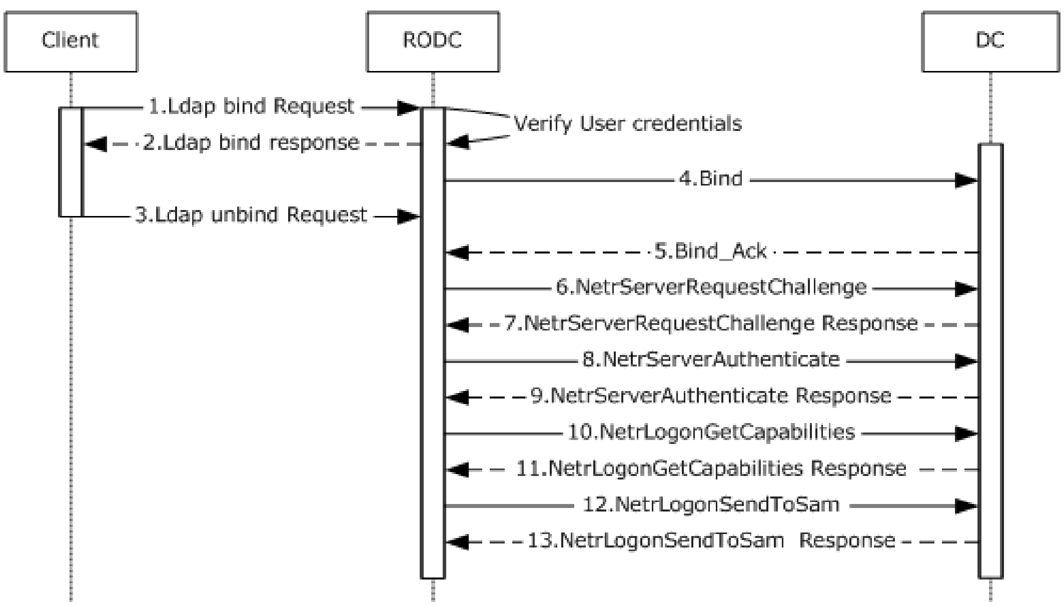

Figure 51: User lastLogonTimeStamp update message flow

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

1. The client starts and an LDAP bind request ([RFC2251] section 4.2) is sent to the RODC with the credentials of the user.

2. The RODC uses one of the methods defined elsewhere (see [MS-AUTHSOD] section 2) to verify the credentials. Depending on the negotiated authentication method, this might involve additional client and server interactions that are not directly relevant to this discussion. After verification, the directory server sends an LDAP bind response ([RFC2251] section 4.2.3) to the client.

3. The client sends an unbind request to clean up the bind operation. This step can occur in any order after step 2; it is not dependent on the timing of the subsequent steps in this example.

4-11. In these steps, the RODC establishes a secure channel with a DC that contains a writable NC replica of the domain, as specified in [MS-NRPC] section 3.1.4.1.

12. The RODC sends a password update request to the DC, as specified in the [MS-SAMS] section 3.2.4.6, using the NetrLogonSendToSam method defined in [MS-NRPC] section 3.5.4.8.4, and according the processing rules in [MS-NRPC] section 3.4.5.6.4.

13. The DC processes the request ([MS-SAMS] section 3.3.5.6 and [MS-NRPC] section 3.5.4.8.4), updates the user account's lastLogonTimeStamp attribute, and returns a response.

3.2.7 Example 7: Determine the Group Membership of a User

In this example, 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 example applies only to AD DS.

This example uses the SAMR protocol.

This example covers the use case in section 2.7.2.7, Query an Account's Group Membership - Client Application.

Prerequisites

The general requirements set forth in section 2.6, Assumptions and Preconditions.

The Active Directory System must meet all preconditions described in section 2.7.2.7.

Initial System State

None.

Final System State

The user's group membership information has been returned to the client application.

Sequence of Events

The following sequence diagram depicts the message flow associated with this example.

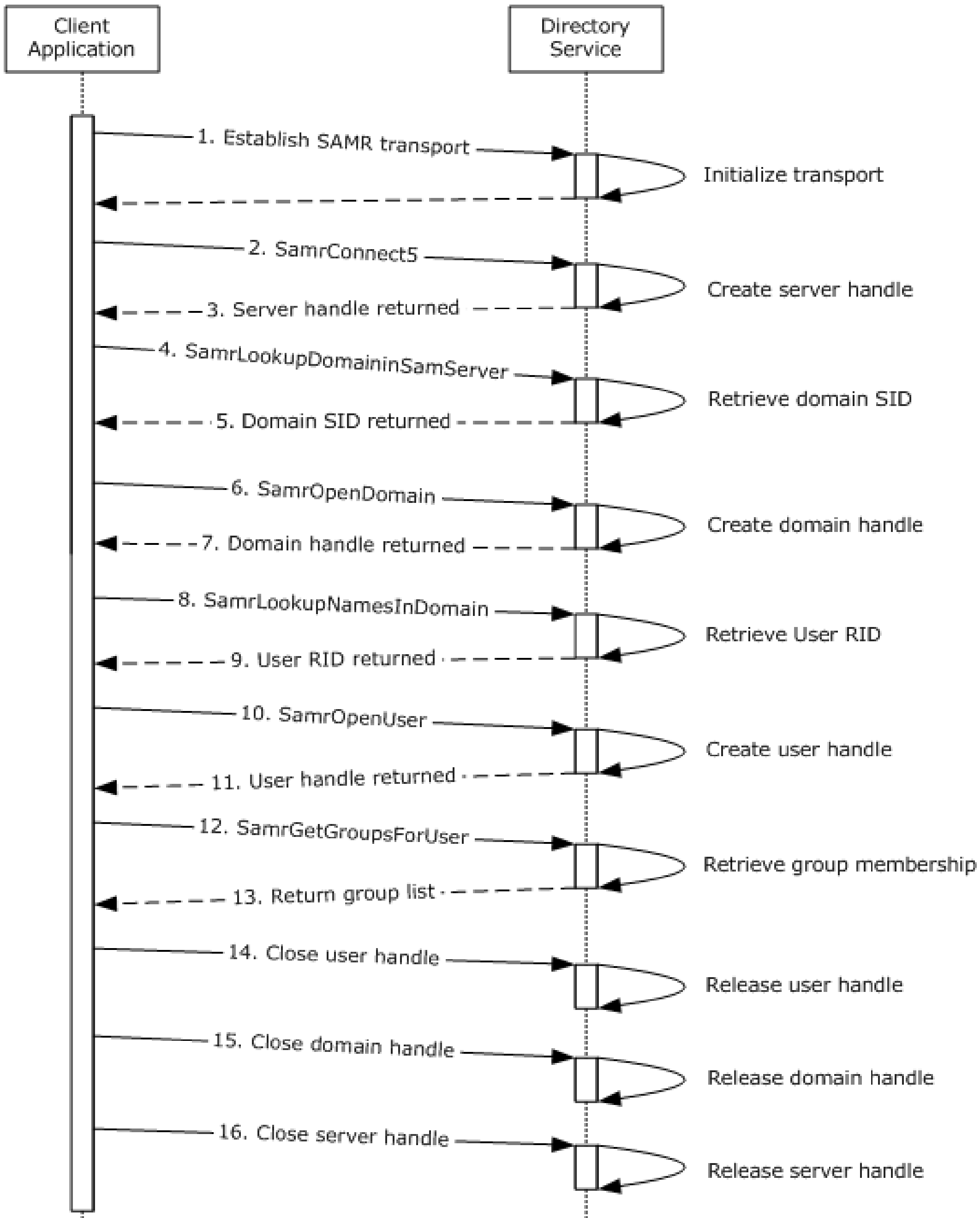

Figure 52: Message flow for determining the group membership of a user

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 described in [MS-SAMR] section 2.1.

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) with the *DesiredAccess* parameter set to MAXIMUM_ALLOWED ([MS-SAMR] section 2.2.1.1) to the server, requesting a server handle.

3. The server processes the request ([MS-SAMR] section 3.1.5.1.1), 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 and 5.

4. The client application sends a SamrLookupDomainInSamServer request ([MS-SAMR] section 3.1.5.11.1) using the server handle from step 3. 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), 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) using the server handle it previously obtained in step 3 and the domain SID it obtained in step 5, with the *DesiredAccess* parameter set to MAXIMUM_ALLOWED ([MS-SAMR] section 2.2.1.1).

7. The server processes this request ([MS-SAMR] section 3.1.5.1.5), and returns a response with a domain handle.

   The client application must now obtain 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). The request includes the user name and the domain handle.

9. The server processes the request ([MS-SAMR] section 3.1.5.11.2) 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) with the *DesiredAccess* parameter set to MAXIMUM_ALLOWED ([MS-SAMR] section 2.2.1.1). 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), 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) to retrieve the group membership of the user.

13. The server processes the request ([MS-SAMR] section 3.1.5.9.1), 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) with SamHandle set to the handle that the client application is attempting to close. The client application closes the handles in the reverse order in which they were received (that is, the user handle, the domain handle, and then the server handle).

3.2.8 Example 8: Delete a User Account

In this example, 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 account by using the LDAP protocol.

This example applies only to AD DS.

This example uses the LDAP protocol.

This example covers the use case in section 2.7.2.8, Delete an Account - Client Application.

Prerequisites

The general requirements set forth in section 2.6, Assumptions and Preconditions.

The Active Directory System must meet all preconditions described in section 2.7.2.8.

Initial System State

None.

Final System State

The specified user object is successfully converted into a **tombstone** or **deleted-object**, depending on whether the recycle bin optional feature is enabled, as described in [MS-ADTS] sections 3.1.1.5.5.1.1 and 3.1.1.5.5.1.2.

Sequence of Events

The following sequence diagram depicts the message flow associated with this example.

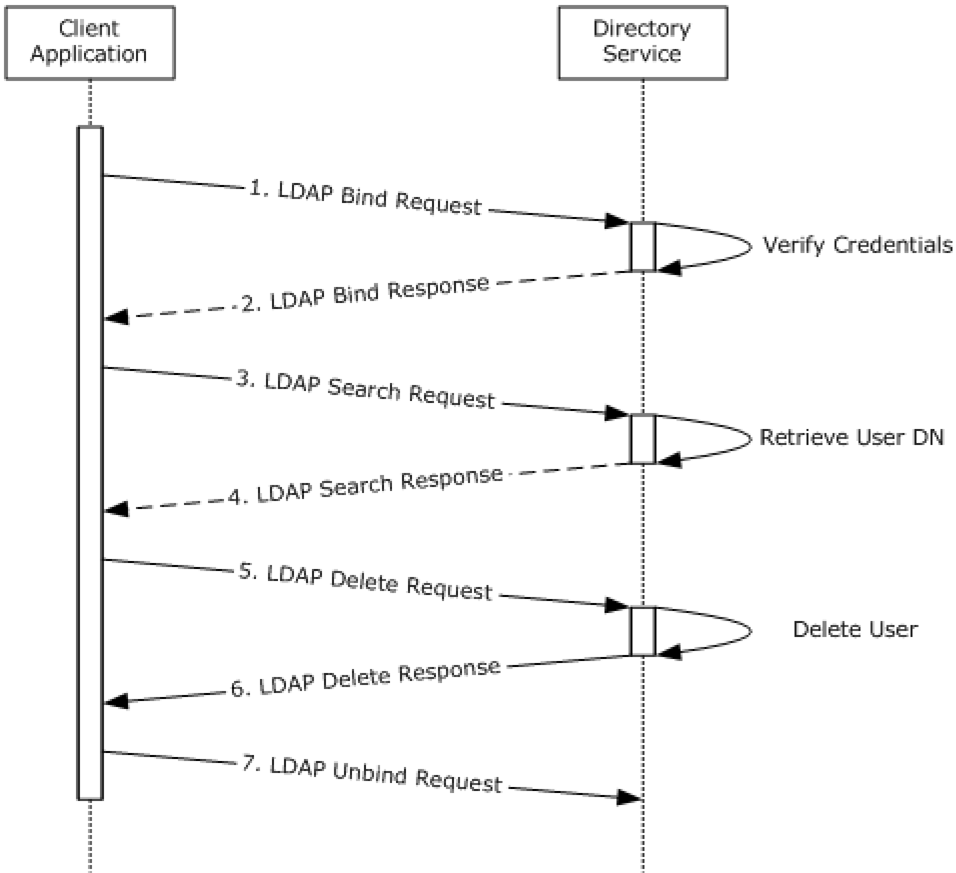

Figure 53: Message flow for deleting a user account

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 starts and an LDAP bind request ([RFC2251] section 4.2) is sent to the directory server with credentials.

2. The directory server verifies the credentials ([MS-AUTHSOD] section 2), and sends an LDAP bind response ([RFC2251] section 4.2.3) 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 ([RFC2251] section 4.5.1) to the server, querying the entire domain, starting at the root of the domain, looking for the user ([MS-ADSC] section 2.241) and requesting the user's distinguishedName attribute.

4. The server sends an LDAP search response ([RFC2251] section 4.5.2) containing the distinguishedName attribute of the user.

5. The client application sends an LDAP delete request ([RFC2251] section 4.8) to the server that contains the distinguishedName attribute of the user to be deleted.

6. The server processes the delete request ([RFC2251] section 4.8), verifies the processing rules and constraints, and then deletes the user object ([MS-ADTS] section 3.1.1.5.5). It then sends an LDAP delete response ([RFC2251] section 4.8) indicating success.

7. The client application sends an LDAP unbind request ([RFC2251] section 4.3) to the server. The LDAP connection to the directory server is closed.

3.2.9 Example 9: 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 [WSENUM]. 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 example uses the WS-Enumeration protocol to communicate with the directory service.

This example covers the use case in section 2.7.1.2, Search for Directory Object - Client Application.

Prerequisites

The general requirements set forth in section 2.6, Assumptions and Preconditions.

The Active Directory System must meet all preconditions described in section 2.7.1.2.

Initial System State

The system must support the Web Services protocols (see section 2.8).

Final System State

The requested information for the user object(s) is returned to the client application.

Sequence of Events

The following sequence diagram depicts the message flow associated with this example.

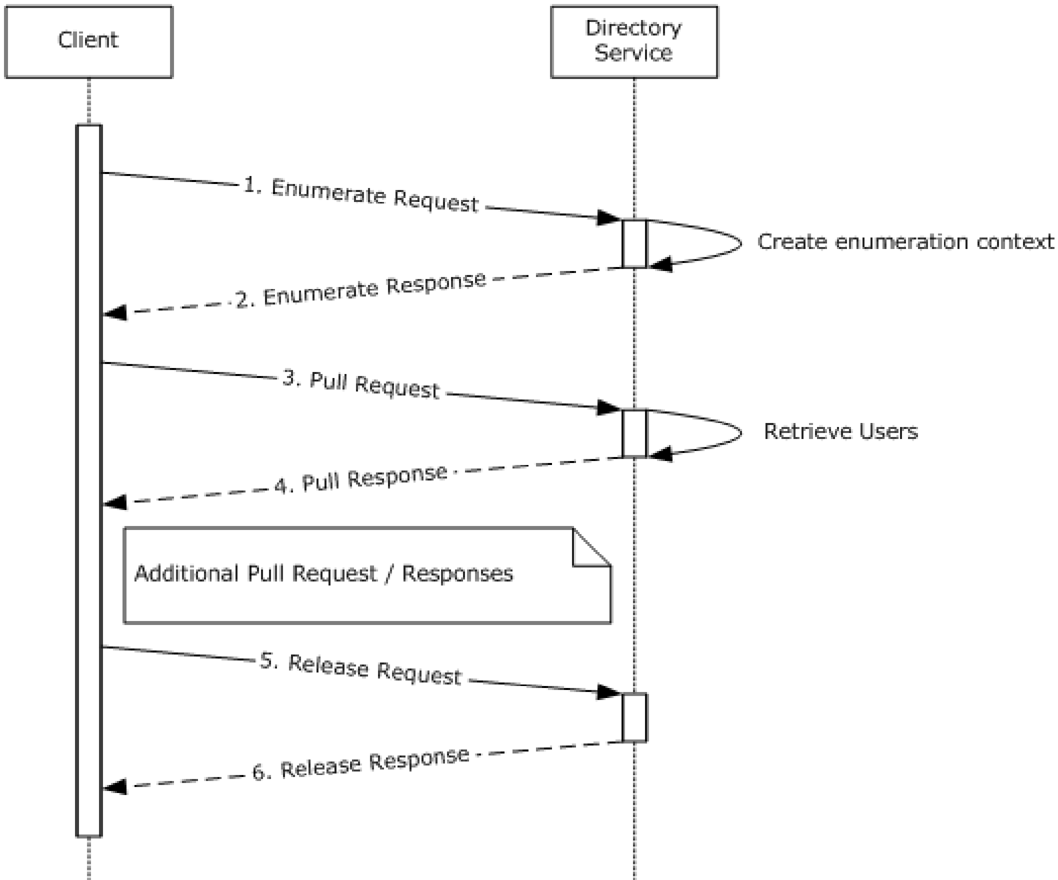

Figure 54: Communication flow for obtaining a list of user accounts using the Web Services protocols

The client application establishes a connection to the directory service using the net.tcp transport. In this example, 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). 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). 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).

2. The directory server processes the request, and generates an enumeration context ([WSENUM] section 3) 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). 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) from the directory. It returns a SOAP message containing a Pull response to the client application ([WSENUM] section 3.2). 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).

5. Having retrieved all the data that was requested, the client application terminates the enumeration by sending a SOAP message containing a Release request ([WSENUM] section 3.5). This tells the server that 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.

   Note If the wsen:EndOfSequence element is obtained by the client in the Pull response, then the Release request is not sent.

6. The server performs any necessary processing of the Release request and returns a Release response to the client.

3.2.10 Example 10: 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 example covers the use case in section 2.7.1.2, Search for Directory Object - Client Application.

Prerequisites

The general requirements set forth in section 2.6, Assumptions and Preconditions.

The Active Directory System must meet all preconditions described in section 2.7.1.2.

Initial System State

None.

Final System State

The requested information for the user object(s) is returned to the client application.

Sequence of Events

The following sequence diagram depicts the message flow associated with this example.

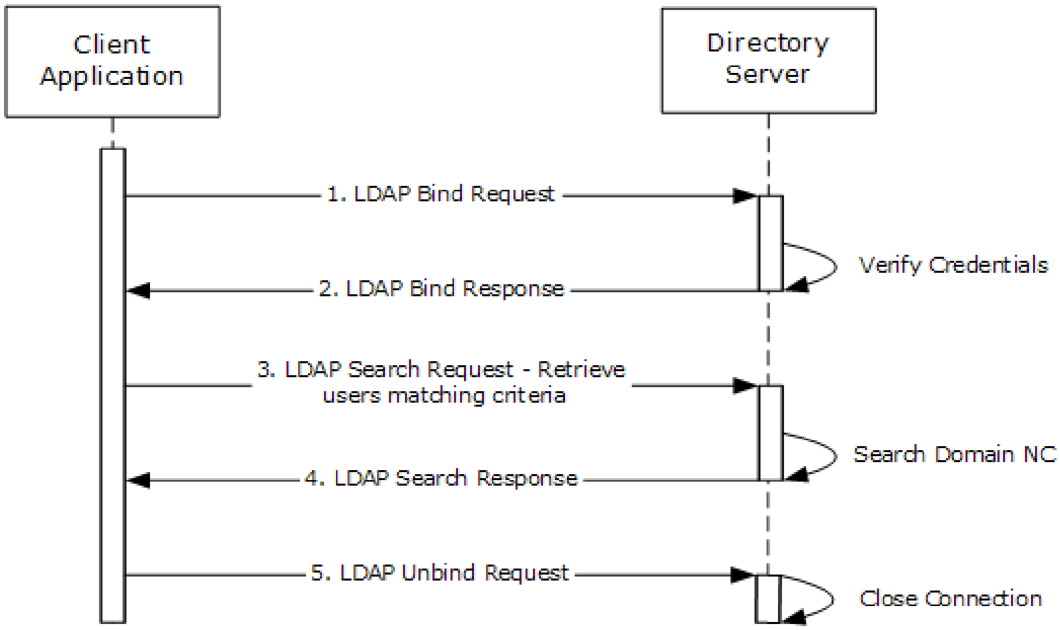

Figure 55: Message flow for obtaining a list of user accounts using the LDAP protocol

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 starts and an LDAP bind request ([RFC2251] section 4.2) is sent to the directory server with credentials.

2. The directory server verifies the credentials ([MS-AUTHSOD] section 2), and sends an LDAP bind response ([RFC2251] section 4.2.3) 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 ([RFC2251] section 4.5.1) to the directory, querying the entire domain, starting at the root of the domain, looking for users ([MS-ADSC] section 2.241 or [MS-ADLS] section 3.62) and requesting all attributes.

4. The server sends an LDAP search response ([RFC2251] section 4.5.2) containing the list of users under the domain NC. The client application organizes this information and displays it to the user. Search filters and results are additionally validated by the server's processing rules and constraints described in [MS-ADTS] sections 3.1.1.3.1.3 and 3.1.1.3.4.6.

5. The client application sends an LDAP unbind request ([RFC2251] section 4.3) to the server. The LDAP connection to the directory server is closed.

3.2.11 Example 11: Manage Groups and Their Memberships

This section discusses the process of creating a group, 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 example covers the use cases in sections 2.7.2.9, Create a Security Group - Client Application, 2.7.2.10, Modify Group Member List - Client Application, and 2.7.2.11, Query Members of a Group - Client Application.

Prerequisites

The general requirements set forth in section 2.6, Assumptions and Preconditions.

The Active Directory System must meet all preconditions described in section 2.7.1.3.

Initial System State

None.

Final System State

The new group object has been created and provisioned in the directory, and updated with the specified membership.

Sequence of Events

The following sequence diagram depicts the message flow associated with this example.

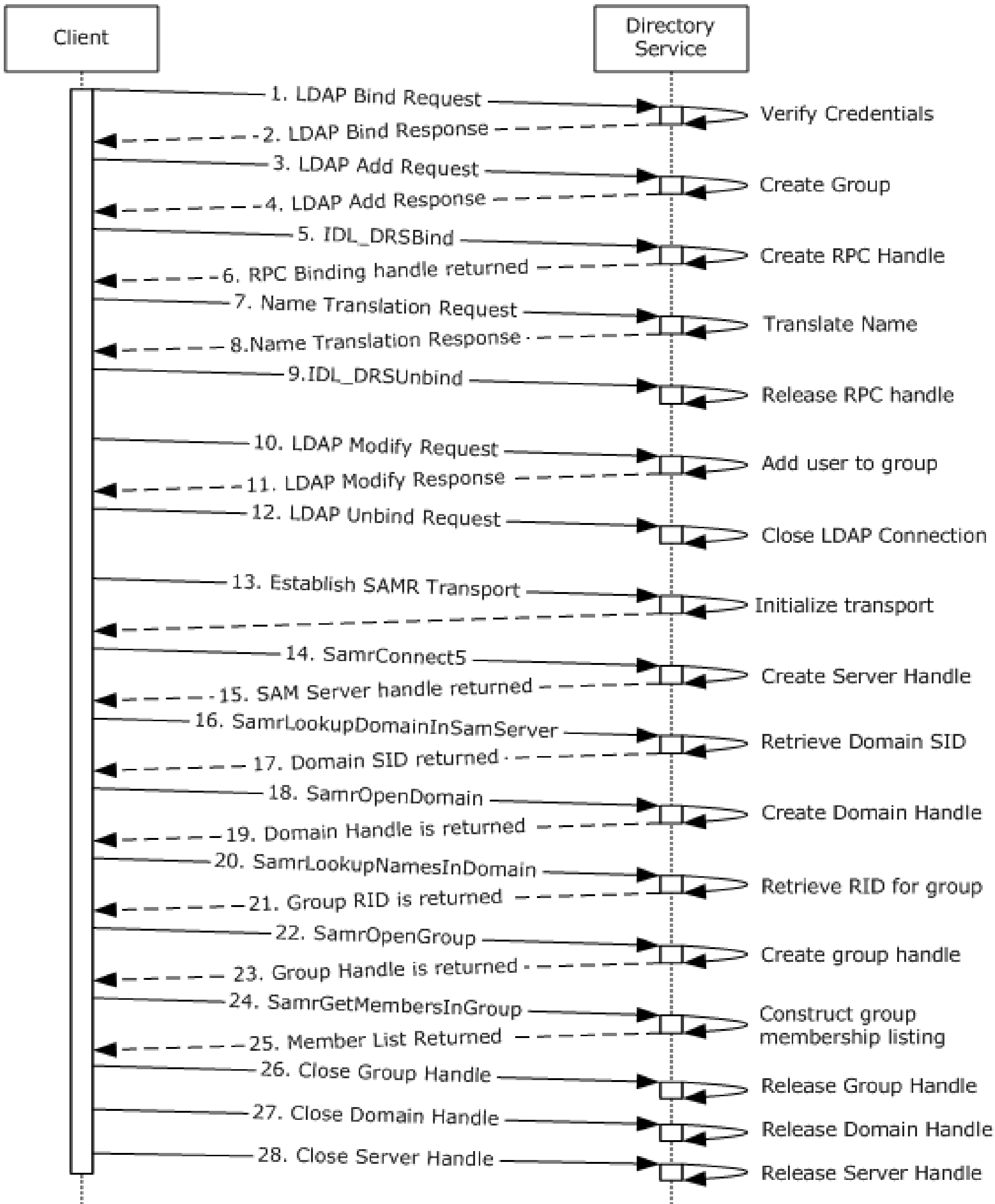

Figure 56: Communication flow for managing groups and their memberships

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 establishes an LDAP connection to the directory server. An LDAP bind request ([RFC2251] section 4.2) is sent to the directory server with the credentials of an administrator.

2. The directory server verifies the credentials ([MS-AUTHSOD] section 2), and sends an LDAP bind response ([RFC2251] section 4.2.3) to the client application.

3. An LDAP add request ([RFC2251] section 4.7) 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) and performs validation, as described in [MS-ADTS] sections 3.1.1.5.1 and 3.1.1.5.2. 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 DRSR protocol, as defined in [MS-DRSR] section 4.1.3.

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. 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.

9. The client requests to release the RPC binding handle it received in step 6, as defined in [MS-DRSR] section 4.1.25. The server processes that request as described in [MS-DRSR] section 4.1.25.1.

10. An LDAP modify request is sent to the server to add the user to the group, as described in [RFC2251] section 4.6. The distinguished name of the user is added to the multivalued linked attribute "member" of the group object ([MS-ADA2] section 2.43 or [MS-ADLS] section 2.136).

11. The server processes the request and sends a response ([RFC2251] section 4.6).

12. The client sends an LDAP unbind request ([RFC2251] section 4.3) 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 of that group with only the information about that user's NT4 account name. The remainder of this example 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.

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) to the server, with the *DesiredAccess* parameter set to MAXIMUM_ALLOWED ([MS-SAMR] section 2.2.1.1), 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 and 17.

16. The client application sends a SamrLookupDomainInSamServer request ([MS-SAMR] section 3.1.5.11.1) 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) by using the server handle that it previously obtained in step 15 and the domain SID it obtained in step 17, with the *DesiredAccess* parameter set to MAXIMUM_ALLOWED ([MS-SAMR] section 2.2.1.1).

19. The server processes this request and returns a response with a domain handle.

    The client application must now obtain 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). 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). The request includes the domain handle and the RID of the group, with the *DesiredAccess* parameter set to MAXIMUM_ALLOWED ([MS-SAMR] section 2.2.1.1).

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 group, it calls SamrGetMembersInGroup ([MS-SAMR] section 3.1.5.8.3) to retrieve the group's member list.

25. The server processes the request 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) with SamHandle set to the handle that the client application is attempting to close. The client application closes the handles in the reverse order in which they were received (that is, the group handle, the domain handle, and then the server handle).

3.2.12 Example12: Delete a Group

In this example, 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 example covers the use case in section 2.7.1.4, Delete Directory Object - Client Application.

Prerequisites

The general requirements set forth in section 2.6, Assumptions and Preconditions.

The Active Directory System must meet all preconditions specified in section 2.7.1.4.

Initial System State

None.

Final System State

The security group object is successfully converted into a tombstone or deleted-object, depending upon whether the Recycle Bin optional feature is enabled, as described in [MS-ADTS] sections 3.1.1.5.5.1.1 and 3.1.1.5.5.1.2.

Sequence of Events

The following sequence diagram depicts the message flow associated with this example.

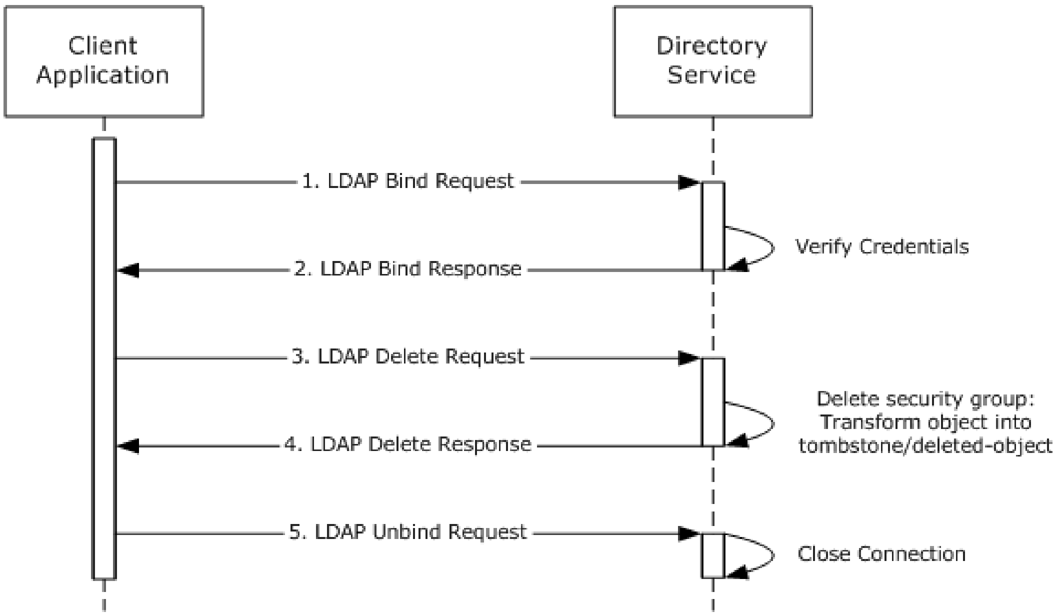

Figure 57: Message flow for deleting a security group

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 starts and an LDAP bind request ([RFC2251] section 4.2) is sent to the directory server with credentials.
2. The directory server verifies the credentials ([MS-AUTHSOD] section 2), and sends an LDAP bind response ([RFC2251] section 4.2.3) 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) 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) 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 and 3.1.1.5.5. It then sends an LDAP delete response ([RFC2251] section 4.8) indicating success.
5. The client application sends an LDAP unbind request ([RFC2251] section 4.3) to the server. The LDAP connection to the directory server is closed.

3.2.13 Example 13: Extend the Schema to Support an Application by Adding a New Class

In this example, an administrator extends the schema by adding a class that is 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 that owns 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 example uses the LDAP protocol.

This example covers the use case in section 2.7.3.1, Add a New Class to the Schema - Client Application.

Prerequisites

The general requirements set forth in section 2.6, Assumptions and Preconditions.

The Active Directory System must meet all preconditions described in section 2.7.3.1.

Initial System State

None.

Final System State

A new class has been added to the schema.

Sequence of Events

The diagram that follows illustrates the messages that are exchanged between a client and a directory server when successfully adding a class to the schema.

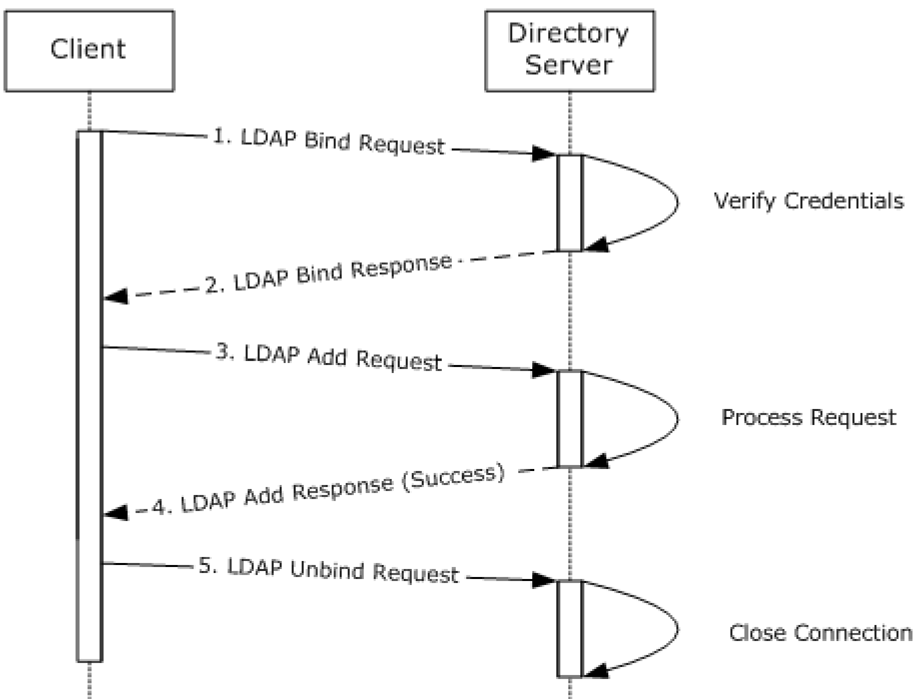

Figure 58: Message flow for extending the schema by adding a class

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) is sent to the directory server with the credentials of the administrator.

2. The directory server verifies the credentials ([MS-AUTHSOD] section 2), and sends an LDAP bind response ([RFC2251] section 4.2.3) to the client application.

3. The client application sends an LDAP add request ([RFC2251] section 4.7) to the server. The request contains the values of the mandatory attributes ([MS-ADTS] section 3.1.1.2.4.8) for the new object of class classSchema.

4. The server verifies all the processing rules and constraints ([MS-ADTS] sections 3.1.1.2.5, 3.1.1.5.1, and 3.1.1.5.2). Upon success, the class is added to the schema and the server sends an LDAP add response ([RFC2252] section 4.7) 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) to the directory server.

3.2.14 Example 14: Extend the Schema to Support an Application by Adding a New Attribute

In this example, an administrator extends the schema by adding an attribute that is 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 that owns 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 example uses the LDAP protocol.

This example covers the use case in section 2.7.3.2, Add a New Attribute to the Schema - Client Application.

Prerequisites

The general requirements set forth in section 2.6, Assumptions and Preconditions.

The Active Directory System must meet all preconditions described in section 2.7.3.2.

Initial System State

None.

Final System State

A new attribute has been added to the schema.

Sequence of Events

The diagram that follows illustrates the messages that are exchanged between a client application and a directory server when successfully adding an attribute to the schema.

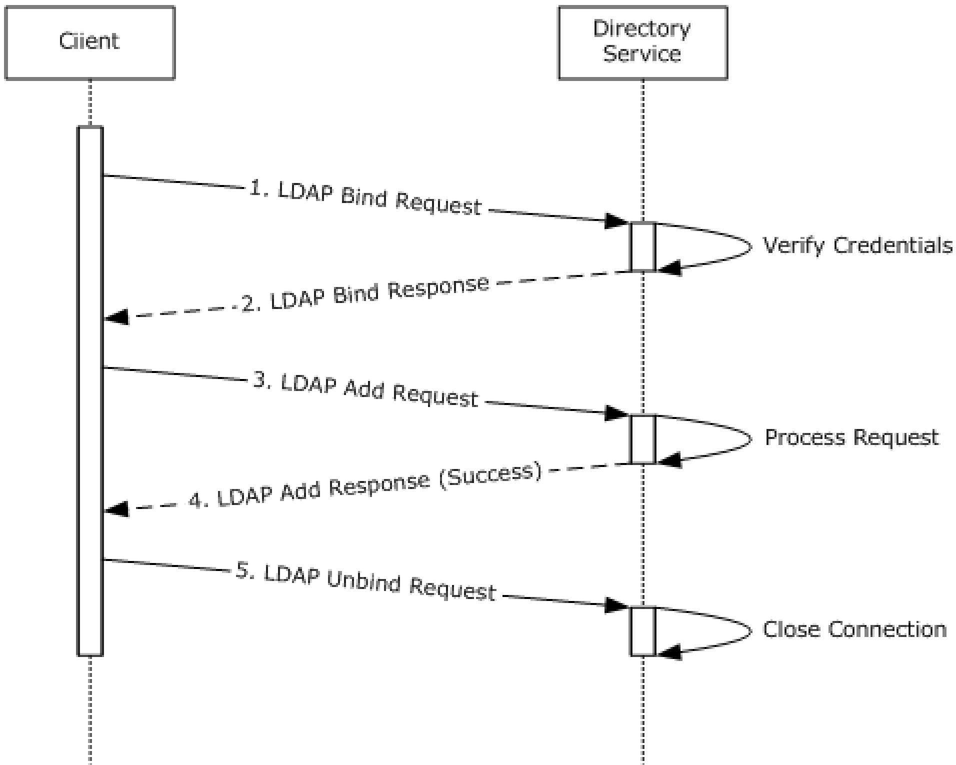

Figure 59: Message flow for extending the schema by adding an attribute

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) is sent to the directory server with the credentials of the administrator.

2. The directory server verifies the credentials ([MS-AUTHSOD] section 2), and sends an LDAP bind response ([RFC2251] section 4.2.3) to the client application.

3. The client application sends an LDAP add request ([RFC2251] section 4.7) to the server. The request contains the values of the mandatory attributes ([MS-ADTS] section 3.1.1.2.3) for the new object of class attributeSchema.

4. The server verifies all the processing rules and constraints ([MS-ADTS] section 3.1.1.2.5, 3.1.1.5.1, and 3.1.1.5.2). 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) 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) to the directory server.

3.2.15 Example 15: Extend the Schema to Support an Application by Adding an Attribute to a Class

In this example, an administrator extends the schema by adding an attribute that is 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 that owns the Schema Master FSMO role in the Active Directory System. The client application adds an attribute to a class using the LDAP protocol.

This example uses the LDAP protocol.

This example covers the use case in section 2.7.3.3, Add an Attribute to a Class - Client Application.

Prerequisites

The general requirements set forth in section 2.6, Assumptions and Preconditions.

The Active Directory System must meet all preconditions described in sections 2.7.1.1 and 2.7.3.3.

Initial System State

None.

Final System State

The schema has been modified to add the specified attribute as a valid attribute for the specified class.

Sequence of Events

The diagram that follows illustrates the messages that are exchanged between a client and a directory server when successfully adding an attribute to a class.

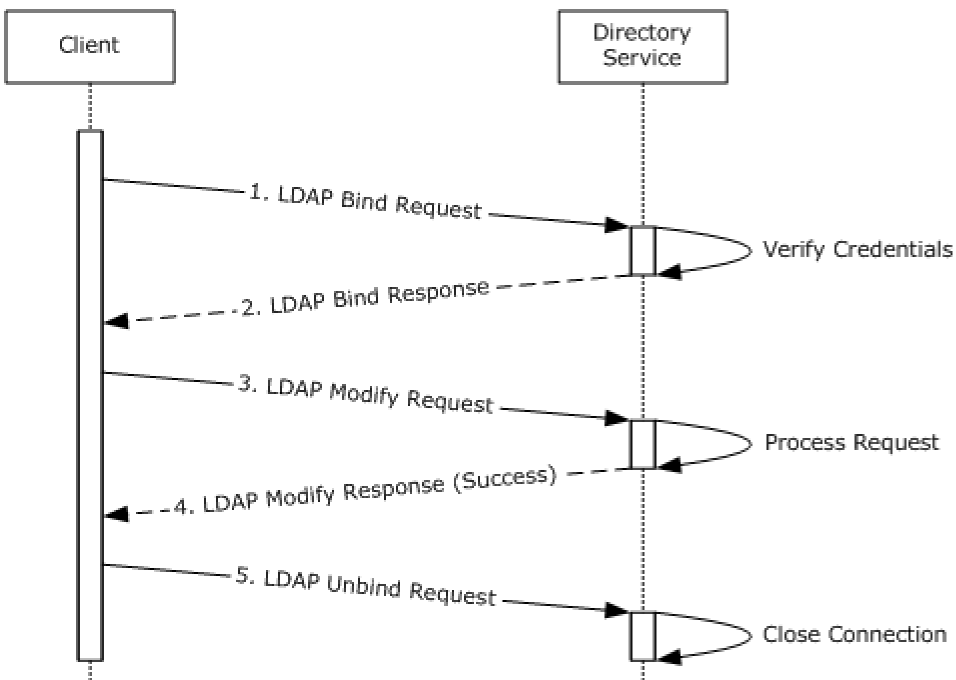

Figure 60: Message flow for extending the schema by adding an attribute to a class

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) is sent to the directory server with the credentials of the administrator.

2. The directory server verifies the credentials ([MS-AUTHSOD] section 2), and sends an LDAP bind response ([RFC2251] section 4.2.3) to the client application.

3. The client application sends an LDAP modify request ([RFC2251] section 4.6) 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 2.294 (or [MS-ADLS] section 2.356) or mayContain as defined in [MS-ADA2] section 2.18 (or [MS-ADLS] section 2.135)).

4. The server verifies all the processing rules and constraints ([MS-ADTS] section 3.1.1.2.5, 3.1.1.5.1, and 3.1.1.5.3). Upon success, the attribute is added to the class and the server sends an LDAP add response ([RFC2252] section 4.7) 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) to the directory server.

3.2.16 Example 16: Partition Directory Data with Organizational Units

In this example, a user partitions the directory data using organizational units (OUs). 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 example covers the use cases in sections 2.7.1.3, Modify Directory Object - Client Application, and 2.7.1.5, Create Organizational Unit - Client Application.

Prerequisites

The general requirements set forth in section 2.6, Assumptions and Preconditions.

The Active Directory System must meet all preconditions described in sections 2.7.1.3 and 2.7.1.5.

Initial System State

None.

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.

Sequence of Events

The following sequence diagram depicts the message flow associated with this example.

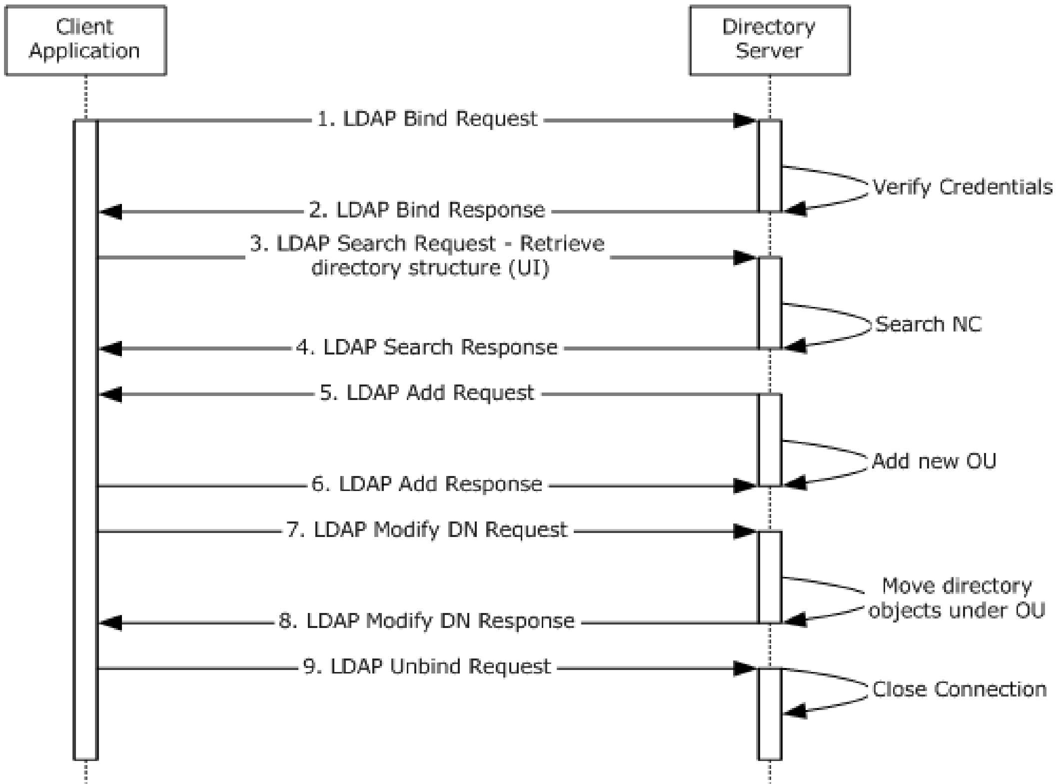

Figure 61: Message flow for partitioning directory data

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 starts, and an LDAP bind request ([RFC2251] section 4.2) is sent to the directory server with credentials.

2. The directory server verifies the credentials ([MS-AUTHSOD] section 2) and sends an LDAP bind response ([RFC2251] section 4.2.3) to the client application.

3. The client application sends an LDAP search request ([RFC2251] section 4.5.1) to the server. The application requests the sub-tree contents of the domain NC or application NC for AD DS or the application NC for AD LDS.

4. The server sends an LDAP search response ([RFC2251] section 4.5.2) containing the list of objects under the NC that is to be used to populate data in the client application's user interface. This step is necessary only for user-interface display purposes specific to this example.

5. The user selects a parent directory object under which the new organizational unit (OU) will be located and provides the name of the new organizational unit to the client application. An LDAP add request ([RFC2251] section 4.7) 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 2.190 or [MS-ADLS] section 3.44).

6. The server processes the add request ([RFC2251] section 4.7), and verifies the processing rules and constraints as described in [MS-ADTS] sections 3.1.1.5.1 and 3.1.1.5.2. It then sends an LDAP add response ([RFC2251] section 4.7) indicating success.

7. The user selects directory object(s) to be moved under the new OU. The directory object(s) are moved by sending a series of LDAP modify DN requests ([RFC2251] section 4.9) 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) for each directory object, and verifies the processing rules and constraints as described in [MS-ADTS] sections 3.1.1.5.1 and 3.1.1.5.4. For each directory object that was moved, the directory server sends an LDAP modify DN response ([RFC2251] section 4.9) to the client application indicating success.

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

3.2.17 Example 17: 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 example covers the use case in section 2.7.1.1, Create Directory Object - Client Application.

Prerequisites

The general requirements set forth in section 2.6, Assumptions and Preconditions.

The Active Directory System must meet all preconditions described in section 2.7.1.1.

Initial System State

None.

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.

Sequence of Events

The following sequence diagram depicts the message flow associated with this example.

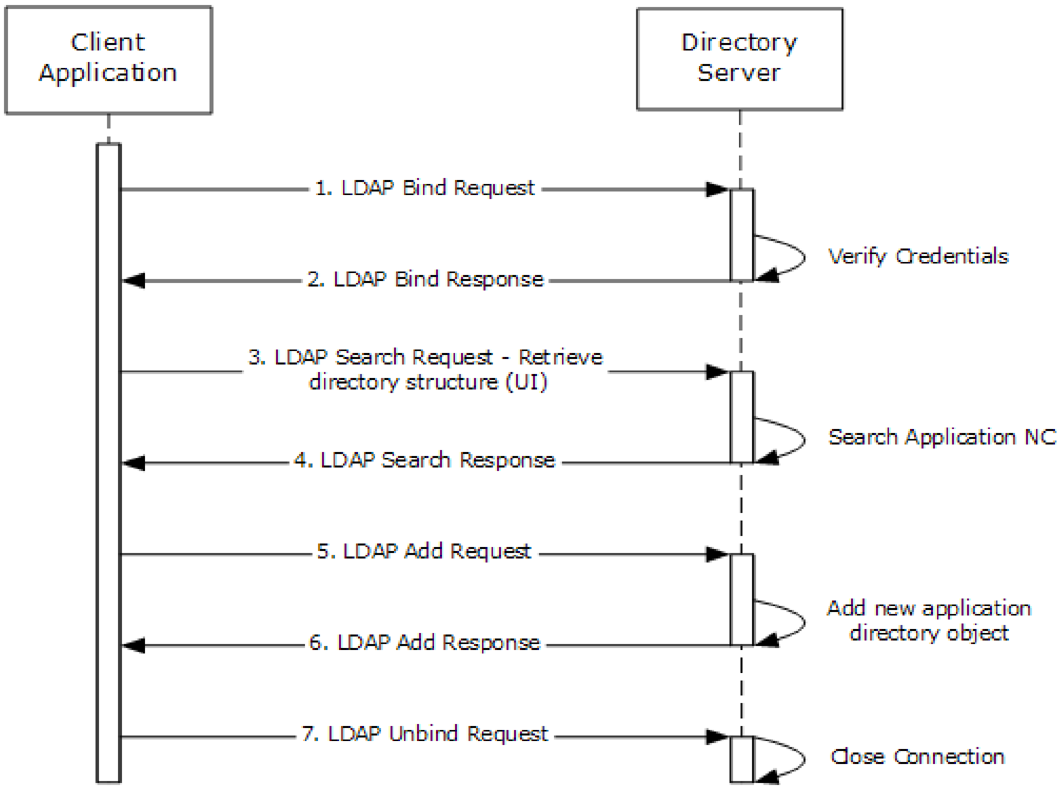

Figure 62: Message flow for storing application data in the directory

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 starts, and an LDAP bind request ([RFC2251] section 4.2) is sent to the directory server with credentials.

2. The directory server verifies the credentials ([MS-AUTHSOD] section 2), and sends an LDAP bind response ([RFC2251] section 4.2.3) to the client application.

3. The client application sends an LDAP search request ([RFC2251] section 4.5.1) to the server. The application requests the sub-tree contents of the application NC.

4. The server sends an LDAP search response ([RFC2251] section 4.5.2) containing the list of objects under the application NC. The client application organizes this information and displays it to the user. Steps 3 and 4 are artifacts of the client application.

5. The user selects a parent directory object under which the new application directory object will be located, and all the necessary data is provided in preparation for the creation of the directory object. An LDAP add request ([RFC2251] section 4.7) is sent to the directory server. The LDAP add operation contains the distinguished name (DN), the object class of the object to be created, and any other mandatory parameters specified in [MS-ADTS] section 3.1.1.2.4.5. Security principal objects can be created only 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.

6. The server processes the add request ([RFC2251] section 4.7) and verifies the processing rules and constraints described in [MS-ADTS] sections 3.1.1.5.1 and 3.1.1.5.2. It then sends an LDAP add response ([RFC2251] section 4.7) to the client application indicating success.

7. The client application sends an LDAP unbind request ([RFC2251] section 4.3) to the server. The LDAP connection to the directory server is closed.

3.2.18 Example 18: Manage Access Control on Directory Objects

In this example, 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 example applies only to AD DS.

This example covers the use case in section 2.7.4.1, Convert a SID to/from a Human-Readable Format - Client Application.

Prerequisites

The general requirements set forth in section 2.6, Assumptions and Preconditions.

The Active Directory System must meet all preconditions described in section 2.7.4.1.

Initial System State

None.

Final System State

The **access control entries (ACEs)** on the object that is specified by the administrator have been updated to secure that object as specified by the administrator.

Sequence of Events

The following sequence diagram depicts the message flow associated with this example.

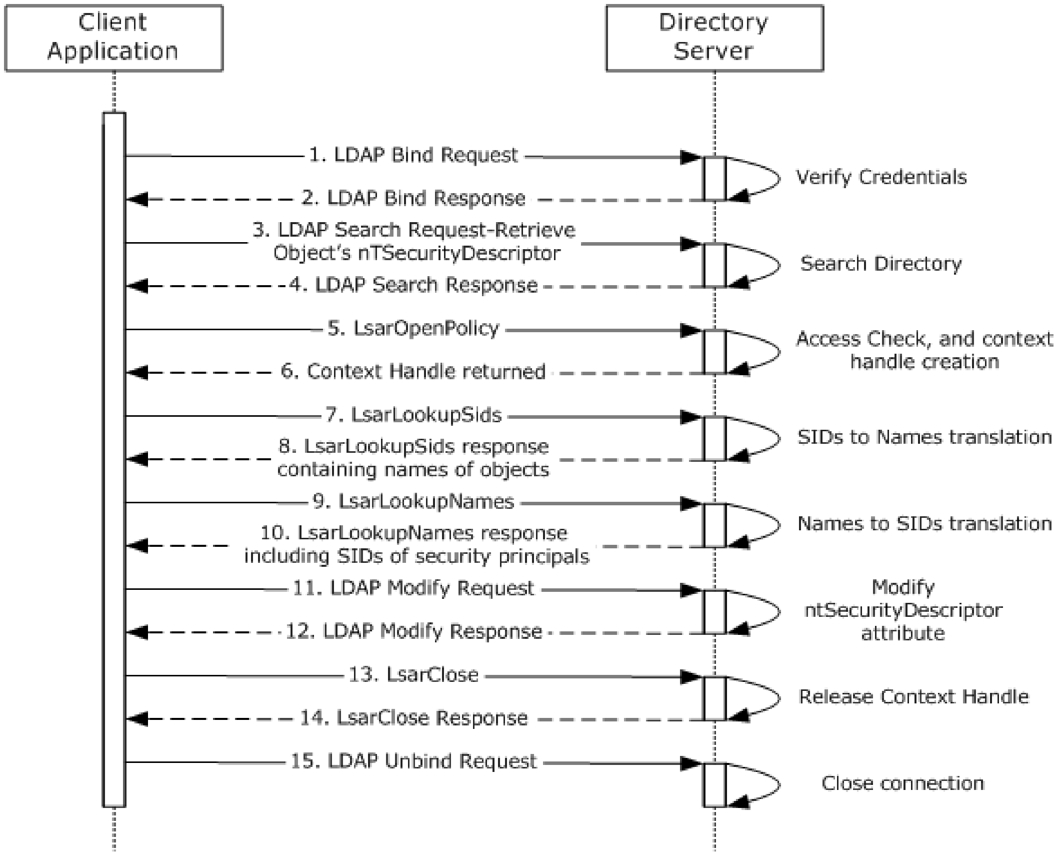

Figure 63: Message flow for managing access control on a directory object

Unless otherwise noted, all responses that include a return code contain a return code indicating that the operation was successfully performed. There is no dependency between LDAP and LSART messages in this scenario. Those messages can come in any order.

1. The client application starts and an LDAP bind request ([RFC2251] section 4.2) is sent to the directory server with credentials.

2. The directory server verifies the credentials ([MS-AUTHSOD] section 2), and sends an LDAP bind response ([RFC2251] section 4.2.3) to the client application.

3. An LDAP search request ([RFC2251] section 4.5.1) is sent to the directory, querying the directory for the object specified by the administrator to the client application in order to retrieve the nTSecurityDescriptor attribute ([MS-ADTS] section 5.1.3 and [MS-ADA3] section 2.36).

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), and then sends an LDAP search response ([RFC2251] section 4.5.2) 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 and [MS-DTYP] section 2.4.6), and then sends an LsarOpenPolicy<8> request ([MS-LSAT] section 3.1.4.2) with the *DesiredAccess* parameter set to POLICY_LOOKUP_NAMES ([MS-LSAD] section 2.2.1.1.2) to the server to retrieve a context handle that it can later use to request name/SID translations.

6. The server processes the LsarOpenPolicy request by performing an access check for the desired access ([MS-LSAD] section 3.1.4.2) and constructing an LSAPR_HANDLE ([MS-LSAD] section 2.2.2.1), as described in [MS-LSAT] section 3.1.4.2. It sends the response to the client.

7. The client sends an LsarLookupSids<9> request ([MS-LSAT] section 3.1.4.11) using the server handle from step 6. 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 described in [MS-LSAT] section 3.1.4.11, and sends a response to the client that includes 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 or she wants to define access control entries for on the object. It then sends an LsarLookupNames<10> request ([MS-LSAT] section 3.1.4.8) including those names in order to retrieve their SIDs.

10. The server translates the names that are included in the request according to the processing defined in [MS-LSAT] section 3.1.4.8, and sends a response to the client that includes 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), and sends an LDAP modify request ([RFC2251] section 4.6) to commit that update.

12. The server verifies that the client has permissions to update the attribute, and then processes the modify request and performs validation as described in [MS-ADTS] sections 3.1.1.5.1 and 3.1.1.5.3. It then sends an LDAP modify response indicating success.

13. The client finally sends an LsarClose Request ([MS-LSAT] section 3.1.4.3 and [MS-LSAD] sections 3.1.4.3 and 3.1.4.9.4) 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) to the server. The LDAP connection to the directory server is closed.

3.2.19 Example 19: Raise the Domain Functional Level

In this example, an administrator modifies the ms-DS-Behavior-Version attribute ([MS-ADA2] section 2.207 and [MS-ADTS] section 3.1.1.5.3.1.1.5) 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).

This example applies only to AD DS.

This example covers the use case in section 2.7.1.3, Modify Directory Object - Client Application.

Prerequisites

The general requirements set forth in section 2.6, Assumptions and Preconditions.

The Active Directory System must meet all preconditions described in section 2.7.1.3.

Initial System State

None.

Final System State

The crossRef object is modified, and the domain functional level is raised.

Sequence of Events

The following sequence diagram depicts the message flow associated with this example.

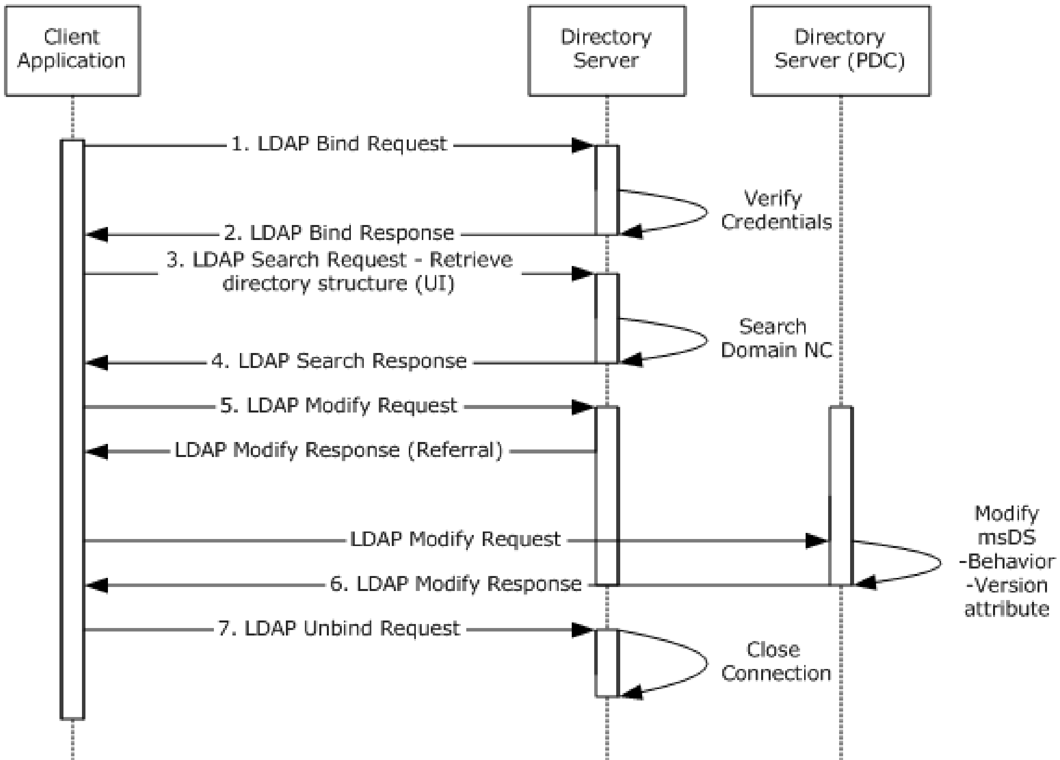

Figure 64: Message flow for raising the domain functional level

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 starts, and an LDAP bind request ([RFC2251] section 4.2) is sent to the directory server with credentials.

2. The directory server verifies the credentials ([MS-AUTHSOD] section 2) and sends an LDAP bind response ([RFC2251] section 4.2.3) to the client application.

3. An LDAP search request ([RFC2251] section 4.5.1) is sent to the directory, querying the base domain, looking for the msDS-Behavior-Version attribute ([MS-ADA2] section 2.207 and [MS-ADTS] section 3.1.1.5.3.1.1.5).

4. The server sends an LDAP search response ([RFC2251] section 4.5.2) containing the current domain functional level value (2, DS_BEHAVIOR_WIN2003, [MS-ADTS] section 6.1.4.2).

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).

   An LDAP modify request ([RFC2251] section 4.6) 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 2.207) as a replace operation with the value of 3.

If necessary, the client application will reroute the modify request ([RFC2251] section 4.6) to the PDC FSMO if a referral was returned from the initial modify request. This is because only the PDC FSMO role holder can perform this modification ([MS-ADTS] section 6.1.4.3).

6. The server processes the modify request and verifies the processing rules and constraints as described in [MS-ADTS] sections 3.1.1.5.1 and 3.1.1.5.3. It then sends an LDAP modify response ([RFC2251] section 4.6) indicating success.

7. The client application sends an LDAP unbind request ([RFC2251] section 4.3) to the server. The LDAP connection to the directory server is closed.

3.2.20 Example 20: Replicate Changes within a Domain

This example shows how the originating updates to a domain controller are replicated to other domain controllers within the same domain.

This example covers the use case in section 2.7.5.1, Replicate Changes within a Domain - Domain Controller.

Prerequisites

The general requirements set forth in section 2.6, Assumptions and Preconditions.

The Active Directory System must meet all preconditions described in section 2.7.5.1.

Initial State

An administrator adds a user to a domain controller, DC1, and these changes are not yet replicated to other domain controllers, DC2 and DC3.

Final State

Originating updates for DC1 are replicated to DC2 and DC3.

Sequence of Events

The following sequence diagram depicts the message flow associated with this example.

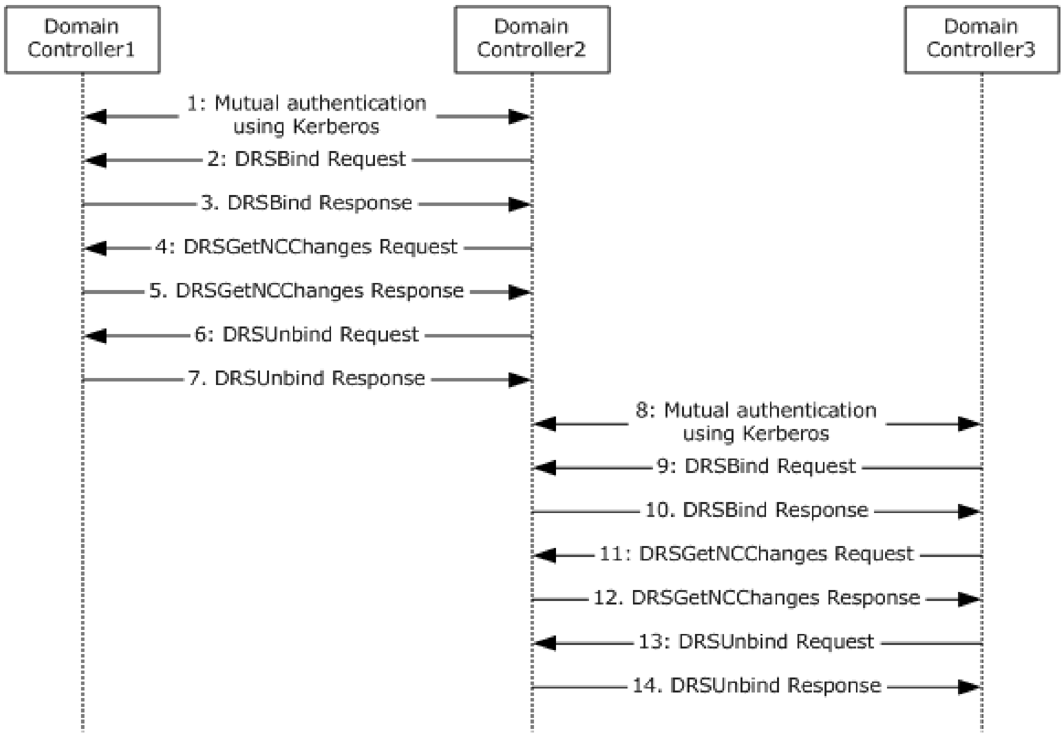

Figure 65: Message flow for replication changes within a domain

1. DC1 and DC2 perform mutual authentication using Kerberos ([MS-DRSR] section 2.2.3.2).
2. DC2 sends an IDL_DRSBind request to DC1, which creates a context handle that is necessary in order to call any other methods in the interface ([MS-DRSR] section 4.1.3).
3. Upon a successful response from DC1, DC2 obtains a context handle.
4. DC2 invokes IDL_DRSGetNCChanges on DC1 periodically to replicate the changes performed to DC1 ([MS-DRSR] section 4.1.10).
5. Upon a successful response, DC1 replicates its changes to DC2.
6. DC2 sends an IDL_DRSUnbind request, which destroys the context handle that was previously created by the IDL_DRSBind request ([MS-DRSR] section 4.1.25).
7. Upon a successful response from DC1, the context handle that was created previously is destroyed.
8. DC2 and DC3 perform mutual authentication using Kerberos ([MS-DRSR] section 2.2.3.2).
9. DC3 sends an IDL_DRSBind request to DC2, which creates a context handle that is necessary in order to call any other methods in the interface ([MS-DRSR] section 4.1.3).
10. Upon a successful response from DC2, DC3 obtains a context handle.
11. DC3 invokes IDL_DRSGetNCChanges on DC2 periodically to replicate the changes performed to DC2 ([MS-DRSR] section 4.1.10).
12. Upon a successful response, DC2 replicates its changes to DC3.

13. DC3 sends an IDL_DRSUnbind request, which destroys the context handle that was previously created by the IDL_DRSBind request ([MS-DRSR] section 4.1.25).

14. Upon a successful response from DC2, the context handle that was created previously is destroyed.

3.2.21 Example 21: Transferring FSMO roles

This example shows how the operation to transfer a FSMO role is accomplished by a DC that is being demoted.

This example applies only to AD LDS.

This example covers the use case in section 2.7.5.3, Transferring a FSMO Role - Domain Controller.

Prerequisites

The general requirements set forth in section 2.6, Assumptions and Preconditions.

The Active Directory System must meet all preconditions described in section 2.7.5.3.

DC1, DC2, and DC3 are domain controllers in AD LDS forest.

DC2 and DC3 are replication partners to DC1.

Initial State

DC1 is the owner of the Schema Master FSMO role and the Infrastructure FSMO role.

An administrator initiates demotion of DC1.

Final State

DC1 is demoted.

DC2 becomes the Infrastructure FSMO role owner.

DC3 becomes the Schema Master FSMO role owner.

Sequence of Events

The following sequence diagram depicts the message flow associated with this example.

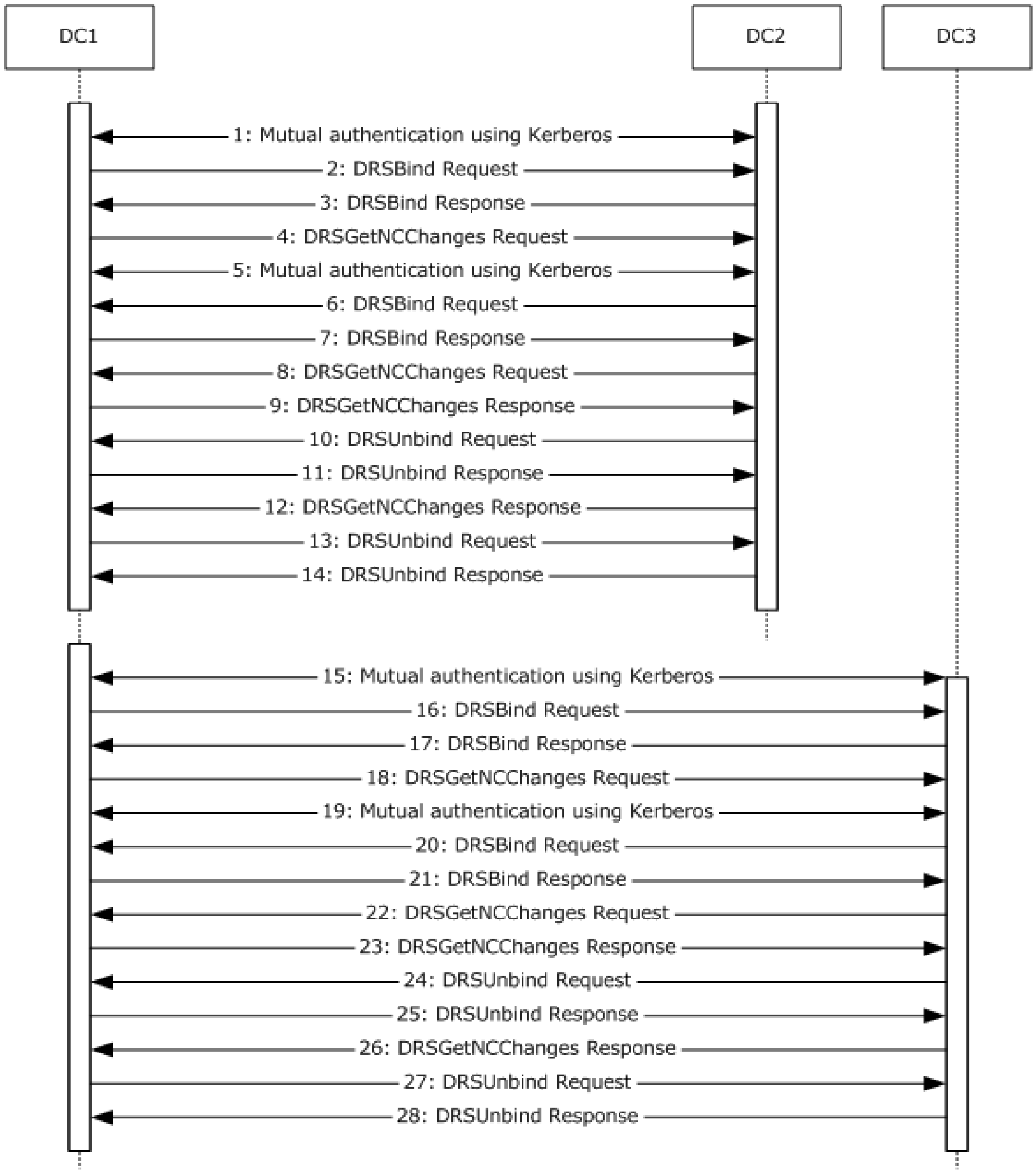

Figure 66: Message flow for transferring FSMO roles

1. DC1 performs mutual authentication with DC2 using Kerberos ([MS-DRSR] section 2.2.3.2).
2. DC1 sends an IDL_DRSBind request to DC2, which creates a context handle that is necessary in order to call any other methods in the interface ([MS-DRSR] section 4.1.3).
3. Upon a successful response from DC2, DC1 obtains a context handle.
4. DC1 invokes IDL_DRSGetNCChanges on DC2 to inform DC2 that it needs to abandon its Infrastructure FSMO role and transfer it to DC2. DC1 chooses DC2 randomly out of all its replication partners.
5. DC2 performs mutual authentication with DC1 using Kerberos ([MS-DRSR] section 2.2.3.2).

6. DC2 sends an IDL_DRSBind request to DC1, which creates a context handle that is necessary in order to call any other methods in the interface ([MS-DRSR] section 4.1.3).

7. Upon a successful response from DC1, DC2 obtains a context handle to DC1.

8. DC2 invokes IDL_DRSGetNCChanges on DC1 to request a transfer of the Infrastructure FSMO role to itself.

9. Upon Successful response, DC1 considers DC2 to be the owner of the Infrastructure FSMO role and returns all the changed objects to DC2.

10. DC2 sends an IDL_DRSUnbind request to DC1, which destroys the context handle that was previously created by the IDL_DRSBind request ([MS-DRSR] section 4.1.25).

11. Upon a successful response from DC1, the context handle that was created previously is destroyed.

12. Upon successful completion of the preceding steps, DC2 returns a successful response for the IDL_DRSGetNCChanges request to DC1.

13. DC1 sends an IDL_DRSUnbind request to DC2, which destroys the context handle that was previously created by the IDL_DRSBind request ([MS-DRSR] section 4.1.25).

14. Upon a successful response from DC2, the context handle that was created previously is destroyed.

15. DC1 performs mutual authentication with DC3 using Kerberos ([MS-DRSR] section 2.2.3.2).

16. DC1 sends an IDL_DRSBind request to DC3, which creates a context handle that is necessary in order to call any other methods in the interface ([MS-DRSR] section 4.1.3).

17. Upon a successful response from DC3, DC1 obtains a context handle.

18. DC1 invokes IDL_DRSGetNCChanges on DC3 to inform DC3 that it needs to abandon its Schema Master FSMO role and transfer it to DC3. DC1 chooses DC3 randomly out of all its replication partners.

19. DC3 performs mutual authentication with DC1 using Kerberos ([MS-DRSR] section 2.2.3.2).

20. DC3 sends an IDL_DRSBind request to DC1, which creates a context handle that is necessary in order to call any other methods in the interface ([MS-DRSR] section 4.1.3).

21. Upon a successful response from DC1, DC3 obtains a context handle to DC1.

22. DC3 invokes IDL_DRSGetNCChanges on DC1 to request a transfer of the Schema Master FSMO role to itself.

23. Upon successful response, DC1 considers DC3 to be the owner of the Schema Master FSMO role and returns all the changed objects to DC3.

24. DC3 sends an IDL_DRSUnbind request to DC1, which destroys the context handle that was previously created by the IDL_DRSBind request ([MS-DRSR] section 4.1.25).

25. Upon a successful response from DC1, the context handle that was created previously is destroyed.

26. Upon successful completion of the preceding steps, DC3 returns a successful response for the IDL_DRSGetNCChanges request to DC1.

27. DC1 sends an IDL_DRSUnbind request to DC3, which destroys the context handle that was previously created by the IDL_DRSBind request ([MS-DRSR] section 4.1.25).

28. Upon a successful response from DC3, the context handle that was created previously is destroyed.

3.2.22 Example 22: Replicate Changes to a GC or a Partial Replica by Using SMTP

This example describes how replication is performed between a full replica domain controller, DC1, and a partial replica domain controller, DC2, by using the SMTP transport.

This example covers the use case in section 2.7.5.2, Replicate Changes to a GC or a Partial Replica Using RPC - Domain Controller, specifically, the SMTP variation.

Prerequisites

The general requirements set forth in section 2.6, Assumptions and Preconditions.

The Active Directory System must meet all preconditions described in the variation of section 2.7.5.2.

Initial State

DC1 has originating updates that have not yet been replicated to DC2.

Final State

The originating updates on DC1 are replicated to DC2.

Sequence of events

The following sequence diagram depicts the message flow associated with this example.

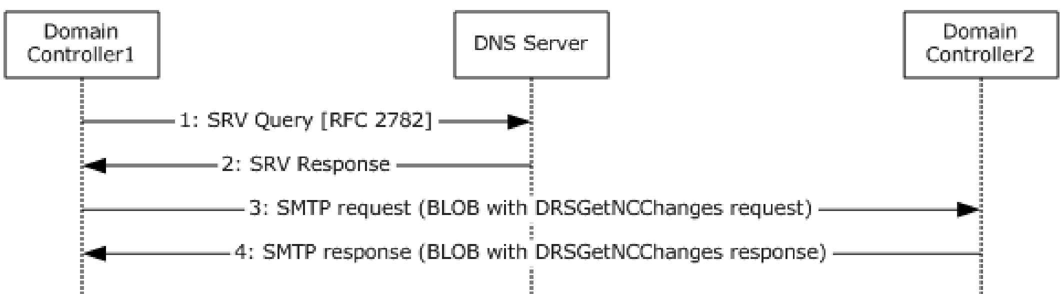

Figure 67: Message flow for replication changes to a GC or a partial replica

1. DC1 sends an SRV query request to the DNS server in order to obtain the IP address of DC2.
2. The DNS server responds back to DC1 with the IP address of DC2 ([MS-ADTS] section 6.3.6).
3. DC1 sends the SMTP message as follows.

   The DRS engine hands the SRPL extension a **binary large object (BLOB)** that contains a "get changes" request. The SRPL extension performs the higher-layer triggered operation ([MS-SRPL] section 3.2.4), encoding the request BLOB as a frame. The frame is then handed to the SMTP service on DC1. The frame, as an attachment to an SMTP mail message, is sent to DC2. DC2 receives the request. The SMTP service on DC2 receives the mail message from DC1 and gives the frame to the SRPL extension. The SRPL extension performs message processing operations

([MS-SRPL] section 3.3.5) and then passes the BLOB to the DRS engine, which processes the request.

4. DC2 sends the SMTP response as follows.

   After it processes the request, the DRS engine on DC2 generates another DRS BLOB, which contains the response. The SRPL extension performs the higher-layer triggered operation and then passes the response to the SMTP service. The SMTP service sends the message to DC1 as an attachment to an SMTP mail message. DC1 receives the response. The SMTP service on DC1 receives the mail message and gives the frame to the SRPL extension. The SRPL extension performs message processing and then passes the BLOB to the DRS engine, which processes the response.

3.2.23 Example 23: Cross-Domain Move

This example demonstrates the use case in section 2.7.1.6, Cross-Domain Move - Client Application.

It is explained in [MS-DRSR] section 4.1.15.4.

4 Microsoft Implementations

There are no variations in the behavior of the Active Directory system in different versions of Windows beyond those that are described in the specifications of the protocols supported by the system, as listed in section 2.4.

The information in this specification overview document is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs:

- Windows NT 3.1 operating system
- Windows NT 3.5 operating system
- Windows NT 3.51 operating system
- Windows NT 4.0 operating system
- Windows 2000 operating system
- Windows XP operating system
- Windows Server 2003 operating system
- Windows Server 2003 R2 operating system
- Active Directory Application Mode (ADAM)
- Windows Vista operating system
- Windows Server 2008 operating system
- Active Directory Lightweight Directory Services (AD LDS) for Windows Vista
- Windows 7 operating system
- Windows Server 2008 R2 operating system
- Active Directory Lightweight Directory Services (AD LDS) for Windows 7
- Windows 8 operating system
- Windows Server 2012 operating system
- Active Directory Lightweight Directory Services (AD LDS) for Windows 8 operating system
- Windows 8.1 operating system
- Windows Server 2012 R2 operating system
- Active Directory Lightweight Directory Services (AD LDS) for Windows 8.1 operating system

Exceptions, if any, are noted below.

4.1 Product Behavior

<1> Section 1: AD LDS 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 1: AD DS uses the operations described in [MS-NRPC] section 3.6 only to maintain the change log that Windows NT 4.0 Backup Domain Control (BDC) replication uses.

<3> Section 2.4: This feature is implemented on Windows Server 2012 and Windows Server 2012 R2.

<4> Section 2.5.1: [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 that are 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 expect Method 1 when parsing dynamic update responses and might 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 accept either method of formatting dynamic update responses.

<5> Section 2.8: 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.

<6> Section 2.11.2: The Microsoft implementation of Active Directory Web Services permits administrators to configure settings that are 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 is 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 automatically releases it.
- Length of time an enumeration context can be kept open (whether idle or in use) by a client before the directory service automatically releases it.
- Maximum expiration time the directory service permits a client to specify in an Enumerate or Renew request (via the Expires element).
- Maximum amount of time the directory service permits a client to specify in a Pull request (via the MaxTime element).
- Maximum number of elements the directory service permits a client to specify in a Pull request (via the MaxElements element).

<7> Section 3.1.1: A workstation that is logging on to a Windows 2000 domain queries DNS for SRV records in the general form:

```
_service._protocol.DnsDomainName
```

<8> Section 3.2.18: Windows uses the highest version of the operation first; that is, LsarOpenPolicy2 is preferred over LsarOpenPolicy.

<9> Section 3.2.18: Windows uses the highest version of the operation first; that is, LsarLookupSids2 is preferred over LsarLookupSids.

<10> Section 3.2.18: Windows uses the highest version of the operation first; that is, LsarLookupNames3 is preferred over LsarLookupNames2 and LsarLookupNames.

5 Change Tracking

This section identifies changes that were made to the [MS-ADOD] protocol document between the January 2013 and August 2013 releases. Changes are classified as New, Major, Minor, Editorial, or No change.

The revision class **New** means that a new document is being released.

The revision class **Major** means that the technical content in the document was significantly revised. Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements or functionality.
- An extensive rewrite, addition, or deletion of major portions of content.
- The removal of a document from the documentation set.
- Changes made for template compliance.

The revision class **Minor** means that the meaning of the technical content was clarified. Minor changes do not affect protocol interoperability or implementation. Examples of minor changes are updates to clarify ambiguity at the sentence, paragraph, or table level.

The revision class **Editorial** means that the language and formatting in the technical content was changed. Editorial changes apply to grammatical, formatting, and style issues.

The revision class **No change** means that no new technical or language changes were introduced. The technical content of the document is identical to the last released version, but minor editorial and formatting changes, as well as updates to the header and footer information, and to the revision summary, may have been made.

Major and minor changes can be described further using the following change types:

- New content added.
- Content updated.
- Content removed.
- New product behavior note added.
- Product behavior note updated.
- Product behavior note removed.
- New protocol syntax added.
- Protocol syntax updated.
- Protocol syntax removed.
- New content added due to protocol revision.
- Content updated due to protocol revision.
- Content removed due to protocol revision.
- New protocol syntax added due to protocol revision.

- Protocol syntax updated due to protocol revision.
- Protocol syntax removed due to protocol revision.
- New content added for template compliance.
- Content updated for template compliance.
- Content removed for template compliance.
- Obsolete document removed.

Editorial changes are always classified with the change type **Editorially updated.**

Some important terms used in the change type descriptions are defined as follows:

- **Protocol syntax** refers to data elements (such as packets, structures, enumerations, and methods) as well as interfaces.
- **Protocol revision** refers to changes made to a protocol that affect the bits that are sent over the wire.

The changes made to this document are listed in the following table. For more information, please contact protocol@microsoft.com.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
1.1 Glossary	Updated content for Windows Server 2012 R2.	Y	Content updated.
4 Microsoft Implementations	Modified this section to include references to Windows Server 2012 operating system, Active Directory Lightweight Directory Services (AD LDS) for Windows 8 operating system, Windows 8.1 operating system, Windows Server 2012 R2 operating system, and Active Directory Lightweight Directory Services (AD LDS) for Windows 8.1 operating system.	Y	Content updated.

6 Index

A

Additional considerations 114
Applicable protocols 25
Architecture 20
Assumptions 30

C

Capability negotiation 103
Change tracking 181
Coherency requirements 108
Component dependencies 29
Concepts 20

D

Dependencies
  with other systems 29
  within the system 28

E

Environment 27
Error handling 104
Examples
  directory 127
  domain-join 115
Examples - overview 115
Extensibility - Microsoft implementations 178
Extensibility - overview 103
External dependencies 28

F

Functional requirements – overview 20

G

Glossary 12

I

Implementer - security considerations 108
Informative references 15
Initial state 30

M

Microsoft implementations 178

O

Overview
  summary of protocols 25
Overview - synopsis 20

P

Product behavior 178

R

References 15
Requirements
  coherency 108
  error handling 104
  preconditions 30

S

Security considerations 108
System dependencies 27
  with other systems 29
  within the system 28
System errors 104
System protocols 25
System use cases - overview 31

T

Table of protocols 25
Tracking changes 181

U

Use cases 31

V

Versioning
  overview 103
Versioning - Microsoft implementations 178