[MS-SAMR]:
Security Account Manager (SAM) Remote Protocol Specification (Client-to-Server)

Revision Summary

Date	Revision History	Revision Class	Comments
02/22/2007	0.01		MCPP Milestone 3 Initial Availability
06/01/2007	1.0	Major	Updated and revised the technical content.
07/03/2007	2.0	Major	Added example.
07/20/2007	3.0	Major	Rewrite of keying algorithms; clarification of user

Date	Revision History	Revision Class	Comments
			account enabling.
08/10/2007	3.0.1	Editorial	Revised and edited the technical content.
09/28/2007	3.0.2	Editorial	Revised and edited the technical content.
10/23/2007	3.1	Minor	Updated the technical content.
11/30/2007	3.2	Minor	Updated the technical content.
01/25/2008	4.0	Major	Updated and revised the technical content.

Table of Contents

1 Introduction

This document is the specification for the Security Account Manager (SAM) Remote Protocol (Client-to-Server), a Microsoft proprietary protocol. It assumes that readers have read the Windows Protocols Overview [MS-PROTO], the Windows Security Overview [MS-SECO], and the Active Directory Technical Specification [MS-ADTS].

In Windows implementations, this protocol exposes the "account database" referred to in [MS-SECO], both for "Local" and "Remote" **domains**. This document specifies the behavior for Local and Remote by having a common data model for both scenarios, the **Active Directory** data model, as specified in [MS-ADTS]. In addition, this document specifies the difference in the behavior between these scenarios when necessary.

1.1 Glossary

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

Access Check
Access Mask
Account Domain Object (Account Domain)
Account Domain SID
Account Group
Access Control Entry (ACE)
Access Control List (ACL)
Active Directory (AD)
Administrators
Alias Object
Authorization Context
Built-in Domain
Control Access Right
Database Object
Domain
Domain Admins
Domain Prefix
Forest
Mixed Mode
Native Mode
OEM Code Page
Relative Identifier (RID)
Resource Group
System Access Control List (SACL)
Security Principal
Security Identifier (SID)
User Profile

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

dsname
Primary Domain Controller (PDC)
Relative Distinguished Name (RDN)

The following terms are specific to this document:

Account: A user (including **machine account**), group, or alias object.

Delta-Time: A negative **FILETIME**. **Delta-Time** specifies a period of time, expressed in a negative number of 100 nanosecond time slices. For example, a period of 20 minutes is represented as -12000000000.

Domain Controller (DC): A server on which the **Active Directory** operating system directory service is installed and operating. It hosts the data store for objects and interoperates with other **DCs** to ensure that an originating change to an object replicates correctly across all **DCs**. For more information, see [MS-SECO] section 2.5.2.

Domain Object: A **database object** that represents an issuing authority as specified in [MS-SECO] section 2.2. An **account** is said to be "in" a particular **domain** if the **domain prefix** of its **SID** is the **SID** of the particular **domain**.

Group Object: A **database object** that represents a collection of **user objects** and **group objects** and has a **SID** value.

LM Hash: A DES-based cryptographic hash of a cleartext password. See LMOWFv1, as specified in [MS-NLMP] section 3.3.1 (NTLM v1 Authentication), for a normative definition.

Machine Account: A **user object** that represents a computer (as opposed to an end user).

NT Hash: An MD4-based cryptographic hash of a cleartext password. See NTOWFv1, as specified in [MS-NLMP] section 3.3.1 (NTLM v1 Authentication), for a normative definition.

RC4: A stream cipher licensed by RSA Data Security [RSADSI] under RSA BSAFE.

Salt: A value consisting of random bits used to increase the complexity of dictionary attacks against secret data that is protected through cryptographic means. For details, see [MENEZES] section 10.2.1.

Security Descriptor: A policy expressing access control. For more information, see [MS-GLOS] section 1.1.20 and [MS-DTYP] sections **2.4.6** and 2.5.

UAS Compatibility: A configuration mode that affects protocol behavior constraints specified in this document. "UAS" is the acronym for "User Account Security (Database)" and refers to products no longer supported, such as Microsoft NT LAN Manager. The default setting in Windows is "off".

Universal Group: Similar to an **account group**, with the distinction that a **universal group** may have members from any **domain** in the responder's **forest**.

User Object: A **database object** that represents a **security principal**, as specified in [MS-SECO] section 2.1.

WorldSid: A **SID** with the specific value of S-1-1-0. **WorldSid** is used in a default **ACL**.

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

1.2 References

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site,

http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

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

[E164] ITU-T, "The International Public Telecommunication Numbering Plan", Recommendation E.164, February 2005, http://www.itu.int/rec/T-REC-E.164/e

Note There is a charge to download the specification.

[FIPS46-2] National Institute of Standards and Technology, "Federal Information Processing Standards Publication 46-2: Data Encryption Standard (DES)", December 1993, http://www.itl.nist.gov/fipspubs/fip46-2.htm

[FIPS81] National Institute of Standards and Technology, "Federal Information Processing Standards Publication 81: DES Modes of Operation", December 1980, http://csrc.nist.gov/publications/fips/fips81/fips81.htm

[MENEZES] Menezes, A., Vanstone, S., and Van Oorschot, P., "The Handbook of Applied Cryptography", CRC Press, 1997, ISBN: 0849385237.

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

[MS-ADA2] Microsoft Corporation, "Active Directory Schema Attributes M", June 2007.

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

[MS-ADSC] Microsoft Corporation, "Active Directory Schema Classes", June 2007.

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

[MS-DRSR] Microsoft Corporation, "Directory Replication Service (DRS) Remote Protocol Specification", July 2007.

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

[MS-ERREF] Microsoft Corporation, "Windows Error Codes", January 2007.

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

[MS-KILE] Microsoft Corporation, "Kerberos Protocol Extensions", January 2007.

[MS-LSAD] Microsoft Corporation, "Local Security Authority (Domain Policy) Remote Protocol Specification", June 2007.

[MS-LSAT] Microsoft Corporation, "Local Security Authority (Translation Methods) Remote Protocol Specification", June 2007.

[MS-NLMP] Microsoft Corporation, "NT LAN Manager (NTLM) Authentication Protocol Specification", June 2007.

[MS-NRPC] Microsoft Corporation, "Netlogon Remote Protocol Specification", March 2007.

[MS-PAC] Microsoft Corporation, "Privilege Attribute Certificate Data Structure", January 2007.

[MS-PROTO] Microsoft Corporation, "Windows Protocols Overview", June 2007.

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

[MS-SECO] Microsoft Corporation, "Windows Security Overview", January 2007.

[MS-SFU] Microsoft Corporation, "Kerberos Protocol Extensions: Service for User and Constrained Delegation Protocol Specification", January 2007.

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

[MSFT-LATIN1] Microsoft Corporation, "Windows 28591", May 2005, http://www.microsoft.com/globaldev/reference/iso/28591.mspx

[RFC1123] Braden, R., "Requirements for Internet Hosts–Application and Support", RFC 1123, October 1989, http://www.ietf.org/rfc/rfc1123.txt

[RFC1321] Rivest, R. "The MD5 Message-Digest Algorithm", RFC 1321, April 1992, http://www.ietf.org/rfc/rfc1321.txt

[RFC1510] Kohl, J., Neuman, C., "The Kerberos Network Authentication Service (V5)", RFC 1510, September 1993, http://www.ietf.org/rfc/rfc1510.txt

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

[RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and Stewart, L., "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999, http://www.ietf.org/rfc/rfc2617.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

[RSADSI] Contact RSA Data Security, Inc., Tel: 866-432-7233

[UNICODE3.1] The Unicode Consortium, "Unicode Data 3.1.0", February 2001, http://www.unicode.org/Public/3.1-Update/UnicodeData-3.1.0.txt

1.2.2 Informative References

[LAMPORT] Lamport, L., "Time, Clocks, and the Ordering of Events in a Distributed System", July 1978, http://research.microsoft.com/users/lamport/pubs/time-clocks.pdf

[MSDN-CP] Microsoft Corporation, "Code Page Identifiers", http://msdn2.microsoft.com/en-us/library/ms776446.aspx

If you have any trouble finding [MSDN-CP], please check here.

[MSDN-NMF] Microsoft Corporation, "Network Management Functions", http://msdn2.microsoft.com/en-us/library/aa370675.aspx

If you have any trouble finding [MSDN-NMF], please check here.

1.3 Protocol Overview (Synopsis)

The goal of this protocol is to enable IT administrators and end users to manage user, group, and computer entities. IT administrators and their delegates generally have full access control to these

entities, and consequently can manage the entities' life cycles. End users are allowed to make changes to their own data (in most cases, limited to just their passwords).

This protocol achieves its goal by enabling the creation, reading, updating, and deleting of security principal information. These security principals could be in any account store. Windows implements this protocol, for example, in a directory service (Active Directory) and in a computer-local security account database. In this specification, normative differences in the protocol between these two cases are indicated by referring to the configuration of the responder as a "**DC**" or "non-DC" configuration, respectively.

It is helpful to consider the following two perspectives when understanding and implementing this protocol:

- Object-based perspective (see section 1.3.1)
- Method-based perspective (see section 1.3.2)

1.3.1 Object-Based Perspective

The object-based perspective shows that the protocol exposes five main object abstractions: a server object, a **domain object**, a **group object**, an **alias object** (an "alias" being a type of group), and a **user object**. A requester obtains a "handle" (an RPC context handle), to one of these objects and then performs one or more actions on the object.

A brief listing of methods that operate on each of the respective object types is shown below.

Server Object:

- **SamrSetSecurityObject**
- **SamrQuerySecurityObject**
- **SamrEnumerateDomainsInSamServer**
- **SamrOpenDomain**
- **SamrLookupDomainInSamServer**
- **SamrCloseHandle**

Domain Object:

- **SamrSetSecurityObject**
- **SamrQuerySecurityObject**
- **SamrLookupNamesInDomain**
- **SamrLookupIdsInDomain**
- **SamrEnumerateGroupsInDomain**
- **SamrEnumerateUsersInDomain**
- **SamrEnumerateAliasesInDomain**
- **SamrOpenGroup**

- SamrOpenAlias
- SamrOpenUser
- SamrQueryInformationDomain
- SamrQueryInformationDomain2
- SamrCreateGroupInDomain
- SamrCreateAliasInDomain
- SamrCreateUserInDomain
- SamrCreateUser2InDomain
- SamrSetInformationDomain
- SamrGetAliasMembership
- SamrGetDisplayEnumerationIndex
- SamrGetDisplayEnumerationIndex2
- SamrQueryDisplayInformation
- SamrQueryDisplayInformation2
- SamrQueryDisplayInformation3
- **SamrCloseHandle**
- SamrRemoveMemberFromForeignDomain

Group Object:

- **SamrSetSecurityObject**
- **SamrQuerySecurityObject**
- SamrQueryInformationGroup
- SamrSetInformationGroup
- SamrDeleteGroup
- SamrAddMemberToGroup
- SamrRemoveMemberFromGroup
- SamrGetMembersInGroup
- **SamrCloseHandle**
- SamrSetMemberAttributesOfGroup
- SamrRidToSid

Alias Object:

- **SamrSetSecurityObject**
- **SamrQuerySecurityObject**
- **SamrQueryInformationAlias**
- **SamrSetInformationAlias**
- **SamrDeleteAlias**
- **SamrAddMemberToAlias**
- **SamrRemoveMemberFromAlias**
- **SamrGetMembersInAlias**
- **SamrAddMultipleMembersToAlias**
- **SamrRemoveMultipleMembersFromAlias**
- **SamrRidToSid**

User Object:

- **SamrSetSecurityObject**
- **SamrQuerySecurityObject**
- **SamrQueryInformationUser**
- **SamrQueryInformationUser2**
- **SamrSetInformationUser**
- **SamrSetInformationUser2**
- **SamrDeleteUser**
- **SamrGetGroupsForUser**
- **SamrChangePasswordUser**
- **SamrGetUserDomainPasswordInformation**
- **SamrCloseHandle**
- **SamrRidToSid**

For example, to set a policy that limits the minimum length of passwords to eight characters for all users, a requester opens a handle to a domain object and updates the minimum length password policy setting via a parameter field called *MinPasswordLength*. The call sequence from the requester appears as follows (with the parameter information removed for brevity):

(a) Send a **SamrConnect5** request; receive the **SamrConnect5** reply.

(b) Send a **SamrOpenDomain** request; receive the **SamrOpenDomain** reply.

(c) Send a **SamrSetInformationDomain** request; receive the **SamrSetInformationDomain** reply.

(d) Send a **SamrCloseHandle** request; receive the **SamrCloseHandle** reply.

(e) Send a **SamrCloseHandle** request; receive the **SamrCloseHandle** reply.

This sequence is expanded in the following brief explanation:

Step (a): Using the network address of a responder that implements this protocol, a requester makes a **SamrConnect5** request to obtain a handle to a server object. This server handle is necessary to obtain a subsequent handle to a domain object.

Step (b): Using the handle returned from **SamrConnect5**, the requester makes a **SamrOpenDomain** request to obtain a handle to a domain object.

Step (c): Using the handle returned from **SamrOpenDomain**, the requester makes a **SamrSetInformationDomain** request, setting the *MinPasswordLength* parameter field to eight.

Steps (d) and (e): The requester closes the handles returned from **SamrOpenDomain** and **SamrConnect5** using **SamrCloseHandle**. These steps release responder resources associated with the handle; the order in which the handles are released is not important.

Section 4.1 provides an additional example.

1.3.2 Method-Based Perspective

The method-based perspective is used to show a common set of operations for each object type. The operations fall into patterns. A list of the patterns and associated methods, along with a description of each pattern, is shown below.

- Open Pattern

  This pattern returns an RPC context handle that references a specific object type. A requester uses this pattern by specifying a specific access for the handle in the request, and using the returned handle to call other methods that require the returned handle along with the associated access. For example, calling the method **SamrSetInformationDomain** requires a domain handle that has been opened with DOMAIN_WRITE_PASSWORD_PARAMS. For more information on the range of accesses for a domain object, see section 2.2.1.4.

  SamrConnect2, **SamrConnect4**, and **SamrConnect5** are distinguished from the other methods in this pattern by the following two characteristics:

  - They are the first methods that a requester must call before a call to any other handle-based methods.
  - A network address, rather than a context handle, is required to identify the responder.

  The methods that follow the open pattern are as follows:

  - **SamrConnect5**
  - **SamrConnect4**
  - **SamrConnect2**
  - **SamrOpenDomain**
  - **SamrOpenGroup**
  - **SamrOpenAlias**

- **SamrOpenUser**

- Enumerate Pattern

  This pattern allows a requester to obtain a complete list of all objects of a certain type (domain, group, alias, or user).

  The methods that follow the enumerate pattern are as follows:

  - **SamrEnumerateDomainsInSamServer**
  - **SamrEnumerateGroupsInDomain**
  - **SamrEnumerateAliasesInDomain**
  - **SamrEnumerateUsersInDomain**

- Selective Enumerate Pattern

  This pattern allows a requester to obtain a partial list of objects based on the name of the objects. These methods, for example, allow a requester to obtain a bounded number of objects from a virtual list of objects sorted alphabetically by name starting with a requester-specified prefix, such as "Chr". User interface programs use these methods to allow the end user to quickly find an object, given partial knowledge of the object's name.

  The methods that follow the selective enumerate pattern are as follows:

  - **SamrQueryDisplayInformation3**
  - **SamrQueryDisplayInformation2**
  - **SamrQueryDisplayInformation**
  - **SamrGetDisplayEnumerationIndex2**
  - **SamrGetDisplayEnumerationIndex**

- Create Pattern

  This pattern allows specified objects to be created. A handle to the newly created object is returned.

  The methods that follow the create pattern are as follows:

  - **SamrCreateGroupInDomain**
  - **SamrCreateAliasInDomain**
  - **SamrCreateUser2InDomain**
  - **SamrCreateUserInDomain**

- Query Pattern

  This pattern allows specified attributes of an object to be returned. The requester specifies which attributes to return by using an "information level". The information level is an enumeration that the responder understands and translates into a specific structure to return; the structure contains the attributes indicated by the information level.

To retrieve the name of a user, for example, a requester specifies the "UserAccountNameInformation" information level in the **SamrQueryInformationUser** method.

The methods that follow the query pattern are as follows:

- **SamrQueryInformationDomain2**
- **SamrQueryInformationDomain**
- **SamrQueryInformationGroup**
- **SamrQueryInformationAlias**
- **SamrQueryInformationUser2**
- **SamrQueryInformationUser**

- Set Pattern

  This pattern allows specified object attributes to be set. The requester indicates the attributes that are to be updated by specifying an "information level". Similar to the query pattern of methods, the information level specifies the attributes that are being sent in the request.

  The methods that follow the set pattern are as follows:

  - **SamrSetInformationDomain**
  - **SamrSetInformationGroup**
  - **SamrSetInformationAlias**
  - **SamrSetInformationUser2**
  - **SamrSetInformationUser**

- Delete Pattern

  This pattern allows a requester to delete a specified object.

  The methods that follow the delete pattern are as follows:

  - **SamrDeleteGroup**
  - **SamrDeleteAlias**
  - **SamrDeleteUser**

- Membership Pattern

  This pattern allows a requester to add to, remove from, or query the membership list for either a group or an alias object.

  The methods that follow the membership pattern are as follows:

  - **SamrAddMemberToGroup**
  - **SamrRemoveMemberFromGroup**
  - **SamrAddMemberToAlias**

- **SamrRemoveMemberFromAlias**
- **SamrRemoveMemberFromForeignDomain**
- **SamrGetMembersInGroup**
- **SamrGetMembersInAlias**
- **SamrAddMultipleMembersToAlias**
- **SamrRemoveMultipleMembersFromAlias**

- Membership-Of Pattern

  This pattern allows a requester to obtain the groups or aliases that a user or collection of security identifiers (SIDs) is a member of.

  The methods that follow the membership-of pattern are as follows:

  - **SamrGetGroupsForUser**
  - **SamrGetAliasMembership**

- Change Password Pattern

  This pattern allows a requester to change a password on a user object. The requester provides the current password and new password, and the responder verifies that the requester-presented current password matches the responder-persisted current password for the user. If there is a match, the new password is persisted.

  The methods that follow the change password pattern are as follows:

  - **SamrChangePasswordUser**
  - **SamrOemChangePasswordUser2**
  - **SamrUnicodeChangePasswordUser2**

- Lookup Pattern

  This pattern allows a requester to translate between a relative identifier (RID) or SID, and a user-friendly display name (the name of the object).

  The methods that follow the lookup pattern are as follows:

  - **SamrLookupDomainInSamServer**
  - **SamrLookupNamesInDomain**
  - **SamrLookupIdsInDomain**

- Security Pattern

  This pattern allows a requester to specify or query access control with a granularity of individual objects.

  The methods that follow the security pattern are as follows:

  - **SamrSetSecurityObject**

- **SamrQuerySecurityObject**

- Miscellaneous

  The following methods do not fall into a general pattern; see the message processing sections for details about each one. A brief description of each method follows:

  - **SamrGetUserDomainPasswordInformation**: This method obtains information about the password policy on the account domain given a user handle. Applications that allow an end user to change their password can use this method to display policy information to the end user.

  - **SamrGetDomainPasswordInformation**: This method is similar to the **SamrGetUserDomainPasswordInformation** method, except that the responder does not enforce any security, and a user handle is not needed.

  - **SamrRidToSid**: This method returns a SID given a RID returned by any of the methods in this interface.<1>

  - **SamrSetDSRMPassword**: This method allows a requester to set the password on a local account (an account not stored in Active Directory) on a DC. This is useful for recovery scenarios where Active Directory does not start.

  - **SamrValidatePassword**: This method allows applications that store passwords to validate the strength of the passwords against the account domain policy.

  - **SamrSetMemberAttributesOfGroup**: This method allows a responder to configure extra authorization information associated with a group membership. This method is ignored in DC scenarios.

  - **SamrCloseHandle**: This method releases responder resources associated with the RPC context handle that is passed as a parameter.

1.4 Relationship to Other Protocols

This protocol depends on the RPC protocol because it uses RPC as a transport.

In the DC configuration, the data manipulated by the responder of this protocol is stored in Active Directory and is therefore replicated by the replication protocol (as specified in [MS-DRSR]), made available through the LDAP interface (see [MS-ADTS] section 3.1.1.3), and replicated by the NETLOGON replication interface (as specified in [MS-NRPC]).<2>

1.5 Prerequisites/Preconditions

An **OEM code page** MUST be configured in the responder implementation. This requirement enables the responder to accept data that is encoded in an OEM code page, as well as to return select results that are encoded in an OEM code page.

The requester implementation must know the network address of the responder. The network address must satisfy the requirements of a network address for the underlying transport of RPC. When using RPC over SMB, for example, the network address must be a network address compatible with the Server Message Block (SMB) Protocol [MS-SMB], such as a NETBIOS name.

1.6 Applicability Statement

This protocol is useful for manipulating an account database consisting of users, groups, and other **security principals**. This protocol can be used equally well for a database that is backed by a distributed, replicated system, as well as a small, single-instance scenario, such as a single machine.<3><4>

1.7 Versioning and Capability Negotiation

1.7.1 Method Introduction

The following table depicts a timeline of when each method was introduced. The Product column indicates the Windows version in which each method was introduced. Unless otherwise noted, when a method is supported in a particular version, it is also supported in all subsequent versions.

Opnum	Friendly name	Product
0	SamrConnect	Windows NT 3.1
1	SamrCloseHandle	Windows NT 3.1
2	SamrSetSecurityObject	Windows NT 3.1
3	SamrQuerySecurityObject	Windows NT 3.1
4	SamrShutdownSamServer Reserved (not intended for network traffic)	Windows NT 3.1
5	SamrLookupDomainInSamServer	Windows NT 3.1
6	SamrEnumerateDomainsInSamServer	Windows NT 3.1
7	SamrOpenDomain	Windows NT 3.1
8	SamrQueryInformationDomain	Windows NT 3.1
9	SamrSetInformationDomain	Windows NT 3.1
10	SamrCreateGroupInDomain	Windows NT 3.1
11	SamrEnumerateGroupsInDomain	Windows NT 3.1
12	SamrCreateUserInDomain	Windows NT 3.1
13	SamrEnumerateUsersInDomain	Windows NT 3.1
14	SamrCreateAliasInDomain	Windows NT 3.1
15	SamrEnumerateAliasesInDomain	Windows NT 3.1
16	SamrGetAliasMembership	Windows NT 3.1
17	SamrLookupNamesInDomain	Windows NT 3.1
18	SamrLookupIdsInDomain	Windows NT 3.1
19	SamrOpenGroup	Windows NT 3.1

Opnum	Friendly name	Product
20	SamrQueryInformationGroup	Windows NT 3.1
21	SamrSetInformationGroup	Windows NT 3.1
22	SamrAddMemberToGroup	Windows NT 3.1
23	SamrDeleteGroup	Windows NT 3.1
24	SamrRemoveMemberFromGroup	Windows NT 3.1
25	SamrGetMembersInGroup	Windows NT 3.1
26	SamrSetMemberAttributesOfGroup	Windows NT 3.1
27	SamrOpenAlias	Windows NT 3.1
28	SamrQueryInformationAlias	Windows NT 3.1
29	SamrSetInformationAlias	Windows NT 3.1
30	SamrDeleteAlias	Windows NT 3.1
31	SamrAddMemberToAlias	Windows NT 3.1
32	SamrRemoveMemberFromAlias	Windows NT 3.1
33	SamrGetMembersInAlias	Windows NT 3.1
34	SamrOpenUser	Windows NT 3.1
35	SamrDeleteUser	Windows NT 3.1
36	SamrQueryInformationUser	Windows NT 3.1
37	SamrSetInformationUser	Windows NT 3.1
38	SamrChangePasswordUser	Windows NT 3.1
39	SamrGetGroupsForUser	Windows NT 3.1
40	SamrQueryDisplayInformation	Windows NT 3.1
41	SamrGetDisplayEnumerationIndex	Windows NT 3.1
42	Reserved (not intended for network traffic)	-
43	Reserved (not intended for network traffic)	-
44	SamrGetUserDomainPasswordInformation	Windows NT 3.1
45	SamrRemoveMemberFromForeignDomain	Windows NT 3.1
46	SamrQueryInformationDomain2	Windows NT 3.5
47	SamrQueryInformationUser2	Windows NT 3.5
48	SamrQueryDisplayInformation2	Windows NT 3.5

Opnum	Friendly name	Product
49	SamrGetDisplayEnumerationIndex2	Windows NT 3.5
50	SamrCreateUser2InDomain	Windows NT 3.5
51	SamrQueryDisplayInformation3	Windows NT 3.5
52	SamrAddMultipleMembersToAlias	Windows NT 3.51
53	SamrRemoveMultipleMembersFromAlias	Windows NT 3.51
54	SamrOemChangePasswordUser2	Windows NT 3.51
55	SamrUnicodeChangePasswordUser2	Windows NT 3.51
56	SamrGetDomainPasswordInformation	Windows NT 3.51
57	SamrConnect2	Windows NT 3.51
58	SamrSetInformationUser2	Windows NT 3.51
59	Reserved (not intended for network traffic)	-
60	Reserved (not intended for network traffic)	-
61	Reserved (not intended for network traffic)	-
62	SamrConnect4	Windows 2000
63	SamrUnicodeChangePasswordUser3 Reserved (never released a client that called this)	Windows XP
64	SamrConnect5	Windows XP
65	SamrRidToSid	Windows XP
66	SamrSetDSRMPassword	Windows 2000 SP2 and Windows XP
67	SamrValidatePassword	Windows Server 2003
68	Reserved (not intended for network traffic)	-
69	Reserved (not intended for network traffic)	-

1.7.2 Method Versioning

Requesters determine whether a method is supported by attempting to invoke the method. If the transport, RPC, returns the error RPC_S_PROCNUM_OUT_OF_RANGE (defined in section 2.2.1.16), the requester MUST try the deprecated equivalent of the invoked method if there is one. The following table describes the deprecated method to invoke if the current method is not supported.<5>

Current method	Old method (in order of preference)
SamrQueryInformationDomain2	SamrQueryInformationDomain
SamrCreateUser2InDomain	SamrCreateUserInDomain

Current method	Old method (in order of preference)
SamrQueryDisplayInformation3	SamrQueryDisplayInformation2 SamrQueryDisplayInformation
SamrGetDisplayEnumerationIndex2	SamrGetDisplayEnumerationIndex
SamrQueryInformationUser2	SamrQueryInformationUser
SamrSetInformationUser2	SamrSetInformationUser
SamrConnect5	SamrConnect4 SamrConnect2

1.7.3 Introduction to Information Levels

The set, query, and selective enumerate patterns of methods use information levels to communicate the set of object attributes that are to be set or queried in the method request. Information levels are enumerations (that is, numerical values).

It is possible that future versions of the protocol will introduce new information levels, creating a situation in which a requester may specify an information level that is not supported by the responder. This situation can occur, for example, when a later requester communicates with an earlier responder.<6>

1.7.4 SamrConnect5 Versioning

The **SamrConnect5** method allows the requester and responder to indicate their protocol revision level. This information enables the requester to make decisions about the methods to call; similarly, this information can enable the responder to make decisions about particular behaviors to exhibit to the requester. For more information, see section 2.2.3.15.<7>

1.8 Vendor-Extensible Fields

The SAM Remote Protocol (Client-to-Server) contains no vendor-extensible fields.

1.9 Standards Assignments

The SAM Remote Protocol (Client-to-Server) contains no standards assignments.

2 Messages

2.1 Transport

This protocol configures the RPC runtime to perform a strict Network Data Representation (NDR) data consistency check at target level 5.0, as specified in [MS-RPCE] section 3.

This protocol uses UUID 12345778-1234-ABCD-EF00-0123456789AC, which is a private allocation by Microsoft, to identify the RPC interface.

This protocol enables the ms_union extension that is specified in [MS-RPCE] section 2.2.4.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles that are created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

This protocol uses the following RPC protocol sequences:<8>

- RPC over SMB, as specified in [MS-RPCE] section 2.1.1.2

  This protocol uses the pipe name "\PIPE\samr" for the endpoint name.<9>

- RPC over TCP

  This protocol uses RPC dynamic endpoints, as specified in [C706-Ch6RPCCallModel].

This protocol MUST use the UUID as specified previously. The RPC version number is 1.0.

The protocol uses the underlying RPC protocol to retrieve the identity of the requester that made the method call, as specified in [MS-RPCE] section 3.3.3.4.3. The responder SHOULD<10> use this identity to perform method-specific **access checks**, as specified in the message processing section of each method.<11>

For the **SamrValidatePassword** method, the requester SHOULD use transport security to encrypt the message because the message contents contain cleartext password data. That is, the requester SHOULD use a SPNEGO security provider, as specified in [MS-RPCE] section 3.1.1.5.4, and SHOULD use the packet authentication level, as specified in [MS-RPCE] section 3.3.1.5.2.<12>

2.2 Common Data Types

2.2.1 Constant Value Definitions

This section is used as a reference from one or more message syntax and message processing sections.

2.2.1.1 Common ACCESS_MASK Values

These values specify an access control that is applicable to all object types exposed by this protocol. These values can appear in the **Mask** field of an **access control entry (ACE)**, or in methods to obtain a handle; for example, **SamrConnect5**.

Constant/value	Description
DELETE 0x00010000	Specifies the ability to delete the object.

Constant/value	Description
READ_CONTROL 0x00020000	Specifies the ability to read the security descriptor.
WRITE_DAC 0x00040000	Specifies the ability to update the discretionary access control list (DACL) of the security descriptor.
WRITE_OWNER 0x00080000	Specifies the ability to update the **Owner** field of the security descriptor.
ACCESS_SYSTEM_SECURITY 0x01000000	Specifies access to the system security portion of the security descriptor.
MAXIMUM_ALLOWED 0x02000000	Indicates that the caller is requesting the most access possible to the object.

For more information, see [MS-DTYP] section **2.4.3**. Values that are not listed have no meaning in this protocol.

2.2.1.2 Generic ACCESS_MASK Values

These values appear in methods that are used to obtain a handle; for example, **SamrConnect5**. They are translated by the responder into specific ACCESS_MASK values. For more information on object-specific semantics, see sections 2.2.1.3, 2.2.1.4, 2.2.1.5, 2.2.1.6, and 2.2.1.7.

Constant/value	Description
GENERIC_READ 0x80000000	Specifies access control suitable for reading the object.
GENERIC_WRITE 0x40000000	Specifies access control suitable for updating attributes on the object.
GENERIC_EXECUTE 0x20000000	Specifies access control suitable for executing some action on the object.
GENERIC_ALL 0x10000000	Specifies all defined access control on the object.

2.2.1.3 Server ACCESS_MASK Values

These are the specific values available to describe the access control on a server object. A bitwise OR operation can be performed on these values, along with values from section 2.2.1.1. For more information on the message processing of these values, see section 3.2.5.1.1.

Constant/value	Description
SAM_SERVER_CONNECT 0x00000001	Specifies access control to obtain a server handle.
SAM_SERVER_SHUTDOWN 0x00000002	Does not specify any access control.
SAM_SERVER_INITIALIZE 0x00000004	Does not specify any access control.

Constant/value	Description
SAM_SERVER_CREATE_DOMAIN 0x00000008	Does not specify any access control.
SAM_SERVER_ENUMERATE_DOMAINS 0x00000010	Specifies access control to view domain objects.
SAM_SERVER_LOOKUP_DOMAIN 0x00000020	Specifies access control to perform SID-to-name translation.
SAM_SERVER_ALL_ACCESS 0x000F003F	The specified accesses for a GENERIC_ALL request.
SAM_SERVER_READ 0x00020010	The specified accesses for a GENERIC_READ request.
SAM_SERVER_WRITE 0x0002000E	The specified accesses for a GENERIC_WRITE request.
SAM_SERVER_EXECUTE 0x00020021	The specified accesses for a GENERIC_EXECUTE request.

2.2.1.4 Domain ACCESS_MASK Values

These are the specific values available to describe the access control on a domain object. A bitwise OR operation can be performed on these values, along with values from section 2.2.1.1. For more information on the message processing of these values, see section 3.2.5.1.2.

Constant/value	Description
DOMAIN_READ_PASSWORD_PARAMETERS 0x00000001	Specifies access control to read password policy.
DOMAIN_WRITE_PASSWORD_PARAMS 0x00000002	Specifies access control to write password policy.
DOMAIN_READ_OTHER_PARAMETERS 0x00000004	Specifies access control to read attributes not related to password policy.
DOMAIN_WRITE_OTHER_PARAMETERS 0x00000008	Specifies access control to write attributes not related to password policy.
DOMAIN_CREATE_USER 0x00000010	Specifies access control to create a user object.
DOMAIN_CREATE_GROUP 0x00000020	Specifies access control to create a group object.
DOMAIN_CREATE_ALIAS 0x00000040	Specifies access control to create an alias object.
DOMAIN_GET_ALIAS_MEMBERSHIP 0x00000080	Specifies access control to read the alias membership of a set of SIDs.
DOMAIN_LIST_ACCOUNTS 0x00000100	Specifies access control to enumerate objects.

Constant/value	Description
DOMAIN_LOOKUP 0x00000200	Specifies access control to look up objects by name and SID.
DOMAIN_ADMINISTER_SERVER 0x00000400	Specifies access control to sundry administrative operations on the responder.
DOMAIN_ALL_ACCESS 0x000F07FF	The specified accesses for a GENERIC_ALL request.
DOMAIN_READ 0x00020084	The specified accesses for a GENERIC_READ request.
DOMAIN_ALL_WRITE 0x0002047A	The specified accesses for a GENERIC_WRITE request.
DOMAIN_ALL_EXECUTE 0x00020301	The specified accesses for a GENERIC_EXECUTE request.

2.2.1.5 Group ACCESS_MASK Values

These are the specific values available to describe the access control on a group object. A bitwise OR operation can be performed on these values, along with values from section 2.2.1.1. For more information on the message processing of these values, see section 3.2.5.1.6.

Constant/value	Description
GROUP_READ_INFORMATION 0x00000001	Specifies the ability to read various attributes.
GROUP_WRITE_ACCOUNT 0x00000002	Specifies the ability to write various attributes, not including the **member** attribute.
GROUP_ADD_MEMBER 0x00000004	Specifies the ability to add a value to the **member** attribute.
GROUP_REMOVE_MEMBER 0x00000008	Specifies the ability to remove a value from the **member** attribute.
GROUP_LIST_MEMBERS 0x00000010	Specifies the ability to read the values of the **member** attribute.
GROUP_ALL_ACCESS 0x000F001F	The specified accesses for a GENERIC_ALL request.
GROUP_ALL_READ 0x00020010	The specified accesses for a GENERIC_READ request.
GROUP_ALL_WRITE 0x0002000E	The specified accesses for a GENERIC_WRITE request.
GROUP_ALL_EXECUTE 0x00020001	The specified accesses for a GENERIC_EXECUTE request.

2.2.1.6 Alias ACCESS_MASK Values

These are the specific values available to describe the access control on an alias object. A bitwise OR operation can be performed on these values, along with values from section 2.2.1.1. For more information on the message processing of these values, see section 3.2.5.1.8.

Constant/value	Description
ALIAS_ADD_MEMBER 0x00000001	Specifies the ability to add a value to the **member** attribute.
ALIAS_REMOVE_MEMBER 0x00000002	Specifies the ability to remove a value from the **member** attribute.
ALIAS_LIST_MEMBERS 0x00000004	Specifies the ability to read the **member** attribute.
ALIAS_READ_INFORMATION 0x00000008	Specifies the ability to read various attributes, except the **member** attribute.
ALIAS_WRITE_ACCOUNT 0x00000010	Specifies the ability to write various attributes, not including the **member** attribute.
ALIAS_ALL_ACCESS 0x000F001F	The specified accesses for a GENERIC_ALL request.
ALIAS_ALL_READ 0x00020004	The specified accesses for a GENERIC_READ request.
ALIAS_ALL_WRITE 0x00020013	The specified accesses for a GENERIC_WRITE request.
ALIAS_ALL_EXECUTE 0x00020008	The specified accesses for a GENERIC_EXECUTE request.

2.2.1.7 User ACCESS_MASK Values

These are the specific values available to describe the access control on a user object. A bitwise OR operation can be performed on these values, along with values from section 2.2.1.1. For more information on the message processing of these values, see section 3.2.5.1.9.

Constant/value	Description
USER_READ_GENERAL 0x00000001	Specifies the ability to read sundry attributes.
USER_READ_PREFERENCES 0x00000002	Specifies the ability to read self-service attributes.
USER_WRITE_PREFERENCES 0x00000004	Specifies the ability to write self-service attributes.
USER_READ_LOGON 0x00000008	Specifies the ability to read attributes related to logon statistics.
USER_READ_ACCOUNT 0x00000010	Specifies the ability to read attributes related to the administration of the user object.

Constant/value	Description
USER_WRITE_ACCOUNT 0x00000020	Specifies the ability to write attributes related to the administration of the user object.
USER_CHANGE_PASSWORD 0x00000040	Specifies the ability to change the user's password.
USER_FORCE_PASSWORD_CHANGE 0x00000080	Specifies the ability to set the user's password.
USER_LIST_GROUPS 0x00000100	Specifies the ability to query the membership of the user object.
USER_READ_GROUP_INFORMATION 0x00000200	Does not specify any access control.
USER_WRITE_GROUP_INFORMATION 0x00000400	Does not specify any access control.
USER_ALL_ACCESS 0x000F07FF	The specified accesses for a GENERIC_ALL request.
USER_ALL_READ 0x0002031A	The specified accesses for a GENERIC_READ request.
USER_ALL_WRITE 0x00020044	The specified accesses for a GENERIC_WRITE request.
USER_ALL_EXECUTE 0x00020041	The specified accesses for a GENERIC_EXECUTE request.

2.2.1.8 USER_ALL Values

USER_ALL values are used in the WhichFields bitfield in the **SAMPR_USER_ALL_INFORMATION** structure. All bits can be combined with a logical OR in any combination. Each bit, if set, indicates that a particular field of **SAMPR_USER_ALL_INFORMATION** must be processed by the responder. If not set, the responder MUST ignore the field. The following table also includes the bit-to-field association in the last column.

Constant/value	Description
USER_ALL_USERNAME 0x00000001	UserName
USER_ALL_FULLNAME 0x00000002	FullName
USER_ALL_USERID 0x00000004	UserId
USER_ALL_PRIMARYGROUPID 0x00000008	PrimaryGroupId
USER_ALL_ADMINCOMMENT 0x00000010	AdminComment
USER_ALL_USERCOMMENT	UserComment

Constant/value	Description
0x00000020	
USER_ALL_HOMEDIRECTORY 0x00000040	HomeDirectory
USER_ALL_HOMEDIRECTORYDRIVE 0x00000080	HomeDirectoryDrive
USER_ALL_SCRIPTPATH 0x00000100	ScriptPath
USER_ALL_PROFILEPATH 0x00000200	ProfilePath
USER_ALL_WORKSTATIONS 0x00000400	WorkStations
USER_ALL_LASTLOGON 0x00000800	LastLogon
USER_ALL_LASTLOGOFF 0x00001000	LastLogoff
USER_ALL_LOGONHOURS 0x00002000	LogonHours
USER_ALL_BADPASSWORDCOUNT 0x00004000	BadPasswordCount
USER_ALL_LOGONCOUNT 0x00008000	LogonCount
USER_ALL_PASSWORDCANCHANGE 0x00010000	PasswordCanChange
USER_ALL_PASSWORDMUSTCHANGE 0x00020000	PasswordMustChange
USER_ALL_PASSWORDLASTSET 0x00040000	PasswordLastSet
USER_ALL_ACCOUNTEXPIRES 0x00080000	AccountExpires
USER_ALL_USERACCOUNTCONTROL 0x00100000	UserAccountControl
USER_ALL_PARAMETERS 0x00200000	Parameters
USER_ALL_COUNTRYCODE 0x00400000	CountryCode
USER_ALL_CODEPAGE 0x00800000	CodePage
USER_ALL_NTPASSWORDPRESENT 0x01000000	NtPasswordPresent

Constant/value	Description
USER_ALL_LMPASSWORDPRESENT 0x02000000	LmPasswordPresent
USER_ALL_PASSWORDEXPIRED 0x08000000	PasswordExpired

2.2.1.9 ACCOUNT_TYPE Values

Account type values are associated with **accounts** and indicate the type of account. These values may not be combined through logical operations.

Constant/value	Description
SAM_GROUP_OBJECT 0x10000000	Represents a group object.
SAM_NON_SECURITY_GROUP_OBJECT 0x10000001	Represents a group object that is not used for authorization context generation.
SAM_ALIAS_OBJECT 0x20000000	Represents an alias object.
SAM_NON_SECURITY_ALIAS_OBJECT 0x20000001	Represents an alias object that is not used for authorization context generation.
SAM_USER_OBJECT 0x30000000	Represents a user object.
SAM_MACHINE_ACCOUNT 0x30000001	Represents a computer object.
SAM_TRUST_ACCOUNT 0x30000002	Represents a user object that is used for domain trusts.

2.2.1.10 SE_GROUP Attributes

These values are attributes of a security group membership and can be combined by using the bitwise OR operation. They are used by an access check mechanism to specify if the membership is to be used in an access check decision. The values can be set by using the **SamrSetMemberAttributesOfGroup** method.

Constant/value	Description
SE_GROUP_MANDATORY 0x00000001	The SID cannot have the **SE_GROUP_ENABLED** attribute removed.
SE_GROUP_ENABLED_BY_DEFAULT 0x00000002	The SID is enabled by default (rather than being added by an application).
SE_GROUP_ENABLED 0x00000004	The SID is enabled for access checks.

2.2.1.11 GROUP_TYPE Codes

These values specify the type of a group object. They are used in the **groupType** attribute. The values are mutually exclusive, except for the GROUP_TYPE_SECURITY_ENABLED bit, which can be combined using a logical OR with any other value.

Constant/value	Description
GROUP_TYPE_ACCOUNT_GROUP 0x00000002	Specifies that the group is an Account Group.
GROUP_TYPE_RESOURCE_GROUP 0x00000004	Specifies that the group is a Resource Group.
GROUP_TYPE_UNIVERSAL_GROUP 0x00000008	Specifies that the group is a Universal Group.
GROUP_TYPE_SECURITY_ENABLED 0x80000000	Specifies that the group's membership is to be included in an authorization context.
GROUP_TYPE_SECURITY_ACCOUNT 0x80000002	A combination of two of the bits shown above for the purposes of this specification.
GROUP_TYPE_SECURITY_RESOURCE 0x80000004	A combination of two of the bits shown above for the purposes of this specification.
GROUP_TYPE_SECURITY_UNIVERSAL 0x80000008	A combination of two of the bits shown above for the purposes of this specification.

2.2.1.12 USER_ACCOUNT Codes

These values are attributes of a user account and can be combined by using a bitwise OR operation. They are used in the UserAccountControl field for user objects. For more information, see section 2.2.7.1.

Constant/value	Description
USER_ACCOUNT_DISABLED 0x00000001	Specifies that the account is not enabled for authentication.
USER_HOME_DIRECTORY_REQUIRED 0x00000002	Specifies that the **homeDirectory** attribute is required.
USER_PASSWORD_NOT_REQUIRED 0x00000004	Specifies that the password-length policy does not apply to this user.
USER_TEMP_DUPLICATE_ACCOUNT 0x00000008	This bit is ignored by requesters and responders.
USER_NORMAL_ACCOUNT 0x00000010	Specifies that the user is not a computer object.
USER_MNS_LOGON_ACCOUNT 0x00000020	This bit is ignored by requesters and responders.
USER_INTERDOMAIN_TRUST_ACCOUNT 0x00000040	Specifies that the object represents a trust object. For more information about trust objects, see [MS-NLMP].

Constant/value	Description
USER_WORKSTATION_TRUST_ACCOUNT 0x00000080	Specifies that the object is a member workstation or server.
USER_SERVER_TRUST_ACCOUNT 0x00000100	Specifies that the object is a DC.
USER_DONT_EXPIRE_PASSWORD 0x00000200	Specifies that the maximum-password-age policy does not apply to this user.
USER_ACCOUNT_AUTO_LOCKED 0x00000400	Specifies that the account has been locked out.
USER_ENCRYPTED_TEXT_PASSWORD_ALLOWED 0x00000800	Specifies that the cleartext password is to be persisted.
USER_SMARTCARD_REQUIRED 0x00001000	Specifies that the user can authenticate only with a smart card.
USER_TRUSTED_FOR_DELEGATION 0x00002000	This bit is used by the Kerberos protocol. It indicates that the "OK as Delegate" ticket flag (described in [RFC4120] section 2.8) MUST be set.
USER_NOT_DELEGATED 0x00004000	This bit is used by the Kerberos protocol. It indicates that the ticket-granting tickets (TGTs) of this account and the service tickets obtained by this account are not marked as forwardable or proxiable when the forwardable or proxiable ticket flags are requested. For more information, see [RFC4120].
USER_USE_DES_KEY_ONLY 0x00008000	This bit is used by the Kerberos protocol. It indicates that only des-cbc-md5 or des-cbc-crc keys (as defined in [RFC3961]) are used in the Kerberos protocols for this account.
USER_DONT_REQUIRE_PREAUTH 0x00010000	This bit is used by the Kerberos protocol. It indicates that the account is not required to present valid pre-authentication data, as described in [RFC4120] section 7.5.2.
USER_PASSWORD_EXPIRED 0x00020000	Specifies that the password age on the user has exceeded the maximum password age policy.
USER_TRUSTED_TO_AUTHENTICATE_FOR_DELEGATION 0x00040000	This bit is used by the Kerberos protocol. When set, it indicates that the account (when running as a service) obtains an S4U2self service ticket (as specified in [MS-SFU]) with the forwardable flag set. If this bit is cleared, the forwardable flag is not set in the S4U2self service ticket.
USER_NO_AUTH_DATA_REQUIRED 0x00080000	This bit is used by the Kerberos protocol. It indicates that when the key distribution center (KDC) is issuing a service ticket for this account, the privilege attribute certificate (PAC) MUST NOT be included. For more

Constant/value	Description
	information, see [RFC4120].

2.2.1.13 UF_FLAG Codes

These values are attributes of a user account, as expressed at the data model level (see section 3.2.1 for the data model). See section 3.2.5.14.2 to map these values to USER_ACCOUNT values, and then see section 2.2.1.12 for a description.

Constant/value	Description
UF_SCRIPT 0x00000001	See description in introductory paragraph.
UF_ACCOUNTDISABLE 0x00000002	See description in introductory paragraph.
UF_HOMEDIR_REQUIRED 0x00000008	See description in introductory paragraph.
UF_LOCKOUT 0x00000010	See description in introductory paragraph.
UF_PASSWD_NOTREQD 0x00000020	See description in introductory paragraph.
UF_PASSWD_CANT_CHANGE 0x00000040	See description in introductory paragraph.
UF_ENCRYPTED_TEXT_PASSWORD_ALLOWED 0x00000080	See description in introductory paragraph.
UF_TEMP_DUPLICATE_ACCOUNT 0x00000100	See description in introductory paragraph.
UF_NORMAL_ACCOUNT 0x00000200	See description in introductory paragraph.
UF_INTERDOMAIN_TRUST_ACCOUNT 0x00000800	See description in introductory paragraph.
UF_WORKSTATION_TRUST_ACCOUNT 0x00001000	See description in introductory paragraph.
UF_SERVER_TRUST_ACCOUNT 0x00002000	See description in introductory paragraph.
UF_DONT_EXPIRE_PASSWD 0x00010000	See description in introductory paragraph.
UF_MNS_LOGON_ACCOUNT 0x00020000	See description in introductory paragraph.
UF_SMARTCARD_REQUIRED 0x00040000	See description in introductory paragraph.
UF_TRUSTED_FOR_DELEGATION	See description in introductory paragraph.

Constant/value	Description
0x00080000	
UF_NOT_DELEGATED 0x00100000	See description in introductory paragraph.
UF_USE_DES_KEY_ONLY 0x00200000	See description in introductory paragraph.
UF_DONT_REQUIRE_PREAUTH 0x00400000	See description in introductory paragraph.
UF_PASSWORD_EXPIRED 0x00800000	See description in introductory paragraph.
UF_TRUSTED_TO_AUTHENTICATE_FOR_DELEGATION 0x01000000	See description in introductory paragraph.
UF_NO_AUTH_DATA_REQUIRED 0x02000000	See description in introductory paragraph.

2.2.1.14 Predefined RIDs

These are predefined **RIDs** of users and groups. The description column briefly describes what the user or group is used for.

Constant/value	Description
DOMAIN_USER_RID_ADMIN 0x000001F4	Name: Administrator User for administering the computer or domain.
DOMAIN_USER_RID_GUEST 0x000001F5	Name: Guest User for guest access to the computer or domain.
DOMAIN_USER_RID_KRBTGT 0x000001F6	Name: krbtgt User for Key Distribution Center Service.
DOMAIN_GROUP_RID_USERS 0x00000201	Name: Domain Users A group that represents all domain users.
DOMAIN_GROUP_RID_COMPUTERS 0x00000203	Name: Domain Computers A group that represents all workstations and servers joined to the domain.
DOMAIN_GROUP_RID_CONTROLLERS 0x00000204	Name: Domain Controllers A group that represents all DCs in the domain.
DOMAIN_ALIAS_RID_ADMINS 0x00000220	Name: Administrators A group that has complete and unrestricted access to the computer or domain.

2.2.1.15 STATUS_ Codes

These values are return status codes from the responder. This section is provided as a reference for the message processing sections in section 3.2.5.

Constant/value	Description
STATUS_ACCESS_DENIED 0xC0000022	Returned when a requester has requested access to an object, but has not been granted those access rights.
STATUS_MORE_ENTRIES 0x00000105	Returned by enumeration methods to indicate that more information is available.
STATUS_SOME_NOT_MAPPED 0x00000107	Returned when some of the information to be translated has not been translated.
STATUS_NONE_MAPPED 0xC0000073	Returned when none of the information to be translated has been translated.
STATUS_WRONG_PASSWORD 0xC000006A	Returned when trying to update a password, and the value provided as the current password is not correct.
STATUS_ACCOUNT_LOCKED_OUT 0xC0000234	Returned when the user account has been automatically locked because too many invalid logon attempts or password change attempts have been requested.
STATUS_GROUP_EXISTS 0xC0000065	Returned when the specified group already exists.
STATUS_USER_EXISTS 0xC0000063	Returned when the specified account already exists.
STATUS_LM_CROSS_ENCRYPTION_REQUIRED 0xC000017F	Returned when the requester should retry the request using the current password LM hash as an encryption key. See section 3.2.5.10.1 for details.
STATUS_NT_CROSS_ENCRYPTION_REQUIRED 0xC000015D	Returned when the requester should retry the request using the current password NT hash as an encryption key. See section 3.2.5.10.1 for details.

2.2.1.16 Windows Error Code

This value is a return status code from the transport (RPC).

Constant/value	Description
RPC_S_PROCNUM_OUT_OF_RANGE 0x6D1	The responder does not implement the requested method.

2.2.1.17 AD ACCESS_MASK

These **access mask** values are specific to ACEs that apply to Active Directory objects. More information about these values is specified in [MS-ADTS] section 5.1.3.

Constant/value	Description
ACTRL_DS_LIST 0x00000004	Indicates the ability to read the children of an object in Active Directory.
ACTRL_DS_READ_PROP 0x00000010	Indicates the access control to read a property in Active Directory.

Constant/value	Description
ACTRL_DS_WRITE_PROP 0x00000020	Indicates the access control to write a property in Active Directory.
ACTRL_DS_DELETE_TREE 0x00000040	Indicates the ability to delete a tree of objects.
ACTRL_DS_CONTROL_ACCESS 0x00000100	Indicates the ability to perform an operation on an object as indicated by the **ObjectGuid** field in the ACE.

2.2.2 Basic Data Types

The following basic types are elementary to the SAM Remote Protocol (Client-to-Server), and are used in many methods. These types also appear in other protocols.

2.2.2.1 RPC_STRING, PRPC_STRING

The **RPC_STRING** structure holds a counted string encoded in the OEM code page.

```
typedef struct _RPC_STRING {
  unsigned short Length;
  unsigned short MaximumLength;
  [size_is(MaximumLength), length_is(Length)]
    char* Buffer;
} RPC_STRING,
 *PRPC_STRING;
```

Length: The size, in bytes, not including a NULL terminator, of the string contained in **Buffer**.

MaximumLength: The size, in bytes, of the **Buffer** member.

Buffer: A buffer containing a string encoded in the OEM code page. The string is counted (by the **Length** member), and therefore, is not NULL terminated.

2.2.2.2 RPC_UNICODE_STRING

The **RPC_UNICODE_STRING** structure holds a counted, UTF-16 encoded string.

```
typedef struct _RPC_UNICODE_STRING {
  unsigned short Length;
  unsigned short MaximumLength;
  [size_is(MaximumLength/2), length_is(Length/2)]
    wchar_t* Buffer;
} RPC_UNICODE_STRING,
 *PRPC_UNICODE_STRING;
```

Length: The size, in bytes, not including a NULL terminator, of the string contained in **Buffer**.

MaximumLength: The size, in bytes, of the **Buffer** member.

Buffer: A buffer containing a UTF-16 encoded string. The string is counted (by the **Length** member), and therefore, is not NULL terminated.

2.2.2.3 SID_IDENTIFIER_AUTHORITY

The **SID_IDENTIFIER_AUTHORITY** structure defines a substructure of **RPC_SID (section 2.2.2.4)**.

```
typedef struct _SID_IDENTIFIER_AUTHORITY {
  unsigned char Value[6];
} SID_IDENTIFIER_AUTHORITY,
 *PSID_IDENTIFIER_AUTHORITY;
```

Value: An array of 6 bytes that is the **IdentifierAuthority** member of an **RPC_SID** structure.

2.2.2.4 RPC_SID

The **RPC_SID** structure is a representation of a **SID** (as specified in [MS-DTYP] section **2.4.2**).

```
typedef struct _RPC_SID {
  unsigned char Revision;
  unsigned char SubAuthorityCount;
  SID_IDENTIFIER_AUTHORITY IdentifierAuthority;
  [size_is(SubAuthorityCount)] unsigned long SubAuthority[];
} RPC_SID,
 *PRPC_SID,
 **PPRPC_SID;
```

Individual member semantics are specified in [MS-DTYP] section **2.4.2**.

2.2.2.5 OLD_LARGE_INTEGER

The **OLD_LARGE_INTEGER** structure defines a 64-bit value that is accessible in two 4-byte chunks.

```
typedef struct _OLD_LARGE_INTEGER {
  unsigned long LowPart;
  long HighPart;
} OLD_LARGE_INTEGER,
 *POLD_LARGE_INTEGER;
```

LowPart: The least-significant portion of a 64-bit value.

HighPart: The most-significant portion of a 64-bit value.

2.2.2.6 SID_NAME_USE

An enumeration to specify the type of account that a SID references.

```
typedef  enum _SID_NAME_USE
{
  SidTypeUser = 1,
  SidTypeGroup,
  SidTypeDomain,
  SidTypeAlias,
  SidTypeWellKnownGroup,
  SidTypeDeletedAccount,
  SidTypeInvalid,
  SidTypeUnknown
} SID_NAME_USE,
 *PSID_NAME_USE;
```

SidTypeUser: Indicates a user object.

SidTypeGroup: Indicates a group object.

SidTypeDomain: Indicates a domain object.

SidTypeAlias: Indicates an alias object.

SidTypeWellKnownGroup: Indicates an object whose SID is invariant.

SidTypeDeletedAccount: Indicates an object that has been deleted.

SidTypeInvalid: This member is unused.

SidTypeUnknown: Indicates that the type of object could not be determined. For example, no object with that SID exists.

2.2.3 Miscellaneous Protocol-Specific Types

These types are specific to the SAM Remote Protocol (Client-to-Server). Many types are used by multiple methods, while others may only be used by one method. This section is useful when used as a reference while reading the method syntax in section 3.2.5.

2.2.3.1 PSAMPR_SERVER_NAME

An RPC handle that is represented by a zero-terminated, UTF-16 encoded string. [UNICODE3.1] describes the Unicode encoding.

The string represents the network address of the responder.

This type is declared as follows:

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

2.2.3.2 SAMPR_HANDLE

An RPC context handle, as specified in [C706-Ch6RPCCallModel], "Binding Handles", that is used to share a session between method calls.

This type is declared as follows:

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

2.2.3.3 ENCRYPTED_LM_OWF_PASSWORD, ENCRYPTED_NT_OWF_PASSWORD

The **ENCRYPTED_LM_OWF_PASSWORD** structure defines a block of encrypted data used in various methods to communicate sensitive information.

```
typedef struct _ENCRYPTED_LM_OWF_PASSWORD {
  char data[16];
} ENCRYPTED_LM_OWF_PASSWORD,
 *PENCRYPTED_LM_OWF_PASSWORD,
 ENCRYPTED_NT_OWF_PASSWORD,
 *PENCRYPTED_NT_OWF_PASSWORD;
```

data: 16 bytes of unstructured data used to hold an encrypted 16-byte hash (either an **LM hash** or **NT hash**). The encryption algorithm is specified in section 2.2.11.1. The methods specified in sections 3.2.5.10 and 3.2.5.13.7 use this structure and specify the type of hash and the encryption key.

2.2.3.4 SAMPR_ULONG_ARRAY

The **SAMPR_ULONG_ARRAY** structure holds a counted array of unsigned long values.

```
typedef struct _SAMPR_ULONG_ARRAY {
  unsigned long Count;
  [size_is(Count)] unsigned long* Element;
} SAMPR_ULONG_ARRAY,
 *PSAMPR_ULONG_ARRAY;
```

Count: The number of elements in **Element**. If zero, **Element** MUST be ignored. If nonzero, **Element** MUST point to at least Count * sizeof(unsigned long) bytes of memory.

Element: A pointer to an array of unsigned integers with **Count** elements. The semantic meaning is dependent on the method in which the structure is being used.

2.2.3.5 SAMPR_SID_INFORMATION

The **SAMPR_SID_INFORMATION** structure holds a SID pointer.

```
typedef struct _SAMPR_SID_INFORMATION {
  PRPC_SID SidPointer;
} SAMPR_SID_INFORMATION,
 *PSAMPR_SID_INFORMATION;
```

SidPointer: A pointer to a SID value, as described in section 2.2.2.4.

2.2.3.6 SAMPR_PSID_ARRAY

The **SAMPR_PSID_ARRAY** structure holds an array of SID values.

```
typedef struct _SAMPR_PSID_ARRAY {
  [range(0,1024)] unsigned long Count;
  [size_is(Count)] PSAMPR_SID_INFORMATION Sids;
} SAMPR_PSID_ARRAY,
 *PSAMPR_PSID_ARRAY;
```

Count: The number of elements in **Sids**. If zero, **Sids** MUST be ignored. If nonzero, **Sids** MUST point to at least Count * sizeof(SAMPR_SID_INFORMATION) bytes of memory.

Sids: An array of pointers to SID values. For more information, see section 2.2.3.5.

2.2.3.7 SAMPR_RETURNED_USTRING_ARRAY

The **SAMPR_RETURNED_USTRING_ARRAY** structure holds an array of counted UTF-16 encoded strings.

```
typedef struct _SAMPR_RETURNED_USTRING_ARRAY {
  unsigned long Count;
  [size_is(Count)] PRPC_UNICODE_STRING Element;
} SAMPR_RETURNED_USTRING_ARRAY,
 *PSAMPR_RETURNED_USTRING_ARRAY;
```

Count: The number of elements in **Element**. If zero, **Element** MUST be ignored. If nonzero, **Element** MUST point to at least Count * sizeof(RPC_UNICODE_STRING) bytes of memory.

Element: Array of counted strings (see section 2.2.2.2). The semantic meaning is method-dependent.

2.2.3.8 SAMPR_RID_ENUMERATION

The **SAMPR_RID_ENUMERATION** structure holds the name and RID information about an account.

```
typedef struct _SAMPR_RID_ENUMERATION {
  unsigned long RelativeId;
  RPC_UNICODE_STRING Name;
} SAMPR_RID_ENUMERATION,
 *PSAMPR_RID_ENUMERATION;
```

RelativeId: A RID.

Name: The UTF-16 encoded name of the account that is associated with **RelativeId**.

2.2.3.9 SAMPR_ENUMERATION_BUFFER

The **SAMPR_ENUMERATION_BUFFER** structure holds an array of **SAMPR_RID_ENUMERATION** elements.

```
typedef struct _SAMPR_ENUMERATION_BUFFER {
  unsigned long EntriesRead;
  [size_is(EntriesRead)] PSAMPR_RID_ENUMERATION Buffer;
} SAMPR_ENUMERATION_BUFFER,
 *PSAMPR_ENUMERATION_BUFFER;
```

EntriesRead: The number of elements in **Buffer**. If zero, **Buffer** MUST be ignored. If nonzero, **Buffer** MUST point to at least EntriesRead * sizeof(SAMPR_RID_ENUMERATION) bytes of memory.

Buffer: An array of **SAMPR_RID_ENUMERATION** elements.

2.2.3.10 SAMPR_SR_SECURITY_DESCRIPTOR

The **SAMPR_SR_SECURITY_DESCRIPTOR** structure holds a formatted **security descriptor**.

```
typedef struct _SAMPR_SR_SECURITY_DESCRIPTOR {
  [range(0, 256 * 1024)] unsigned long Length;
  [size_is(Length)] unsigned char* SecurityDescriptor;
} SAMPR_SR_SECURITY_DESCRIPTOR,
 *PSAMPR_SR_SECURITY_DESCRIPTOR;
```

Length: The size, in bytes, of SecurityDescriptor. If zero, SecurityDescriptor MUST be ignored. The maximum size of 256 * 1024 is an arbitrary value chosen to limit the amount of memory a requester can force the responder to allocate.

SecurityDescriptor: A binary format per the **SECURITY_DESCRIPTOR** format in [MS-DTYP] section **2.4.6**.

2.2.3.11 GROUP_MEMBERSHIP

The **GROUP_MEMBERSHIP** structure holds information on a group membership.

```
typedef struct _GROUP_MEMBERSHIP {
  unsigned long RelativeId;
  unsigned long Attributes;
} GROUP_MEMBERSHIP,
 *PGROUP_MEMBERSHIP;
```

RelativeId: A RID that represents one membership value.

Attributes: Characteristics about the membership represented as a bit mask. Values are defined in section 2.2.1.10.

2.2.3.12 SAMPR_GET_GROUPS_BUFFER

The **SAMPR_GET_GROUPS_BUFFER** structure represents the members of a group.

```
typedef struct _SAMPR_GET_GROUPS_BUFFER {
  unsigned long MembershipCount;
  [size_is(MembershipCount)] PGROUP_MEMBERSHIP Groups;
} SAMPR_GET_GROUPS_BUFFER,
 *PSAMPR_GET_GROUPS_BUFFER;
```

MembershipCount: The number of elements in **Groups**. If zero, **Groups** MUST be ignored. If nonzero, **Groups** MUST point to at least MembershipCount * sizeof(GROUP_MEMBERSHIP) bytes of memory.

Groups: An array to hold information about the members of the group.

2.2.3.13 SAMPR_GET_MEMBERS_BUFFER

The **SAMPR_GET_MEMBERS_BUFFER** structure represents the membership of a group.

```
typedef struct _SAMPR_GET_MEMBERS_BUFFER {
  unsigned long MemberCount;
  [size_is(MemberCount)] unsigned long* Members;
  [size_is(MemberCount)] unsigned long* Attributes;
} SAMPR_GET_MEMBERS_BUFFER,
 *PSAMPR_GET_MEMBERS_BUFFER;
```

MemberCount: The number of elements in **Members** and **Attributes**. If zero, **Members** and **Attributes** MUST be ignored. If nonzero, **Members** and **Attributes** MUST point to at least MemberCount * sizeof(unsigned long) bytes of memory.

Members: An array of RIDs.

Attributes: Characteristics about the membership, represented as a bit mask. Values are defined in section 2.2.1.8.

2.2.3.14 SAMPR_REVISION_INFO_V1

The **SAMPR_REVISION_INFO_V1** structure is used to communicate the revision and capabilities of requester and responder. For more information, see **SamrConnect5**.

```
typedef struct _SAMPR_REVISION_INFO_V1 {
  unsigned long Revision;
  unsigned long SupportedFeatures;
} SAMPR_REVISION_INFO_V1,
 *PSAMPR_REVISION_INFO_V1;
```

Revision: The revision of the requester or responder side of this protocol (depending on which side sends the structure). The value MUST be one of the following:

Value	Meaning
1	Pre-Windows 2000
2	Windows 2000
3	Windows XP, Windows Server 2003, and Windows Vista

SupportedFeatures: A bitfield. When sent from the requester, this field MUST be zero and ignored on receipt. When returned from the responder, the following fields are handled by the requester; all other bits are ignored and MUST be zero when returned from the requester.

Value	Meaning
0x00000001	On receipt by the requester, this value, when set, indicates that RID values returned from the responder MUST NOT be concatenated with the domain SID to create the SID for the account referenced by the RID. Instead, the requester MUST call **SamrRidToSid** to obtain the SID. This field can be combined with other bits using a logical OR. See the Windows Behavior citation below for more information (about Windows implementations).
0x00000002	Reserved. See the Windows Behavior citation below for additional details.
0x00000004	Reserved. See the Windows Behavior citation below for additional details.

The following Windows Behavior is relevant to the **SupportedFeatures** field.<13>

2.2.3.15 SAMPR_REVISION_INFO

The **SAMPR_REVISION_INFO** union holds revision information structures that are used in the **SamrConnect5** method.

```
typedef
[switch_type(unsigned long)]
union {
  [case(1)]
    SAMPR_REVISION_INFO_V1 V1;
} SAMPR_REVISION_INFO,
 *PSAMPR_REVISION_INFO;
```

V1: Version 1 revision information, as described in **SAMPR_REVISION_INFO_V1 (section 2.2.3.14)**.

2.2.3.16 USER_DOMAIN_PASSWORD_INFORMATION

The **USER_DOMAIN_PASSWORD_INFORMATION** structure contains domain fields.

```
typedef struct _USER_DOMAIN_PASSWORD_INFORMATION {
  unsigned short MinPasswordLength;
  unsigned long PasswordProperties;
} USER_DOMAIN_PASSWORD_INFORMATION,
 *PUSER_DOMAIN_PASSWORD_INFORMATION;
```

For information on each field, see section 2.2.4.1.

2.2.4 Domain Query/Set Data Types

The structures in this section relate to the following methods:

- **SamrQueryInformationDomain**
- **SamrQueryInformationDomain2**
- **SamrSetInformationDomain**

The model of the methods is for the requester to specify an enumeration that indicates the attributes to be either set or queried. There is duplication among the structures that contain the attributes. For a description of each attribute that is common among structures, see section 2.2.4.1.

2.2.4.1 Domain Fields

There are a number of domain-related structures that use the same fields, as denoted by their field names. This section specifies all such fields. The structures group the available set of domain attributes in different ways to allow the requester to control which attributes are queried or set. While each structure may have a different subset of these attributes, they all draw from this same set of attributes, detailed below.

AliasCount: A 32-bit unsigned integer indicating the number of alias objects in the domain. This field is read-only.

CreationTime: A 64-bit timestamp, equivalent to a **FILETIME**, indicating the time of creation for the domain in 100-nanosecond intervals from 12:00 A.M., January 1, 1601 (UTC). This field is read-only.

DomainModifiedCount: A 64-bit update sequence number representing the number of database updates relevant to the Windows NT 4.0 replication protocol. This field is read-only.

DomainName: A counted Unicode string of type **RPC_UNICODE_STRING**, containing the NetBIOS name of the domain. This field is read-only.

DomainServerRole: An enumerated value (see **DOMAIN_SERVER_ROLE**) indicating the role of the server in the domain. Possible values are Primary Domain Controller (DomainServerRolePrimary) or Backup Domain Controller (DomainServerRoleBackup).

DomainServerState: An enumerated value (see **DOMAIN_SERVER_ENABLE_STATE**) indicating whether the responder is enabled. Possible values are enabled (DomainServerEnabled) or disabled (DomainServerDisabled).<14>

ForceLogoff: A 64-bit value, with **Delta-Time** syntax, indicating the policy setting for the amount of time that an interactive logon session is allowed to continue.

GroupCount: A 32-bit unsigned integer indicating the number of group accounts. This field is read-only.

LockoutDuration: A 64-bit value, with Delta-Time syntax, indicating the duration for which an account is locked out before being automatically reset to an unlocked state.

LockoutObservationWindow: A 64-bit value, with Delta-Time syntax, indicating the time period in which bad password attempts are counted without resetting to the count to zero.

LockoutThreshold: A 16-bit unsigned integer indicating the number of bad password attempts within a LockoutObservationWindow that will cause an account to be locked out.

MaxPasswordAge: A 64-bit value, with Delta-Time syntax, indicating the policy setting for the maximum time allowed before a password reset or change is required.

MinPasswordAge: A 64-bit value, with Delta-Time syntax, indicating the policy setting for the minimum time allowed before a password change operation is allowed.

MinPasswordLength: A 16-bit unsigned integer indicating the minimum password length policy setting.

ModifiedCountAtLastPromotion: A 64-bit update sequence number representing the number of database updates relevant to the Windows NT 4.0 replication protocol that had occurred at the time when the current responder obtained the **PDC** role (see [MS-ADTS] section 7.1.5.3 for more information on the PDC role). This field is read-only.

OemInformation: A counted Unicode string of type **RPC_UNICODE_STRING** that requesters can set to any value. There are no known scenarios that use this field.

PasswordHistoryLength: A 16-bit unsigned integer indicating the policy setting for the password history length.

PasswordProperties: A 32-bit bitfield indicating the password properties policy setting. The defined bits are shown in the following table. All bits can be combined using a logical OR in any combination. Undefined bits (bits not specified in the table) SHOULD<15> be an error condition on receipt at the responder and ignored by the requester.

Name/value	Description
DOMAIN_PASSWORD_COMPLEX 0x00000001	The responder enforces password complexity policy. See section 3.2.1.7.2 for details of the password policy.
DOMAIN_PASSWORD_NO_ANON_CHANGE 0x00000002	Reserved. No effect on password policy.
DOMAIN_PASSWORD_NO_CLEAR_CHANGE 0x00000004	Change-password methods that provide the cleartext password are disabled by the responder.
DOMAIN_LOCKOUT_ADMINS 0x00000008	Reserved. No effect on password policy.
DOMAIN_PASSWORD_STORE_CLEARTEXT 0x00000010	The responder MUST store the cleartext password, not just the computed hashes.
DOMAIN_REFUSE_PASSWORD_CHANGE 0x00000020	Reserved. No effect on password policy.

ReplicaSourceNodeName: A counted Unicode string of type **RPC_UNICODE_STRING** that represents a replication partner.

UasCompatibilityRequired: A 1-byte value that, if nonzero, indicates that **UAS Compatibility** mode is effective; if zero, UAS Compatibility mode is not effective. This field is read-only and the default value is nonzero.

UserCount: A 32-bit unsigned integer indicating the number of group accounts. This field is read-only.

2.2.4.2 DOMAIN_SERVER_ENABLE_STATE

The **DOMAIN_SERVER_ENABLE_STATE** enumeration describes the enabled or disabled state of a responder.

```
typedef  enum _DOMAIN_SERVER_ENABLE_STATE
{
  DomainServerEnabled = 1,
  DomainServerDisabled
} DOMAIN_SERVER_ENABLE_STATE,
 *PDOMAIN_SERVER_ENABLE_STATE;
```

DomainServerEnabled: The responder is considered "enabled" to the requester.

DomainServerDisabled: This field is not used.

2.2.4.3 DOMAIN_STATE_INFORMATION

The **DOMAIN_STATE_INFORMATION** structure holds the enabled/disabled state of the responder.

```
typedef struct _DOMAIN_STATE_INFORMATION {
  DOMAIN_SERVER_ENABLE_STATE DomainServerState;
} DOMAIN_STATE_INFORMATION,
 *PDOMAIN_STATE_INFORMATION;
```

For information on each field, see section 2.2.4.1.

2.2.4.4 DOMAIN_SERVER_ROLE

The **DOMAIN_SERVER_ROLE** enumeration indicates whether a responder is a PDC.

```
typedef  enum _DOMAIN_SERVER_ROLE
{
  DomainServerRoleBackup = 2,
  DomainServerRolePrimary = 3
} DOMAIN_SERVER_ROLE,
 *PDOMAIN_SERVER_ROLE;
```

DomainServerRoleBackup: The DC is not the PDC.

DomainServerRolePrimary: The DC is the PDC.

2.2.4.5 DOMAIN_PASSWORD_INFORMATION

The **DOMAIN_PASSWORD_INFORMATION** structure contains domain fields.

```
typedef struct _DOMAIN_PASSWORD_INFORMATION {
  unsigned short MinPasswordLength;
  unsigned short PasswordHistoryLength;
  unsigned long PasswordProperties;
  OLD_LARGE_INTEGER MaxPasswordAge;
  OLD_LARGE_INTEGER MinPasswordAge;
} DOMAIN_PASSWORD_INFORMATION,
 *PDOMAIN_PASSWORD_INFORMATION;
```

For information on each field, see section 2.2.4.1.

2.2.4.6 DOMAIN_LOGOFF_INFORMATION

The **DOMAIN_LOGOFF_INFORMATION** structure contains domain fields.

```
typedef struct _DOMAIN_LOGOFF_INFORMATION {
  OLD_LARGE_INTEGER ForceLogoff;
} DOMAIN_LOGOFF_INFORMATION,
 *PDOMAIN_LOGOFF_INFORMATION;
```

For information on each field, see section 2.2.4.1.

2.2.4.7 DOMAIN_SERVER_ROLE_INFORMATION

The **DOMAIN_SERVER_ROLE_INFORMATION** structure contains domain fields.

```
typedef struct _DOMAIN_SERVER_ROLE_INFORMATION {
  DOMAIN_SERVER_ROLE DomainServerRole;
} DOMAIN_SERVER_ROLE_INFORMATION,
 *PDOMAIN_SERVER_ROLE_INFORMATION;
```

For information on each field, see section 2.2.4.1.

2.2.4.8 DOMAIN_MODIFIED_INFORMATION

The **DOMAIN_MODIFIED_INFORMATION** structure contains domain fields.

```
typedef struct _DOMAIN_MODIFIED_INFORMATION {
  OLD_LARGE_INTEGER DomainModifiedCount;
  OLD_LARGE_INTEGER CreationTime;
} DOMAIN_MODIFIED_INFORMATION,
 *PDOMAIN_MODIFIED_INFORMATION;
```

For information on each field, see section 2.2.4.1.

2.2.4.9 DOMAIN_MODIFIED_INFORMATION2

The **DOMAIN_MODIFIED_INFORMATION2** structure contains domain fields.

```
typedef struct _DOMAIN_MODIFIED_INFORMATION2 {
  OLD_LARGE_INTEGER DomainModifiedCount;
  OLD_LARGE_INTEGER CreationTime;
  OLD_LARGE_INTEGER ModifiedCountAtLastPromotion;
} DOMAIN_MODIFIED_INFORMATION2,
 *PDOMAIN_MODIFIED_INFORMATION2;
```

For information on each field, see section 2.2.4.1.

2.2.4.10 SAMPR_DOMAIN_GENERAL_INFORMATION

The **SAMPR_DOMAIN_GENERAL_INFORMATION** structure contains domain fields.

```
typedef struct _SAMPR_DOMAIN_GENERAL_INFORMATION {
  OLD_LARGE_INTEGER ForceLogoff;
  RPC_UNICODE_STRING OemInformation;
  RPC_UNICODE_STRING DomainName;
  RPC_UNICODE_STRING ReplicaSourceNodeName;
  OLD_LARGE_INTEGER DomainModifiedCount;
  DOMAIN_SERVER_ENABLE_STATE DomainServerState;
  DOMAIN_SERVER_ROLE DomainServerRole;
  unsigned char UasCompatibilityRequired;
  unsigned long UserCount;
  unsigned long GroupCount;
  unsigned long AliasCount;
} SAMPR_DOMAIN_GENERAL_INFORMATION,
 *PSAMPR_DOMAIN_GENERAL_INFORMATION;
```

For information on each field, see section 2.2.4.1.

2.2.4.11 SAMPR_DOMAIN_GENERAL_INFORMATION2

The **SAMPR_DOMAIN_GENERAL_INFORMATION2** structure contains domain fields.

```
typedef struct _SAMPR_DOMAIN_GENERAL_INFORMATION2 {
  SAMPR_DOMAIN_GENERAL_INFORMATION I1;
  OLD_LARGE_INTEGER LockoutDuration;
  OLD_LARGE_INTEGER LockoutObservationWindow;
  unsigned short LockoutThreshold;
} SAMPR_DOMAIN_GENERAL_INFORMATION2,
 *PSAMPR_DOMAIN_GENERAL_INFORMATION2;
```

For information on each field, see section 2.2.4.1, except for **I1**, which is specified in section 2.2.4.10.

2.2.4.12 SAMPR_DOMAIN_OEM_INFORMATION

The **SAMPR_DOMAIN_OEM_INFORMATION** structure contains domain fields.

```
typedef struct _SAMPR_DOMAIN_OEM_INFORMATION {
  RPC_UNICODE_STRING OemInformation;
} SAMPR_DOMAIN_OEM_INFORMATION,
 *PSAMPR_DOMAIN_OEM_INFORMATION;
```

For information on each field, see section 2.2.4.1.

2.2.4.13 SAMPR_DOMAIN_NAME_INFORMATION

The **SAMPR_DOMAIN_NAME_INFORMATION** structure contains domain fields.

```
typedef struct _SAMPR_DOMAIN_NAME_INFORMATION {
  RPC_UNICODE_STRING DomainName;
} SAMPR_DOMAIN_NAME_INFORMATION,
 *PSAMPR_DOMAIN_NAME_INFORMATION;
```

For information on each field, see section 2.2.4.1.

2.2.4.14 SAMPR_DOMAIN_REPLICATION_INFORMATION

The **SAMPR_DOMAIN_REPLICATION_INFORMATION** structure contains domain fields.

```
typedef struct SAMPR_DOMAIN_REPLICATION_INFORMATION {
  RPC_UNICODE_STRING ReplicaSourceNodeName;
} SAMPR_DOMAIN_REPLICATION_INFORMATION,
 *PSAMPR_DOMAIN_REPLICATION_INFORMATION;
```

For information on each field, see section 2.2.4.1.

2.2.4.15 SAMPR_DOMAIN_LOCKOUT_INFORMATION

The **SAMPR_DOMAIN_LOCKOUT_INFORMATION** structure contains domain fields.

```
typedef struct _SAMPR_DOMAIN_LOCKOUT_INFORMATION {
  OLD_LARGE_INTEGER LockoutDuration;
  OLD_LARGE_INTEGER LockoutObservationWindow;
  unsigned short LockoutThreshold;
} SAMPR_DOMAIN_LOCKOUT_INFORMATION,
 *PSAMPR_DOMAIN_LOCKOUT_INFORMATION;
```

For information on each field, see section 2.2.4.1.

2.2.4.16 DOMAIN_INFORMATION_CLASS

The **DOMAIN_INFORMATION_CLASS** enumeration indicates how to interpret the Buffer parameter for **SamrSetInformationDomain** and **SamrQueryInformationDomain**. For a list of associated structures, see section 2.2.4.17.

```
typedef  enum _DOMAIN_INFORMATION_CLASS
{
  DomainPasswordInformation = 1,
  DomainGeneralInformation = 2,
  DomainLogoffInformation = 3,
  DomainOemInformation = 4,
  DomainNameInformation = 5,
  DomainReplicationInformation = 6,
  DomainServerRoleInformation = 7,
  DomainModifiedInformation = 8,
  DomainStateInformation = 9,
  DomainUasInformation = 10,
  DomainGeneralInformation2 = 11,
  DomainLockoutInformation = 12,
  DomainModifiedInformation2 = 13
} DOMAIN_INFORMATION_CLASS;
```

2.2.4.17 SAMPR_DOMAIN_INFO_BUFFER

The **SAMPR_DOMAIN_INFO_BUFFER** union combines all possible structures used in the **SamrSetInformationDomain** and **SamrQueryInformationDomain** methods. For details on each field, see the associated section for each field structure.

```
typedef
[switch_type(DOMAIN_INFORMATION_CLASS)]
union _SAMPR_DOMAIN_INFO_BUFFER {
  [case(DomainPasswordInformation)]
    DOMAIN_PASSWORD_INFORMATION Password;
  [case(DomainGeneralInformation)]
    SAMPR_DOMAIN_GENERAL_INFORMATION General;
  [case(DomainLogoffInformation)]
    DOMAIN_LOGOFF_INFORMATION Logoff;
  [case(DomainOemInformation)]
    SAMPR_DOMAIN_OEM_INFORMATION Oem;
  [case(DomainNameInformation)]
    SAMPR_DOMAIN_NAME_INFORMATION Name;
  [case(DomainServerRoleInformation)]
    DOMAIN_SERVER_ROLE_INFORMATION Role;
  [case(DomainReplicationInformation)]
    SAMPR_DOMAIN_REPLICATION_INFORMATION Replication;
  [case(DomainModifiedInformation)]
    DOMAIN_MODIFIED_INFORMATION Modified;
  [case(DomainStateInformation)]
    DOMAIN_STATE_INFORMATION State;
  [case(DomainGeneralInformation2)]
    SAMPR_DOMAIN_GENERAL_INFORMATION2 General2;
  [case(DomainLockoutInformation)]
    SAMPR_DOMAIN_LOCKOUT_INFORMATION Lockout;
  [case(DomainModifiedInformation2)]
    DOMAIN_MODIFIED_INFORMATION2 Modified2;
```

```
} SAMPR_DOMAIN_INFO_BUFFER,
 *PSAMPR_DOMAIN_INFO_BUFFER;
```

2.2.5 Group Query/Set Data Types

The structures and fields in this section relate to the following methods:

- **SamrQueryInformationGroup**
- **SamrSetInformationGroup**

The model of the methods is for the requester to specify an enumeration that indicates the attributes to be either set or queried. There is duplication among the structures that contain the attributes. For a description of each attribute that is common among structures, see section 2.2.5.1.

2.2.5.1 Common Group Fields

There are a number of group-related structures that use the same fields, as denoted by their field names. This section specifies all such fields.

The structures group the available set of group attributes in different ways to allow the requester to control which attributes are queried or set. While each structure may have a different subset of these attributes, they all draw from this same set of attributes, detailed below.

AdminComment: A counted Unicode string of type **RPC_UNICODE_STRING**, indicating the description of the group object.

Attributes: A 32-bit bitfield containing characteristics about a group; for possible values, see section 2.2.1.10.

MemberCount: A 32-bit unsigned integer indicating the number of members in the group object. This field is read-only.

Name: A counted Unicode string of type **RPC_UNICODE_STRING**, indicating the name of the group object.

2.2.5.2 GROUP_ATTRIBUTE_INFORMATION

The **GROUP_ATTRIBUTE_INFORMATION** structure contains group fields.

```
typedef struct _GROUP_ATTRIBUTE_INFORMATION {
  unsigned long Attributes;
} GROUP_ATTRIBUTE_INFORMATION,
 *PGROUP_ATTRIBUTE_INFORMATION;
```

For information on each field, see section 2.2.5.1.

2.2.5.3 SAMPR_GROUP_GENERAL_INFORMATION

The **SAMPR_GROUP_GENERAL_INFORMATION** structure contains group fields.

```
typedef struct _SAMPR_GROUP_GENERAL_INFORMATION {
  RPC_UNICODE_STRING Name;
  unsigned long Attributes;
  unsigned long MemberCount;
  RPC_UNICODE_STRING AdminComment;
} SAMPR_GROUP_GENERAL_INFORMATION,
 *PSAMPR_GROUP_GENERAL_INFORMATION;
```

For information on each field, see section 2.2.5.1.

2.2.5.4 SAMPR_GROUP_NAME_INFORMATION

The **SAMPR_GROUP_NAME_INFORMATION** structure contains group fields.

```
typedef struct _SAMPR_GROUP_NAME_INFORMATION {
  RPC_UNICODE_STRING Name;
} SAMPR_GROUP_NAME_INFORMATION,
 *PSAMPR_GROUP_NAME_INFORMATION;
```

For information on each field, see section 2.2.5.1.

2.2.5.5 SAMPR_GROUP_ADM_COMMENT_INFORMATION

The **SAMPR_GROUP_ADM_COMMENT_INFORMATION** structure contains group fields.

```
typedef struct _SAMPR_GROUP_ADM_COMMENT_INFORMATION {
  RPC_UNICODE_STRING AdminComment;
} SAMPR_GROUP_ADM_COMMENT_INFORMATION,
 *PSAMPR_GROUP_ADM_COMMENT_INFORMATION;
```

For information on each field, see section 2.2.5.1.

2.2.5.6 GROUP_INFORMATION_CLASS

The **GROUP_INFORMATION_CLASS** enumeration indicates how to interpret the *Buffer* parameter for **SamrSetInformationGroup** and **SamrQueryInformationGroup**. For a list of associated structures, see section 2.2.5.7.

```
typedef  enum _GROUP_INFORMATION_CLASS
{
  GroupGeneralInformation = 1,
  GroupNameInformation,
  GroupAttributeInformation,
  GroupAdminCommentInformation,
  GroupReplicationInformation
} GROUP_INFORMATION_CLASS;
```

2.2.5.7 SAMPR_GROUP_INFO_BUFFER

The **SAMPR_GROUP_INFO_BUFFER** union combines all possible structures used in the **SamrSetInformationGroup** and **SamrQueryInformationGroup** methods. For information on each field, with the exception of the **DoNotUse** field, see the associated section for the field structure.

```
typedef
[switch_type(GROUP_INFORMATION_CLASS)]
union _SAMPR_GROUP_INFO_BUFFER {
  [case(GroupGeneralInformation)]
    SAMPR_GROUP_GENERAL_INFORMATION General;
  [case(GroupNameInformation)]
    SAMPR_GROUP_NAME_INFORMATION Name;
  [case(GroupAttributeInformation)]
    GROUP_ATTRIBUTE_INFORMATION Attribute;
  [case(GroupAdminCommentInformation)]
    SAMPR_GROUP_ADM_COMMENT_INFORMATION AdminComment;
  [case(GroupReplicationInformation)]
    SAMPR_GROUP_GENERAL_INFORMATION DoNotUse;
} SAMPR_GROUP_INFO_BUFFER,
 *PSAMPR_GROUP_INFO_BUFFER;
```

DoNotUse: This field exists to allow the GroupReplicationInformation enumeration to be specified by the requester.

As specified in section 3.2.5.5.3.1, the **General** field (instead of **DoNotUse**) MUST be used by the responder when GroupReplicationInformation is received. GroupReplicationInformation is not valid for a set operation.

2.2.6 Alias Query/Set Data Types

The structures and fields in this section relate to the following methods:

- **SamrQueryInformationAlias**
- **SamrSetInformationAlias**

The model of the methods is for the requester to specify an enumeration that indicates the attributes to be either set or queried. There is duplication among the structures that contain the attributes. For a description of each attribute that is common among structures, see section 2.2.6.1.

2.2.6.1 Common Alias Fields

There are a number of alias-related structures that use the same fields, as denoted by their field names. This section specifies all such fields.

The structures group the available set of alias attributes in different ways to allow the requester to control which attributes are queried or set. While each structure may have a different subset of these attributes, they all draw from this same set of attributes, detailed below.

AdminComment: A counted Unicode string of type **RPC_UNICODE_STRING**, indicating the description of the alias object.

MemberCount: A 32-bit unsigned integer indicating the number of members in the alias object. This field is read-only.

Name: A counted Unicode string of type **RPC_UNICODE_STRING**, indicating the name of the alias object.

2.2.6.2 SAMPR_ALIAS_GENERAL_INFORMATION

The **SAMPR_ALIAS_GENERAL_INFORMATION** structure contains alias fields.

```
typedef struct _SAMPR_ALIAS_GENERAL_INFORMATION {
  RPC_UNICODE_STRING Name;
  unsigned long MemberCount;
  RPC_UNICODE_STRING AdminComment;
} SAMPR_ALIAS_GENERAL_INFORMATION,
 *PSAMPR_ALIAS_GENERAL_INFORMATION;
```

For information on each field, see section 2.2.6.1.

2.2.6.3 SAMPR_ALIAS_NAME_INFORMATION

The **SAMPR_ALIAS_NAME_INFORMATION** structure contains alias fields.

```
typedef struct _SAMPR_ALIAS_NAME_INFORMATION {
  RPC_UNICODE_STRING Name;
} SAMPR_ALIAS_NAME_INFORMATION,
 *PSAMPR_ALIAS_NAME_INFORMATION;
```

For information on each field, see section 2.2.6.1.

2.2.6.4 SAMPR_ALIAS_ADM_COMMENT_INFORMATION

The **SAMPR_ALIAS_ADM_COMMENT_INFORMATION** structure contains alias fields.

```
typedef struct _SAMPR_ALIAS_ADM_COMMENT_INFORMATION {
  RPC_UNICODE_STRING AdminComment;
} SAMPR_ALIAS_ADM_COMMENT_INFORMATION,
 *PSAMPR_ALIAS_ADM_COMMENT_INFORMATION;
```

For information on each field, see section 2.2.6.1.

2.2.6.5 ALIAS_INFORMATION_CLASS

The **ALIAS_INFORMATION_CLASS** enumeration indicates how to interpret the *Buffer* parameter for **SamrQueryInformationAlias** and **SamrSetInformationAlias**. For a list of the structures associated with each enumeration, see section 2.2.6.6.

```
typedef  enum _ALIAS_INFORMATION_CLASS
{
```

```
  AliasGeneralInformation = 1,
  AliasNameInformation,
  AliasAdminCommentInformation
} ALIAS_INFORMATION_CLASS;
```

2.2.6.6 SAMPR_ALIAS_INFO_BUFFER

The **SAMPR_ALIAS_INFO_BUFFER** union combines all possible structures used in the **SamrSetInformationAlias** and **SamrQueryInformationAlias** methods. For information on each field, see the associated section for the field structure.

```
typedef
[switch_type(ALIAS_INFORMATION_CLASS)]
union _SAMPR_ALIAS_INFO_BUFFER {
  [case(AliasGeneralInformation)]
    SAMPR_ALIAS_GENERAL_INFORMATION General;
  [case(AliasNameInformation)]
    SAMPR_ALIAS_NAME_INFORMATION Name;
  [case(AliasAdminCommentInformation)]
    SAMPR_ALIAS_ADM_COMMENT_INFORMATION AdminComment;
} SAMPR_ALIAS_INFO_BUFFER,
 *PSAMPR_ALIAS_INFO_BUFFER;
```

2.2.7 User Query/Set Data Types

The structures and fields in this section relate to the following methods:

- **SamrQueryInformationUser**
- **SamrQueryInformationUser2**
- **SamrSetInformationUser**
- **SamrSetInformationUser2**

The model of the methods is for the requester to specify an enumeration that indicates the attributes to be either set or queried. There is duplication among the structures that contain the attributes. For a description of each attribute that is common among structures, see section 2.2.7.1.

2.2.7.1 Common User Fields

There are a number of user-related structures that use the same fields, as denoted by their field names. This section specifies all such fields.

These structures group the available set of user attributes in different ways to allow the requester greater control over which attributes are queried or set. While each structure may have a different subset of these attributes, they all draw from this same set of attributes, detailed below.

There are a number of fields that are of type "user profile information" (as indicated in their descriptions). The responder does not enforce any format restrictions on these values during an update. These values are used by authentication protocols—Kerberos, for example, as specified in [MS-PAC] section **2.5**—to communicate end-user environment values to an interactive-logon application running on a member workstation or server. For clarity, Windows behavior is cited in this

section to describe the expectations of such Windows interactive-logon applications with respect to these values. If no Windows behavior is cited, there is no expectation of a specific format.

AccountExpires: A 64-bit value, equivalent to a **FILETIME**, indicating the time at which an account MUST no longer be permitted to log on.

AdminComment: A counted Unicode string of type **RPC_UNICODE_STRING**, indicating the description of the user object.

BadPasswordCount: A 16-bit unsigned integer indicating the number of bad password attempts. This field is read-only.

CodePage: A 16-bit unsigned integer indicating a code page preference specific to this user object. The space of values is the Microsoft code page designation. For more information, see [MSDN-CP].

CountryCode: A 16-bit unsigned integer indicating a country preference specific to this user. The space of values is the international country calling code, as specified in [E164]. For example, the country code of the United Kingdom, in decimal, is 44.

FullName: A counted Unicode string of type **RPC_UNICODE_STRING**, indicating a free format string for any name type (for example, "Akers, Kim").

HomeDirectory: A counted Unicode string of type **RPC_UNICODE_STRING**, indicating a directory in which an end-user interactive-logon application SHOULD start. This is **user profile** information.<16>

HomeDirectoryDrive: A counted Unicode string of type **RPC_UNICODE_STRING**, indicating the disk drive to which HomeDirectory is relative. This is user profile information.<17>

LastLogoff: A 64-bit value, equivalent to a **FILETIME**, indicating the time at which the account last logged off. This field is read-only.<18>

LastLogon: A 64-bit value, equivalent to a **FILETIME**, indicating the time at which the account last logged on. This field is read-only.<19>

LogonCount: A 16-bit unsigned integer indicating the number times the user account has been authenticated. This field is read-only.<20>

LogonHours: A binary value with the structure **SAMPR_LOGON_HOURS**, indicating a logon policy describing the time periods during which the user can authenticate. This policy is specified in detail in section 2.2.7.5.

Parameters: A binary value stored in the **Buffer** field of a **RPC_UNICODE_STRING** for per-user application state. Per-user application state is any binary data that an application associates with a user. However, because there is no requirement for the responder of this protocol to enforce any format, application developers are discouraged from using this mechanism in order to avoid the chance of one application overwriting another application's data.

PasswordCanChange: A 64-bit value, equivalent to a **FILETIME**, indicating the time at which a password change request will be accepted by the responder. This field is read-only.

PasswordExpired: A 1-byte value that, if nonzero, indicates that the password MUST be expired immediately on receipt at the responder (see **SamrSetInformationUser2 (section 3.2.5.6.4)** for details). On receipt at the requester, this field indicates, if nonzero, that the password has expired; if the value is zero, the password has not expired.

PasswordLastSet: A 64-bit value, equivalent to a **FILETIME**, indicating the time at which a password was last updated. This field is read-only.

PasswordMustChange: A 64-bit value, equivalent to a **FILETIME**, indicating the time at which authentications will fail unless a password reset or change occurs. This field is read-only.

PrimaryGroupId: A 32-bit unsigned integer indicating the primary group ID of the user.

ProfilePath: A counted Unicode string of type **RPC_UNICODE_STRING**, containing a UNC path to a network-based user profile. This is user profile information.

ScriptPath: A counted Unicode string of type **RPC_UNICODE_STRING**, containing a UNC path to a network-based script or executable that is executed during an interactive logon. This is user profile information.

UserAccountControl: A 32-bit bitfield specifying characteristics of the account. See section 2.2.1.12 for possible values.

UserComment: A counted Unicode string of type **RPC_UNICODE_STRING** containing an end-user–writable comment about the user. This is distinguished from **AdminComment** by the fact that, by default, an end user can update this value on their own account.

UserId: A 32-bit unsigned integer representing the RID of the account. This field is read-only.

UserName: A counted Unicode string of type **RPC_UNICODE_STRING** containing the name of the account.

WorkStations: A binary value stored in an **RPC_UNICODE_STRING** structure containing the list of workstations from which the account can interactively log on. For information on the required format of the binary value, see section 3.2.1.6.

2.2.7.2 USER_PRIMARY_GROUP_INFORMATION

The **USER_PRIMARY_GROUP_INFORMATION** structure contains user fields.

```
typedef struct _USER_PRIMARY_GROUP_INFORMATION {
  unsigned long PrimaryGroupId;
} USER_PRIMARY_GROUP_INFORMATION,
 *PUSER_PRIMARY_GROUP_INFORMATION;
```

For information on each field, see section 2.2.7.1.

2.2.7.3 USER_CONTROL_INFORMATION

The **USER_CONTROL_INFORMATION** structure contains user fields.

```
typedef struct _USER_CONTROL_INFORMATION {
  unsigned long UserAccountControl;
} USER_CONTROL_INFORMATION,
 *PUSER_CONTROL_INFORMATION;
```

For information on each field, see section 2.2.7.1.

2.2.7.4 USER_EXPIRES_INFORMATION

The **USER_EXPIRES_INFORMATION** structure contains user fields.

```
typedef struct _USER_EXPIRES_INFORMATION {
  OLD_LARGE_INTEGER AccountExpires;
} USER_EXPIRES_INFORMATION,
 *PUSER_EXPIRES_INFORMATION;
```

For information on each field, see section 2.2.7.1.

2.2.7.5 SAMPR_LOGON_HOURS

The **SAMPR_LOGON_HOURS** structure contains logon policy information that describes when a user account is permitted to authenticate.

```
typedef struct _SAMPR_LOGON_HOURS {
  unsigned short UnitsPerWeek;
  [size_is(1260), length_is((UnitsPerWeek+7)/8)]
    unsigned char* LogonHours;
} SAMPR_LOGON_HOURS,
 *PSAMPR_LOGON_HOURS;
```

UnitsPerWeek: A division of the week (7 days). For example, the value 7 means that each unit is a day; a value of (7*24) means that the units are hours. The minimum granularity of time is one minute, where the UnitsPerWeek would be 10080; therefore, the maximum size of LogonHours is 10080/8, or 1,260 bytes.

LogonHours: A pointer to a bitfield containing at least **UnitsPerWeek** number of bits. The leftmost bit represents the first unit, starting at Sunday 12 a.m. If a bit is set, authentication is allowed to occur; otherwise, authentication is not allowed to occur.

For example, if the **UnitsPerWeek** value is 168 (that is, the units per week is hours, resulting in a 21-byte bitfield), and the leftmost bit is set, and the rightmost bit is set, the user is able to log on for two consecutive hours between Saturday 11 p.m. and Sunday 1 a.m.

2.2.7.6 SAMPR_USER_ALL_INFORMATION

The **SAMPR_USER_ALL_INFORMATION** structure contains user attribute information. Most fields are described in section 2.2.7.1. The exceptions are described below.

```
typedef struct _SAMPR_USER_ALL_INFORMATION {
  OLD_LARGE_INTEGER LastLogon;
  OLD_LARGE_INTEGER LastLogoff;
  OLD_LARGE_INTEGER PasswordLastSet;
  OLD_LARGE_INTEGER AccountExpires;
  OLD_LARGE_INTEGER PasswordCanChange;
  OLD_LARGE_INTEGER PasswordMustChange;
  RPC_UNICODE_STRING UserName;
  RPC_UNICODE_STRING FullName;
  RPC_UNICODE_STRING HomeDirectory;
  RPC_UNICODE_STRING HomeDirectoryDrive;
```

```
  RPC_UNICODE_STRING ScriptPath;
  RPC_UNICODE_STRING ProfilePath;
  RPC_UNICODE_STRING AdminComment;
  RPC_UNICODE_STRING WorkStations;
  RPC_UNICODE_STRING UserComment;
  RPC_UNICODE_STRING Parameters;
  RPC_UNICODE_STRING LmOwfPassword;
  RPC_UNICODE_STRING NtOwfPassword;
  RPC_UNICODE_STRING PrivateData;
  SAMPR_SR_SECURITY_DESCRIPTOR SecurityDescriptor;
  unsigned long UserId;
  unsigned long PrimaryGroupId;
  unsigned long UserAccountControl;
  unsigned long WhichFields;
  SAMPR_LOGON_HOURS LogonHours;
  unsigned short BadPasswordCount;
  unsigned short LogonCount;
  unsigned short CountryCode;
  unsigned short CodePage;
  unsigned char LmPasswordPresent;
  unsigned char NtPasswordPresent;
  unsigned char PasswordExpired;
  unsigned char PrivateDataSensitive;
} SAMPR_USER_ALL_INFORMATION,
 *PSAMPR_USER_ALL_INFORMATION;
```

LmOwfPassword: An **RPC_UNICODE_STRING** structure where the Length and MaximumLength MUST be 16 bytes, and the Buffer MUST be formatted with an **ENCRYPTED_LM_OWF_PASSWORD** structure with the cleartext value being an LM hash, and the encryption key being the 16-byte SMB [MS-SMB] session key established by the underlying authentication protocol (either Kerberos [MS-KILE] or NTLM [MS-NLMP]).

NtOwfPassword: An **RPC_UNICODE_STRING** structure where the Length and MaximumLength MUST be 16 bytes, and the Buffer MUST be formatted with an **ENCRYPTED_NT_OWF_PASSWORD** structure with the cleartext value being an NT hash, and the encryption key being the 16-byte SMB [MS-SMB] session key established by the underlying authentication protocol (either Kerberos [MS-KILE] or NTLM [MS-NLMP]).

PrivateData: Not used. Ignored on receipt at the responder and requester. Requesters MUST set to zero on send and responders MUST set to zero on return.

SecurityDescriptor: Not used. Ignored on receipt at the responder and requester. Requesters MUST set to zero on send and responders MUST set to zero on return.

WhichFields: A 32-bit bitfield indicating which fields within the **SAMPR_USER_ALL_INFORMATION** structure MUST be processed, and which fields MUST be ignored by the responder on receipt. See section 2.2.1.8 for details on the valid bits of WhichFields and, if a given bit is set, which field MUST be processed (if a given bit is not set, then the associated field MUST be ignored).

LmPasswordPresent: If zero, LmOwfPassword MUST be ignored; otherwise, LmOwfPassword MUST be processed.

NtPasswordPresent: If zero, NtOwfPassword MUST be ignored; otherwise, NtOwfPassword MUST be processed.

PrivateDataSensitive: Not used. Ignored on receipt at the responder and requester.

2.2.7.7 SAMPR_USER_GENERAL_INFORMATION

The **SAMPR_USER_GENERAL_INFORMATION** structure contains user fields.

```
typedef struct _SAMPR_USER_GENERAL_INFORMATION {
  RPC_UNICODE_STRING UserName;
  RPC_UNICODE_STRING FullName;
  unsigned long PrimaryGroupId;
  RPC_UNICODE_STRING AdminComment;
  RPC_UNICODE_STRING UserComment;
} SAMPR_USER_GENERAL_INFORMATION,
 *PSAMPR_USER_GENERAL_INFORMATION;
```

For information on each field, see section 2.2.7.1.

2.2.7.8 SAMPR_USER_PREFERENCES_INFORMATION

The **SAMPR_USER_PREFERENCES_INFORMATION** structure contains user fields.

```
typedef struct _SAMPR_USER_PREFERENCES_INFORMATION {
  RPC_UNICODE_STRING UserComment;
  RPC_UNICODE_STRING Reserved1;
  unsigned short CountryCode;
  unsigned short CodePage;
} SAMPR_USER_PREFERENCES_INFORMATION,
 *PSAMPR_USER_PREFERENCES_INFORMATION;
```

Reserved1: Ignored by the requester and responder and MUST be a zero-length string on send and return.

For information on all other fields, see section 2.2.7.1.

2.2.7.9 SAMPR_USER_PARAMETERS_INFORMATION

The **SAMPR_USER_PARAMETERS_INFORMATION** structure contains user fields.

```
typedef struct _SAMPR_USER_PARAMETERS_INFORMATION {
  RPC_UNICODE_STRING Parameters;
} SAMPR_USER_PARAMETERS_INFORMATION,
 *PSAMPR_USER_PARAMETERS_INFORMATION;
```

For information on each field, see section 2.2.7.1.

2.2.7.10 SAMPR_USER_LOGON_INFORMATION

The **SAMPR_USER_LOGON_INFORMATION** structure contains user fields.

```
typedef struct _SAMPR_USER_LOGON_INFORMATION {
  RPC_UNICODE_STRING UserName;
  RPC_UNICODE_STRING FullName;
  unsigned long UserId;
  unsigned long PrimaryGroupId;
  RPC_UNICODE_STRING HomeDirectory;
  RPC_UNICODE_STRING HomeDirectoryDrive;
  RPC_UNICODE_STRING ScriptPath;
  RPC_UNICODE_STRING ProfilePath;
  RPC_UNICODE_STRING WorkStations;
  OLD_LARGE_INTEGER LastLogon;
  OLD_LARGE_INTEGER LastLogoff;
  OLD_LARGE_INTEGER PasswordLastSet;
  OLD_LARGE_INTEGER PasswordCanChange;
  OLD_LARGE_INTEGER PasswordMustChange;
  SAMPR_LOGON_HOURS LogonHours;
  unsigned short BadPasswordCount;
  unsigned short LogonCount;
  unsigned long UserAccountControl;
} SAMPR_USER_LOGON_INFORMATION,
 *PSAMPR_USER_LOGON_INFORMATION;
```

For information on each field, see section 2.2.7.1.

2.2.7.11 SAMPR_USER_ACCOUNT_INFORMATION

The **SAMPR_USER_ACCOUNT_INFORMATION** structure contains user fields.

```
typedef struct _SAMPR_USER_ACCOUNT_INFORMATION {
  RPC_UNICODE_STRING UserName;
  RPC_UNICODE_STRING FullName;
  unsigned long UserId;
  unsigned long PrimaryGroupId;
  RPC_UNICODE_STRING HomeDirectory;
  RPC_UNICODE_STRING HomeDirectoryDrive;
  RPC_UNICODE_STRING ScriptPath;
  RPC_UNICODE_STRING ProfilePath;
  RPC_UNICODE_STRING AdminComment;
  RPC_UNICODE_STRING WorkStations;
  OLD_LARGE_INTEGER LastLogon;
  OLD_LARGE_INTEGER LastLogoff;
  SAMPR_LOGON_HOURS LogonHours;
  unsigned short BadPasswordCount;
  unsigned short LogonCount;
  OLD_LARGE_INTEGER PasswordLastSet;
  OLD_LARGE_INTEGER AccountExpires;
  unsigned long UserAccountControl;
} SAMPR_USER_ACCOUNT_INFORMATION,
 *PSAMPR_USER_ACCOUNT_INFORMATION;
```

For information on each field, see section 2.2.7.1.

2.2.7.12 SAMPR_USER_A_NAME_INFORMATION

The **SAMPR_USER_A_NAME_INFORMATION** structure contains user fields.

```
typedef struct _SAMPR_USER_A_NAME_INFORMATION {
  RPC_UNICODE_STRING UserName;
} SAMPR_USER_A_NAME_INFORMATION,
 *PSAMPR_USER_A_NAME_INFORMATION;
```

For information on each field, see section 2.2.7.1.

2.2.7.13 SAMPR_USER_F_NAME_INFORMATION

The **SAMPR_USER_F_NAME_INFORMATION** structure contains user fields.

```
typedef struct _SAMPR_USER_F_NAME_INFORMATION {
  RPC_UNICODE_STRING FullName;
} SAMPR_USER_F_NAME_INFORMATION,
 *PSAMPR_USER_F_NAME_INFORMATION;
```

For information on each field, see section 2.2.7.1.

2.2.7.14 SAMPR_USER_NAME_INFORMATION

The **SAMPR_USER_NAME_INFORMATION** structure contains user fields.

```
typedef struct _SAMPR_USER_NAME_INFORMATION {
  RPC_UNICODE_STRING UserName;
  RPC_UNICODE_STRING FullName;
} SAMPR_USER_NAME_INFORMATION,
 *PSAMPR_USER_NAME_INFORMATION;
```

For information on each field, see section 2.2.7.1.

2.2.7.15 SAMPR_USER_HOME_INFORMATION

The **SAMPR_USER_HOME_INFORMATION** structure contains user fields.

```
typedef struct _SAMPR_USER_HOME_INFORMATION {
  RPC_UNICODE_STRING HomeDirectory;
  RPC_UNICODE_STRING HomeDirectoryDrive;
} SAMPR_USER_HOME_INFORMATION,
 *PSAMPR_USER_HOME_INFORMATION;
```

For information on each field, see section 2.2.7.1.

2.2.7.16 SAMPR_USER_SCRIPT_INFORMATION

The **SAMPR_USER_SCRIPT_INFORMATION** structure contains user fields.

```
typedef struct _SAMPR_USER_SCRIPT_INFORMATION {
  RPC_UNICODE_STRING ScriptPath;
} SAMPR_USER_SCRIPT_INFORMATION,
 *PSAMPR_USER_SCRIPT_INFORMATION;
```

For information on each field, see section 2.2.7.1.

2.2.7.17 SAMPR_USER_PROFILE_INFORMATION

The **SAMPR_USER_PROFILE_INFORMATION** structure contains user fields.

```
typedef struct _SAMPR_USER_PROFILE_INFORMATION {
  RPC_UNICODE_STRING ProfilePath;
} SAMPR_USER_PROFILE_INFORMATION,
 *PSAMPR_USER_PROFILE_INFORMATION;
```

For information on each field, see section 2.2.7.1.

2.2.7.18 SAMPR_USER_ADMIN_COMMENT_INFORMATION

The **SAMPR_USER_ADMIN_COMMENT_INFORMATION** structure contains user fields.

```
typedef struct _SAMPR_USER_ADMIN_COMMENT_INFORMATION {
  RPC_UNICODE_STRING AdminComment;
} SAMPR_USER_ADMIN_COMMENT_INFORMATION,
 *PSAMPR_USER_ADMIN_COMMENT_INFORMATION;
```

For information on each field, see section 2.2.7.1.

2.2.7.19 SAMPR_USER_WORKSTATIONS_INFORMATION

The **SAMPR_USER_WORKSTATIONS_INFORMATION** structure contains user fields.

```
typedef struct _SAMPR_USER_WORKSTATIONS_INFORMATION {
  RPC_UNICODE_STRING WorkStations;
} SAMPR_USER_WORKSTATIONS_INFORMATION,
 *PSAMPR_USER_WORKSTATIONS_INFORMATION;
```

For information on each field, see section 2.2.7.1.

2.2.7.20 SAMPR_USER_LOGON_HOURS_INFORMATION

The **SAMPR_USER_LOGON_HOURS_INFORMATION** structure contains user fields.

```
typedef struct _SAMPR_USER_LOGON_HOURS_INFORMATION {
  SAMPR_LOGON_HOURS LogonHours;
} SAMPR_USER_LOGON_HOURS_INFORMATION,
 *PSAMPR_USER_LOGON_HOURS_INFORMATION;
```

For information on each field, see section 2.2.7.1.

2.2.7.21 SAMPR_ENCRYPTED_USER_PASSWORD

The **SAMPR_ENCRYPTED_USER_PASSWORD** structure carries an encrypted string.

```
typedef struct _SAMPR_ENCRYPTED_USER_PASSWORD {
  unsigned char Buffer[(256 * 2) + 4];
} SAMPR_ENCRYPTED_USER_PASSWORD,
 *PSAMPR_ENCRYPTED_USER_PASSWORD;
```

Buffer: An array to carry encrypted cleartext password data. The encryption key is method-specific, while the algorithm specified in section 3.1.2.1 is common for all methods that use this structure. See the message syntax for **SamrOemChangePasswordUser2 (section 3.2.5.10.2)** and **SamrUnicodeChangePasswordUser2 (section 3.2.5.10.3)**, and the message processing for **SamrSetInformationUser2 (section 3.2.5.6.4)**, for details on the encryption key selection. The size of (256 * 2) + 4 for **Buffer** is determined by the size of the structure that is encrypted, SAMPR_USER_PASSWORD; see below for more details.

For all protocol uses, the decrypted format of **Buffer** is the following structure:

```
typedef struct _SAMPR_USER_PASSWORD {
    wchar_t       Buffer[256];
    unsigned long Length;
} SAMPR_USER_PASSWORD, *PSAMPR_USER_PASSWORD;
```

Buffer: This array contains the cleartext value at the end of the buffer. The start of the string is **Length** number of bytes from the end of the buffer. The cleartext value can be no more than 512 bytes. The unused portions of SAMPR_USER_PASSWORD.Buffer SHOULD<21> be filled with random bytes by the requester. The value 512 is chosen because that is the longest password allowed by this protocol (and enforced by the responder).

Length: An unsigned integer, in little-endian byte order, that indicates the number of bytes of the cleartext value located in SAMPR_USER_PASSWORD.Buffer.

Implementations of this protocol MUST protect the **SAMPR_ENCRYPTED_USER_PASSWORD** structure by encrypting the 516 bytes of data referenced in its **Buffer** field on request (and reply), and decrypting on receipt. See section 3.1.2.1 for the specification of the algorithm performing encryption and decryption.

2.2.7.22 SAMPR_ENCRYPTED_USER_PASSWORD_NEW

The **SAMPR_ENCRYPTED_USER_PASSWORD_NEW** structure carries an encrypted string.

```
typedef struct _SAMPR_ENCRYPTED_USER_PASSWORD_NEW {
```

```
  unsigned char Buffer[(256 * 2) + 4 + 16];
} SAMPR_ENCRYPTED_USER_PASSWORD_NEW,
 *PSAMPR_ENCRYPTED_USER_PASSWORD_NEW;
```

Buffer: An array to carry encrypted cleartext password data.

For all protocol uses, the decrypted format of **Buffer** is the following structure:

```
typedef struct _SAMPR_USER_PASSWORD_NEW {
    WCHAR Buffer[256];
    ULONG Length;
    UCHAR ClearSalt[16];
} SAMPR_USER_PASSWORD_NEW, *PSAMPR_USER_PASSWORD_NEW;
```

Buffer: This array contains the cleartext value at the end of the buffer. The cleartext value can be no more than 512 bytes. The start of the string is **Length** number of bytes from the end of the buffer. The unused portions of SAMPR_USER_PASSWORD_NEW.Buffer SHOULD<22> be filled with random bytes by the requester.

Length: An unsigned integer, in little-endian byte order, that indicates the number of bytes of the cleartext value (located in SAMPR_USER_PASSWORD_NEW.Buffer).

ClearSalt: This value (a **salt**) MUST be filled with random bytes by the requester and MUST NOT be encrypted. The length of 16 was chosen in particular because 128 bits of randomness was deemed sufficiently secure when this protocol was introduced (circa 1998).

Implementations of this protocol MUST protect the **SAMPR_ENCRYPTED_USER_PASSWORD_NEW** structure by encrypting the first 516 bytes of data referenced in its **Buffer** field on request (and reply), and decrypting on receipt. See section 3.1.2.1 for the specification of the algorithm performing encryption and decryption.

The first 516 bytes are defined as the first 516 bytes of the SAMPR_USER_PASSWORD_NEW structure defined above. The last 16 bytes of the **SAMPR_ENCRYPTED_USER_PASSWORD_NEW** structure are defined as the last 16 bytes of the SAMPR_USER_PASSWORD_NEW structure defined above, and MUST NOT be encrypted or decrypted.

2.2.7.23 USER_PWD_CHANGE_FAILURE_INFORMATION

The **USER_PWD_CHANGE_FAILURE_INFORMATION** structure stores information regarding failure to change a user's password. This structure is reserved.<23>

```
typedef struct _USER_PWD_CHANGE_FAILURE_INFORMATION {
  unsigned long ExtendedFailureReason;
  RPC_UNICODE_STRING FilterModuleName;
} USER_PWD_CHANGE_FAILURE_INFORMATION,
 *PUSER_PWD_CHANGE_FAILURE_INFORMATION;
```

ExtendedFailureReason: A value that indicates the reason that the new password was not accepted. The following values are possible.

Value	Meaning
SAM_PWD_CHANGE_NO_ERROR 0	No error.
SAM_PWD_CHANGE_PASSWORD_TOO_SHORT 1	New password does not meet the password minimum length policy.
SAM_PWD_CHANGE_PWD_IN_HISTORY 2	New password is in the history.
SAM_PWD_CHANGE_USERNAME_IN_PASSWORD 3	Complexity check failed because the new password includes the user's account name.
SAM_PWD_CHANGE_FULLNAME_IN_PASSWORD 4	Complexity check failed because the new password includes the user's full name.
SAM_PWD_CHANGE_NOT_COMPLEX 5	Complexity check failed because the new password is not complex enough.
SAM_PWD_CHANGE_MACHINE_PASSWORD_NOT_DEFAULT 6	Only the default password is allowed. This restriction is set by the domain using the DOMAIN_REFUSE_PASSWORD_CHANGE flag in the **DOMAIN_PASSWORD_INFORMATION** structure.
SAM_PWD_CHANGE_FAILED_BY_FILTER 7	New password failed to pass through the filter. The FilterModuleName member provides the name of the filter dynamic-link library (DLL).
SAM_PWD_CHANGE_PASSWORD_TOO_LONG 8	New password does not meet the password maximum length policy.

FilterModuleName: Name of the filter DLL, returned as a **UNICODE_STRING**. This member is valid only if SAM_PWD_CHANGE_FAILED_BY_FILTER is returned in the **ExtendedFailureReason** member.

2.2.7.24 SAMPR_USER_INTERNAL1_INFORMATION

The **SAMPR_USER_INTERNAL1_INFORMATION** structure holds the hashed form of a cleartext password.

```
typedef struct _SAMPR_USER_INTERNAL1_INFORMATION {
  ENCRYPTED_NT_OWF_PASSWORD EncryptedNtOwfPassword;
  ENCRYPTED_LM_OWF_PASSWORD EncryptedLmOwfPassword;
  unsigned char NtPasswordPresent;
  unsigned char LmPasswordPresent;
  unsigned char PasswordExpired;
} SAMPR_USER_INTERNAL1_INFORMATION,
 *PSAMPR_USER_INTERNAL1_INFORMATION;
```

EncryptedNtOwfPassword: An NT hash encrypted with the 16-byte SMB [MS-SMB] session key for the connection established by the underlying authentication protocol (either Kerberos [MS-KILE] or NTLM [MS-NLMP]).

EncryptedLmOwfPassword: An LM hash encrypted with the 16-byte SMB [MS-SMB] session key for the connection established by the underlying authentication protocol (either Kerberos [MS-KILE] or NTLM [MS-NLMP]).

NtPasswordPresent: If nonzero, indicates that the **EncryptedNtOwfPassword** value is valid; otherwise, **EncryptedNtOwfPassword** MUST be ignored.

LmPasswordPresent: If nonzero, indicates that the **EncryptedLmOwfPassword** value is valid; otherwise, **EncryptedLmOwfPassword** MUST be ignored.

PasswordExpired: See section 2.2.7.1.

2.2.7.25 SAMPR_USER_INTERNAL4_INFORMATION

The **SAMPR_USER_INTERNAL4_INFORMATION** structure holds all attributes of a user, along with an encrypted password.

```
typedef struct _SAMPR_USER_INTERNAL4_INFORMATION {
  SAMPR_USER_ALL_INFORMATION I1;
  SAMPR_ENCRYPTED_USER_PASSWORD UserPassword;
} SAMPR_USER_INTERNAL4_INFORMATION,
 *PSAMPR_USER_INTERNAL4_INFORMATION;
```

I1: See section 2.2.7.6.

UserPassword: See section 2.2.7.21.

2.2.7.26 SAMPR_USER_INTERNAL4_INFORMATION_NEW

The **SAMPR_USER_INTERNAL4_INFORMATION_NEW** structure holds all attributes of a user, along with an encrypted password. The encrypted password uses a salt to improve the encryption algorithm. See the specification for **SAMPR_ENCRYPTED_USER_PASSWORD_NEW (section 2.2.7.22)** for details on salt value selection.

```
typedef struct _SAMPR_USER_INTERNAL4_INFORMATION_NEW {
  SAMPR_USER_ALL_INFORMATION I1;
  SAMPR_ENCRYPTED_USER_PASSWORD_NEW UserPassword;
} SAMPR_USER_INTERNAL4_INFORMATION_NEW,
 *PSAMPR_USER_INTERNAL4_INFORMATION_NEW;
```

I1: See section 2.2.7.6.

UserPassword: See section 2.2.7.22.

2.2.7.27 SAMPR_USER_INTERNAL5_INFORMATION

The **SAMPR_USER_INTERNAL5_INFORMATION** structure holds an encrypted password.

This structure is used to carry a new password for a particular account from the client to the server, encrypted in a way that protects it from disclosure or tampering while in transit. The encryption method that is used requires the client to encrypt the cleartext password and the server to decrypt the password, allowing the server to verify that the password meets implementation-specific policies such as complexity requirements and excluded words.

```
typedef struct _SAMPR_USER_INTERNAL5_INFORMATION {
  SAMPR_ENCRYPTED_USER_PASSWORD UserPassword;
  unsigned char PasswordExpired;
} SAMPR_USER_INTERNAL5_INFORMATION,
 *PSAMPR_USER_INTERNAL5_INFORMATION;
```

UserPassword: A cleartext password, encrypted per the specification for **SAMPR_ENCRYPTED_USER_PASSWORD**, with the encryption key being the 16-byte SMB [MS-SMB] session key established by the underlying authentication protocol (either Kerberos [MS-KILE] or NTLM [MS-NLMP]).

PasswordExpired: See section 2.2.7.1.

2.2.7.28 SAMPR_USER_INTERNAL5_INFORMATION_NEW

The **SAMPR_USER_INTERNAL5_INFORMATION_NEW** structure communicates an encrypted password. The encrypted password uses a salt to improve the encryption algorithm. See the specification for **SAMPR_ENCRYPTED_USER_PASSWORD_NEW (section 2.2.7.22)** for details on salt value selection.

This structure is used to carry a new password for a particular account from the client to the server, encrypted in a way that protects it from disclosure or tampering while in transit. A random value, a salt, is used by the client to seed the encryption routine; see section 2.2.7.22 for details. The encryption method that is used requires the client to encrypt the cleartext password and the server to decrypt the password, allowing the server to verify that the password meets implementation-specific policies such as complexity requirements and excluded words.

```
typedef struct _SAMPR_USER_INTERNAL5_INFORMATION_NEW {
  SAMPR_ENCRYPTED_USER_PASSWORD_NEW UserPassword;
  unsigned char PasswordExpired;
} SAMPR_USER_INTERNAL5_INFORMATION_NEW,
 *PSAMPR_USER_INTERNAL5_INFORMATION_NEW;
```

UserPassword: A password, encrypted per the specification for **SAMPR_ENCRYPTED_USER_PASSWORD_NEW**, with the encryption key being the 16-byte SMB [MS-SMB] session key established by the underlying authentication protocol (either Kerberos [MS-KILE] or NTLM [MS-NLMP]).

PasswordExpired: See section 2.2.7.1.

2.2.7.29 USER_INFORMATION_CLASS

The **USER_INFORMATION_CLASS** enumeration indicates how to interpret the *Buffer* parameter for **SamrSetInformationUser** and **SamrQueryInformationUser**. For a list of associated structures, see section 2.2.7.30.

```
typedef  enum _USER_INFORMATION_CLASS
{
  UserGeneralInformation = 1,
  UserPreferencesInformation = 2,
  UserLogonInformation = 3,
  UserLogonHoursInformation = 4,
  UserAccountInformation = 5,
  UserNameInformation = 6,
  UserAccountNameInformation = 7,
  UserFullNameInformation = 8,
  UserPrimaryGroupInformation = 9,
  UserHomeInformation = 10,
  UserScriptInformation = 11,
  UserProfileInformation = 12,
  UserAdminCommentInformation = 13,
  UserWorkStationsInformation = 14,
  UserControlInformation = 16,
  UserExpiresInformation = 17,
  UserInternal1Information = 18,
  UserParametersInformation = 20,
  UserAllInformation = 21,
  UserInternal4Information = 23,
  UserInternal5Information = 24,
  UserInternal4InformationNew = 25,
  UserInternal5InformationNew = 26
} USER_INFORMATION_CLASS,
 *PUSER_INFORMATION_CLASS;
```

2.2.7.30 SAMPR_USER_INFO_BUFFER

The **SAMPR_USER_INFO_BUFFER** union combines all possible structures used in the SamrSetInformationUser and SamrQueryInformationUser methods (see sections 3.2.5.6.5, 3.2.5.6.4, 3.2.5.5.6 and 3.2.5.5.5). For details on each field, see the associated section for the field structure.

```
typedef
[switch_type(USER_INFORMATION_CLASS)]
union _SAMPR_USER_INFO_BUFFER {
  [case(UserGeneralInformation)]
    SAMPR_USER_GENERAL_INFORMATION General;
  [case(UserPreferencesInformation)]
    SAMPR_USER_PREFERENCES_INFORMATION Preferences;
  [case(UserLogonInformation)]
    SAMPR_USER_LOGON_INFORMATION Logon;
  [case(UserLogonHoursInformation)]
    SAMPR_USER_LOGON_HOURS_INFORMATION LogonHours;
  [case(UserAccountInformation)]
    SAMPR_USER_ACCOUNT_INFORMATION Account;
  [case(UserNameInformation)]
    SAMPR_USER_NAME_INFORMATION Name;
  [case(UserAccountNameInformation)]
    SAMPR_USER_A_NAME_INFORMATION AccountName;
  [case(UserFullNameInformation)]
    SAMPR_USER_F_NAME_INFORMATION FullName;
  [case(UserPrimaryGroupInformation)]
    USER_PRIMARY_GROUP_INFORMATION PrimaryGroup;
```

```
    [case(UserHomeInformation)]
      SAMPR_USER_HOME_INFORMATION Home;
    [case(UserScriptInformation)]
      SAMPR_USER_SCRIPT_INFORMATION Script;
    [case(UserProfileInformation)]
      SAMPR_USER_PROFILE_INFORMATION Profile;
    [case(UserAdminCommentInformation)]
      SAMPR_USER_ADMIN_COMMENT_INFORMATION AdminComment;
    [case(UserWorkStationsInformation)]
      SAMPR_USER_WORKSTATIONS_INFORMATION WorkStations;
    [case(UserControlInformation)]
      USER_CONTROL_INFORMATION Control;
    [case(UserExpiresInformation)]
      USER_EXPIRES_INFORMATION Expires;
    [case(UserInternal1Information)]
      SAMPR_USER_INTERNAL1_INFORMATION Internal1;
    [case(UserParametersInformation)]
      SAMPR_USER_PARAMETERS_INFORMATION Parameters;
    [case(UserAllInformation)]
      SAMPR_USER_ALL_INFORMATION All;
    [case(UserInternal4Information)]
      SAMPR_USER_INTERNAL4_INFORMATION Internal4;
    [case(UserInternal5Information)]
      SAMPR_USER_INTERNAL5_INFORMATION Internal5;
    [case(UserInternal4InformationNew)]
      SAMPR_USER_INTERNAL4_INFORMATION_NEW Internal4New;
    [case(UserInternal5InformationNew)]
      SAMPR_USER_INTERNAL5_INFORMATION_NEW Internal5New;
  } SAMPR_USER_INFO_BUFFER,
   *PSAMPR_USER_INFO_BUFFER;
```

2.2.8 Selective Enumerate Associated Structures

The structures and fields in this section relate to the following methods:

- **SamrQueryDisplayInformation3**
- **SamrQueryDisplayInformation2**
- **SamrQueryDisplayInformation**
- **SamrGetDisplayEnumerationIndex2**
- **SamrGetDisplayEnumerationIndex**

The model of the methods is for the requester to specify an enumeration that indicates the attributes that are to be queried. There is duplication among the structures that contain the attributes. For a description of each attribute that is common among structures, see section 2.2.8.1.

2.2.8.1 Common Selective Enumerate Fields

There are a number of selective enumerate–related structures that use the same fields, as denoted by their field names. This section describes all such fields, and subsequent sections specify the fields in protocol structures. While each structure may have a different subset of these attributes, they all draw from this same set of attributes, detailed below.

When specified in a given structure, these fields all contain information about the same user or machine account, or group. The selective enumerate methods return an array of structures, thereby returning information about a set of users, machines, or groups.

AccountControl: A 32-bit bitfield representing the **UserAccountControl** field as described in section 2.2.7.1.

AccountName: A counted Unicode string of type **RPC_UNICODE_STRING**. When this field is used with a group object, it represents the **Name** field as described in section 2.2.5.1 (common group fields). Otherwise, this field represents the **UserName** field as described in section 2.2.7.1 (common user fields).

AdminComment: A counted Unicode string of type **RPC_UNICODE_STRING**. When this field is used with a group object, it represents the **AdminComment** field as described in section 2.2.5.1 (common group fields). Otherwise, this field represents the **AdminComment** field as described in section 2.2.7.1 (common user fields).

Attributes: A 32-bit bitfield representing the **Attributes** field, as described in section 2.2.5.1 (common group fields).

Index: A 32-bit unsigned integer; see the message processing of **SamrQueryDisplayInformation3 (section 3.2.5.3.1)** for details on the semantics of this field.

Rid: A 32-bit unsigned integer representing the RID of an account.

2.2.8.2 SAMPR_DOMAIN_DISPLAY_USER

The **SAMPR_DOMAIN_DISPLAY_USER** structure contains a subset of user account information sufficient to show a summary of the account for an account management application.

```
typedef struct _SAMPR_DOMAIN_DISPLAY_USER {
  unsigned long Index;
  unsigned long Rid;
  unsigned long AccountControl;
  RPC_UNICODE_STRING AccountName;
  RPC_UNICODE_STRING AdminComment;
  RPC_UNICODE_STRING FullName;
} SAMPR_DOMAIN_DISPLAY_USER,
 *PSAMPR_DOMAIN_DISPLAY_USER;
```

For information on each field, see section 2.2.8.1.

2.2.8.3 SAMPR_DOMAIN_DISPLAY_MACHINE

The **SAMPR_DOMAIN_DISPLAY_MACHINE** structure contains a subset of **machine account** information sufficient to show a summary of the account for an account management application.

```
typedef struct _SAMPR_DOMAIN_DISPLAY_MACHINE {
  unsigned long Index;
  unsigned long Rid;
  unsigned long AccountControl;
  RPC_UNICODE_STRING AccountName;
  RPC_UNICODE_STRING AdminComment;
} SAMPR_DOMAIN_DISPLAY_MACHINE,
```

```
 *PSAMPR_DOMAIN_DISPLAY_MACHINE;
```

For information on each field, see section 2.2.8.1.

2.2.8.4 SAMPR_DOMAIN_DISPLAY_GROUP

The **SAMPR_DOMAIN_DISPLAY_GROUP** structure contains a subset of group information sufficient to show a summary of the account for an account management application.

```
typedef struct _SAMPR_DOMAIN_DISPLAY_GROUP {
  unsigned long Index;
  unsigned long Rid;
  unsigned long Attributes;
  RPC_UNICODE_STRING AccountName;
  RPC_UNICODE_STRING AdminComment;
} SAMPR_DOMAIN_DISPLAY_GROUP,
 *PSAMPR_DOMAIN_DISPLAY_GROUP;
```

For information on each field, see section 2.2.8.1.

2.2.8.5 SAMPR_DOMAIN_DISPLAY_OEM_USER

The **SAMPR_DOMAIN_DISPLAY_OEM_USER** structure contains a subset of user account information sufficient to show a summary of the account for an account management application. This structure exists to support non–Unicode-based systems.

```
typedef struct _SAMPR_DOMAIN_DISPLAY_OEM_USER {
  unsigned long Index;
  RPC_STRING OemAccountName;
} SAMPR_DOMAIN_DISPLAY_OEM_USER,
 *PSAMPR_DOMAIN_DISPLAY_OEM_USER;
```

For information on each field, see section 2.2.8.1.

2.2.8.6 SAMPR_DOMAIN_DISPLAY_OEM_GROUP

The **SAMPR_DOMAIN_DISPLAY_OEM_GROUP** structure contains a subset of group information sufficient to show a summary of the account for an account management application. This structure exists to support non–Unicode-based systems.

```
typedef struct _SAMPR_DOMAIN_DISPLAY_OEM_GROUP {
  unsigned long Index;
  RPC_STRING OemAccountName;
} SAMPR_DOMAIN_DISPLAY_OEM_GROUP,
 *PSAMPR_DOMAIN_DISPLAY_OEM_GROUP;
```

For information on each field, see section 2.2.8.1.

2.2.8.7 SAMPR_DOMAIN_DISPLAY_USER_BUFFER

The **SAMPR_DOMAIN_DISPLAY_USER_BUFFER** structure holds an array of **SAMPR_DOMAIN_DISPLAY_USER** elements used to return a list of users through the SamrQueryDisplayInformation family of methods (see section 3.2.5.3).

```
typedef struct _SAMPR_DOMAIN_DISPLAY_USER_BUFFER {
  unsigned long EntriesRead;
  [size_is(EntriesRead)] PSAMPR_DOMAIN_DISPLAY_USER Buffer;
} SAMPR_DOMAIN_DISPLAY_USER_BUFFER,
 *PSAMPR_DOMAIN_DISPLAY_USER_BUFFER;
```

EntriesRead: The number of elements in **Buffer**. If zero, **Buffer** MUST be ignored. If nonzero, **Buffer** MUST point to at least **EntriesRead** number of elements.

Buffer: An array of **SAMPR_DOMAIN_DISPLAY_USER** elements.

2.2.8.8 SAMPR_DOMAIN_DISPLAY_MACHINE_BUFFER

The **SAMPR_DOMAIN_DISPLAY_MACHINE_BUFFER** structure holds an array of **SAMPR_DOMAIN_DISPLAY_MACHINE** elements used to return a list of machine accounts through the SamrQueryDisplayInformation family of methods (see section 3.2.5.3).

```
typedef struct _SAMPR_DOMAIN_DISPLAY_MACHINE_BUFFER {
  unsigned long EntriesRead;
  [size_is(EntriesRead)] PSAMPR_DOMAIN_DISPLAY_MACHINE Buffer;
} SAMPR_DOMAIN_DISPLAY_MACHINE_BUFFER,
 *PSAMPR_DOMAIN_DISPLAY_MACHINE_BUFFER;
```

EntriesRead: The number of elements in **Buffer**. If zero, **Buffer** MUST be ignored. If nonzero, **Buffer** MUST point to at least **EntriesRead** number of elements.

Buffer: An array of **SAMPR_DOMAIN_DISPLAY_MACHINE** elements.

2.2.8.9 SAMPR_DOMAIN_DISPLAY_GROUP_BUFFER

The **SAMPR_DOMAIN_DISPLAY_GROUP_BUFFER** structure holds an array of **SAMPR_DOMAIN_DISPLAY_GROUP** elements used to return a list of groups through the SamrQueryDisplayInformation family of methods (see section 3.2.5.3).

```
typedef struct _SAMPR_DOMAIN_DISPLAY_GROUP_BUFFER {
  unsigned long EntriesRead;
  [size_is(EntriesRead)] PSAMPR_DOMAIN_DISPLAY_GROUP Buffer;
} SAMPR_DOMAIN_DISPLAY_GROUP_BUFFER,
 *PSAMPR_DOMAIN_DISPLAY_GROUP_BUFFER;
```

EntriesRead: The number of elements in **Buffer**. If zero, **Buffer** MUST be ignored. If nonzero, **Buffer** MUST point to at least **EntriesRead** number of elements.

Buffer: An array of **SAMPR_DOMAIN_DISPLAY_GROUP** elements.

2.2.8.10 SAMPR_DOMAIN_DISPLAY_OEM_USER_BUFFER

The **SAMPR_DOMAIN_DISPLAY_OEM_USER_BUFFER** structure holds an array of **SAMPR_DOMAIN_DISPLAY_OEM_USER** elements used to return a list of users through the SamrQueryDisplayInformation family of methods (see section 3.2.5.3).

```
typedef struct _SAMPR_DOMAIN_DISPLAY_OEM_USER_BUFFER {
  unsigned long EntriesRead;
  [size_is(EntriesRead)] PSAMPR_DOMAIN_DISPLAY_OEM_USER Buffer;
} SAMPR_DOMAIN_DISPLAY_OEM_USER_BUFFER,
 *PSAMPR_DOMAIN_DISPLAY_OEM_USER_BUFFER;
```

EntriesRead: The number of elements in **Buffer**. If zero, **Buffer** MUST be ignored. If nonzero, **Buffer** MUST point to at least **EntriesRead** number of elements.

Buffer: An array of **SAMPR_DOMAIN_DISPLAY_OEM_USER** elements.

2.2.8.11 SAMPR_DOMAIN_DISPLAY_OEM_GROUP_BUFFER

The **SAMPR_DOMAIN_DISPLAY_OEM_GROUP_BUFFER** structure holds an array of **SAMPR_DOMAIN_DISPLAY_OEM_GROUP** elements used to return a list of user accounts through the SamrQueryDisplayInformation family of methods (see section 3.2.5.3).

```
typedef struct _SAMPR_DOMAIN_DISPLAY_OEM_GROUP_BUFFER {
  unsigned long EntriesRead;
  [size_is(EntriesRead)] PSAMPR_DOMAIN_DISPLAY_OEM_GROUP Buffer;
} SAMPR_DOMAIN_DISPLAY_OEM_GROUP_BUFFER,
 *PSAMPR_DOMAIN_DISPLAY_OEM_GROUP_BUFFER;
```

EntriesRead: The number of elements in **Buffer**. If zero, **Buffer** MUST be ignored. If nonzero, **Buffer** MUST point to at least **EntriesRead** number of elements.

Buffer: An array of **SAMPR_DOMAIN_DISPLAY_OEM_GROUP** elements.

2.2.8.12 DOMAIN_DISPLAY_INFORMATION

The **DOMAIN_DISPLAY_INFORMATION** enumeration indicates how to interpret the Buffer parameter for **SamrQueryDisplayInformation**, **SamrQueryDisplayInformation2**, **SamrQueryDisplayInformation3**, **SamrGetDisplayEnumerationIndex**, and **SamrGetDisplayEnumerationIndex2**. See section 2.2.8.13 for the list of the structures that are associated with each enumeration.

```
typedef  enum _DOMAIN_DISPLAY_INFORMATION
{
  DomainDisplayUser = 1,
  DomainDisplayMachine,
  DomainDisplayGroup,
  DomainDisplayOemUser,
  DomainDisplayOemGroup
} DOMAIN_DISPLAY_INFORMATION,
 *PDOMAIN_DISPLAY_INFORMATION;
```

2.2.8.13 SAMPR_DISPLAY_INFO_BUFFER

The **SAMPR_DISPLAY_INFO_BUFFER** union is a union of display structures returned by the SamrQueryDisplayInformation family of methods (see section 3.2.5.3). For details on each field, see the associated section for the field structure.

```
typedef
[switch_type(DOMAIN_DISPLAY_INFORMATION)]
union _SAMPR_DISPLAY_INFO_BUFFER {
  [case(DomainDisplayUser)]
    SAMPR_DOMAIN_DISPLAY_USER_BUFFER UserInformation;
  [case(DomainDisplayMachine)]
    SAMPR_DOMAIN_DISPLAY_MACHINE_BUFFER MachineInformation;
  [case(DomainDisplayGroup)]
    SAMPR_DOMAIN_DISPLAY_GROUP_BUFFER GroupInformation;
  [case(DomainDisplayOemUser)]
    SAMPR_DOMAIN_DISPLAY_OEM_USER_BUFFER OemUserInformation;
  [case(DomainDisplayOemGroup)]
    SAMPR_DOMAIN_DISPLAY_OEM_GROUP_BUFFER OemGroupInformation;
} SAMPR_DISPLAY_INFO_BUFFER,
 *PSAMPR_DISPLAY_INFO_BUFFER;
```

2.2.9 SamrValidatePassword Data Types

The following structures are used exclusively for the **SamrValidatePassword** method.

As stated in section 2.1, all structures SHOULD be encrypted by the client using transport layer security to hide any cleartext data embedded in the structures.

The authentication, password change, and password reset structures (sections 2.2.9.5, 2.2.9.6, and 2.2.9.7) refer to a password-related operation that occurs in an application external to this protocol. A canonical scenario is an application such as Microsoft SQL Server that may maintain its own account database (independent of an operating system's account data) and may require that the passwords of those accounts be subject to the same policy as the policy enforced by the responder of this protocol (such as Active Directory). Such an application uses the **SamrValidatePassword** method and these structures to accomplish this goal. Said application is also responsible for storing, in whatever manner it chooses, the **SAM_VALIDATE_PERSISTED_FIELDS (section 2.2.9.2)** structure returned by **SamrValidatePassword**.

2.2.9.1 SAM_VALIDATE_PASSWORD_HASH

The **SAM_VALIDATE_PASSWORD_HASH** structure holds a binary value that represents a cryptographic hash.

```
typedef struct _SAM_VALIDATE_PASSWORD_HASH {
  unsigned long Length;
  [unique, size_is(Length)] unsigned char* Hash;
} SAM_VALIDATE_PASSWORD_HASH,
 *PSAM_VALIDATE_PASSWORD_HASH;
```

Length: The size, in bytes, of **Hash**. If zero, **Hash** MUST be ignored.

Hash: A binary value.

2.2.9.2 SAM_VALIDATE_PERSISTED_FIELDS

The **SAM_VALIDATE_PERSISTED_FIELDS** structure holds various characteristics about password state.

```
typedef struct _SAM_VALIDATE_PERSISTED_FIELDS {
  unsigned long PresentFields;
  LARGE_INTEGER PasswordLastSet;
  LARGE_INTEGER BadPasswordTime;
  LARGE_INTEGER LockoutTime;
  unsigned long BadPasswordCount;
  unsigned long PasswordHistoryLength;
  [unique, size_is(PasswordHistoryLength)]
    PSAM_VALIDATE_PASSWORD_HASH PasswordHistory;
} SAM_VALIDATE_PERSISTED_FIELDS,
 *PSAM_VALIDATE_PERSISTED_FIELDS;
```

PresentFields: A bitmask to indicate which of the fields are valid. The following table shows the defined values. If a bit is set, the corresponding field is valid; if a bit is not set, the field is not valid.

Value	Meaning
SAM_VALIDATE_PASSWORD_LAST_SET 0x00000001	PasswordLastSet
SAM_VALIDATE_BAD_PASSWORD_TIME 0x00000002	BadPasswordTime
SAM_VALIDATE_LOCKOUT_TIME 0x00000004	LockoutTime
SAM_VALIDATE_BAD_PASSWORD_COUNT 0x00000008	BadPasswordCount
SAM_VALIDATE_PASSWORD_HISTORY_LENGTH 0x00000010	PasswordHistoryLength
SAM_VALIDATE_PASSWORD_HISTORY 0x00000020	PasswordHistory

PasswordLastSet: This field represents the time at which the password was last reset or changed. It uses **FILETIME** syntax.

BadPasswordTime: This field represents the time at which an invalid password was presented to either a password change request or an authentication request. It uses **FILETIME** syntax.

LockoutTime: This field represents the time at which the owner of the password data was locked out. It uses **FILETIME** syntax.

BadPasswordCount: Indicates how many invalid passwords have accumulated (see message processing for details).

PasswordHistoryLength: Indicates how many previous passwords are in the **PasswordHistory** field.

PasswordHistory: An array of hash values representing the previous **PasswordHistoryLength** passwords.

2.2.9.3 SAM_VALIDATE_VALIDATION_STATUS

The **SAM_VALIDATE_VALIDATION_STATUS** enumeration defines policy evaluation outcomes.

```
typedef  enum _SAM_VALIDATE_VALIDATION_STATUS
{
  SamValidateSuccess = 0,
  SamValidatePasswordMustChange,
  SamValidateAccountLockedOut,
  SamValidatePasswordExpired,
  SamValidatePasswordIncorrect,
  SamValidatePasswordIsInHistory,
  SamValidatePasswordTooShort,
  SamValidatePasswordTooLong,
  SamValidatePasswordNotComplexEnough,
  SamValidatePasswordTooRecent,
  SamValidatePasswordFilterError
} SAM_VALIDATE_VALIDATION_STATUS,
 *PSAM_VALIDATE_VALIDATION_STATUS;
```

See the message processing of **SamrValidatePassword (section 3.2.5.13.8)** for the semantic meanings of the enumeration values.

2.2.9.4 SAM_VALIDATE_STANDARD_OUTPUT_ARG

The **SAM_VALIDATE_STANDARD_OUTPUT_ARG** structure holds the output of **SamrValidatePassword**.

```
typedef struct _SAM_VALIDATE_STANDARD_OUTPUT_ARG {
  SAM_VALIDATE_PERSISTED_FIELDS ChangedPersistedFields;
  SAM_VALIDATE_VALIDATION_STATUS ValidationStatus;
} SAM_VALIDATE_STANDARD_OUTPUT_ARG,
 *PSAM_VALIDATE_STANDARD_OUTPUT_ARG;
```

ChangedPersistedFields: The password state that has changed. See section 2.2.9.2.

ValidationStatus: The result of the policy evaluation. See section 2.2.9.3.

2.2.9.5 SAM_VALIDATE_AUTHENTICATION_INPUT_ARG

The **SAM_VALIDATE_AUTHENTICATION_INPUT_ARG** structure holds information about an authentication request.

```
typedef struct _SAM_VALIDATE_AUTHENTICATION_INPUT_ARG {
  SAM_VALIDATE_PERSISTED_FIELDS InputPersistedFields;
  unsigned char PasswordMatched;
} SAM_VALIDATE_AUTHENTICATION_INPUT_ARG,
```

```
 *PSAM_VALIDATE_AUTHENTICATION_INPUT_ARG;
```

InputPersistedFields: Password state.

PasswordMatched: A nonzero value indicates that a valid password was presented to the change-password request.

2.2.9.6 SAM_VALIDATE_PASSWORD_CHANGE_INPUT_ARG

The **SAM_VALIDATE_PASSWORD_CHANGE_INPUT_ARG** structure holds information about a password change request.

```
typedef struct _SAM_VALIDATE_PASSWORD_CHANGE_INPUT_ARG {
  SAM_VALIDATE_PERSISTED_FIELDS InputPersistedFields;
  RPC_UNICODE_STRING ClearPassword;
  RPC_UNICODE_STRING UserAccountName;
  SAM_VALIDATE_PASSWORD_HASH HashedPassword;
  unsigned char PasswordMatch;
} SAM_VALIDATE_PASSWORD_CHANGE_INPUT_ARG,
 *PSAM_VALIDATE_PASSWORD_CHANGE_INPUT_ARG;
```

InputPersistedFields: Password state. See section 2.2.9.2.

ClearPassword: The cleartext password of the change-password operation.

UserAccountName: The application-specific logon name of an account performing the change-password operation.

HashedPassword: A binary value containing a hashed form of the value contained in the **ClearPassword** field. The structure of this binary value is specified in section 2.2.9.1. The hash function used to generate this value is chosen by the requester. An example hash function might be MD5 (as specified in [RFC1321]). The responder implementation is independent of that choice; that is, through this protocol, the responder is exposed to a sequence of bytes formatted per section 2.2.9.1, and is, therefore, not exposed to the hash function chosen by the requester. Furthermore, there is no processing by the responder that requires knowledge of the specific hash function chosen. Section 2.2.9 contains more information about a scenario in which this field is used.

PasswordMatch: A nonzero value indicates that a valid password was presented to the change-password request.

2.2.9.7 SAM_VALIDATE_PASSWORD_RESET_INPUT_ARG

The **SAM_VALIDATE_PASSWORD_RESET_INPUT_ARG** structure holds various information about a password reset request.

```
typedef struct _SAM_VALIDATE_PASSWORD_RESET_INPUT_ARG {
  SAM_VALIDATE_PERSISTED_FIELDS InputPersistedFields;
  RPC_UNICODE_STRING ClearPassword;
  RPC_UNICODE_STRING UserAccountName;
  SAM_VALIDATE_PASSWORD_HASH HashedPassword;
```

```
  unsigned char PasswordMustChangeAtNextLogon;
  unsigned char ClearLockout;
} SAM_VALIDATE_PASSWORD_RESET_INPUT_ARG,
 *PSAM_VALIDATE_PASSWORD_RESET_INPUT_ARG;
```

InputPersistedFields: Password state. See section 2.2.9.2.

ClearPassword: The cleartext password of the reset-password operation.

UserAccountName: The application-specific logon name of the account performing the reset-password operation.

HashedPassword: See the specification for **SAM_VALIDATE_PASSWORD_CHANGE_INPUT_ARG (section 2.2.9.6)** for the field with the same name.

PasswordMustChangeAtNextLogon: Nonzero indicates that a password change MUST occur before an authentication request can succeed.

ClearLockout: Nonzero indicates that the lockout state should be reset.

2.2.9.8 PASSWORD_POLICY_VALIDATION_TYPE

The **PASSWORD_POLICY_VALIDATION_TYPE** enumeration indicates the type of policy validation that is being requested.

```
typedef  enum _PASSWORD_POLICY_VALIDATION_TYPE
{
  SamValidateAuthentication = 1,
  SamValidatePasswordChange,
  SamValidatePasswordReset
} PASSWORD_POLICY_VALIDATION_TYPE;
```

SamValidateAuthentication: Indicates a request to execute the password policy validation performed at logon.

SamValidatePasswordChange: Indicates a request to execute the password policy validation performed during a password change request.

SamValidatePasswordReset: Indicates a request to execute the password policy validation performed during a password reset.

2.2.9.9 SAM_VALIDATE_INPUT_ARG

The **SAM_VALIDATE_INPUT_ARG** union holds the various input types to **SamrValidatePassword**.

```
typedef
[switch_type(PASSWORD_POLICY_VALIDATION_TYPE)]
union _SAM_VALIDATE_INPUT_ARG {
  [case(SamValidateAuthentication)]
    SAM_VALIDATE_AUTHENTICATION_INPUT_ARG ValidateAuthenticationInput;
  [case(SamValidatePasswordChange)]
```

```
    SAM_VALIDATE_PASSWORD_CHANGE_INPUT_ARG ValidatePasswordChangeInput;
  [case(SamValidatePasswordReset)]
    SAM_VALIDATE_PASSWORD_RESET_INPUT_ARG ValidatePasswordResetInput;
} SAM_VALIDATE_INPUT_ARG,
 *PSAM_VALIDATE_INPUT_ARG;
```

For more information, see the message processing of **SamrValidatePassword** (section 3.2.5.13.8).

2.2.9.10 SAM_VALIDATE_OUTPUT_ARG

The **SAM_VALIDATE_OUTPUT_ARG** union holds the output of **SamrValidatePassword**.

```
typedef
[switch_type(PASSWORD_POLICY_VALIDATION_TYPE)]
union _SAM_VALIDATE_OUTPUT_ARG {
  [case(SamValidateAuthentication)]
    SAM_VALIDATE_STANDARD_OUTPUT_ARG ValidateAuthenticationOutput;
  [case(SamValidatePasswordChange)]
    SAM_VALIDATE_STANDARD_OUTPUT_ARG ValidatePasswordChangeOutput;
  [case(SamValidatePasswordReset)]
    SAM_VALIDATE_STANDARD_OUTPUT_ARG ValidatePasswordResetOutput;
} SAM_VALIDATE_OUTPUT_ARG,
 *PSAM_VALIDATE_OUTPUT_ARG;
```

For more information, see the message processing of **SamrValidatePassword** (section 3.2.5.13.8).

2.2.10 Supplemental Credentials Structures

These structures define the format of the **supplementalCredentials** attribute in Active Directory that the responder of this protocol updates in the DC configuration. The structures are not part of the SAM Remote Protocol (Client-to-Server), but are listed here in normative detail because the persisted value (in the **supplementalCredentials** attribute) is replicated in Active Directory. See section 3.2.1.8.11 for details on how this attribute is updated.

2.2.10.1 USER_PROPERTIES

The **USER_PROPERTIES** structure defines the format of the **supplementalCredentials** attribute.

```
typedef struct _USER_PROPERTIES {
  unsigned short Reserved[48];
  unsigned short PropertySignature;
  unsigned short PropertyCount;
} USER_PROPERTIES,
 *PUSER_PROPERTIES;
```

Reserved: This value MAY be ignored by the recipient and MAY contain arbitrary values.<24>

PropertySignature: This field MUST be the value 0x50, in little-endian byte order. This is an arbitrary value used to indicate whether the structure is corrupt. That is, if this value is not 0x50 on read, the structure is considered corrupt, processing MUST be aborted, and an error code MUST be returned.

PropertyCount: The number of **USER_PROPERTY** elements (which are variable-length themselves) that follow the **USER_PROPERTIES** structure.

2.2.10.2 USER_PROPERTY

The **USER_PROPERTY** structure defines an array element that contains a single property name and value for the **supplementalCredentials** attribute.

```
typedef struct _USER_PROPERTY {
  unsigned short PropertyLength;
  unsigned short ValueLength;
  unsigned short Reserved;
} USER_PROPERTY,
 *PUSER_PROPERTY;
```

PropertyLength: The number of bytes, in little-endian byte order, of the property name. The property name is located at an offset of zero bytes just following the **Reserved** field. For more information, see the message processing section for supplementalCredentials (section 3.2.1.8.11).

ValueLength: The number of bytes of the property value. The property value is located at an offset of **PropertyLength** bytes following the **Reserved** field. The value MUST be hexadecimal-encoded using an 8-bit character size, and the values '0' through '9' inclusive and 'a' through 'f' inclusive (the specification of 'a' through 'f' is case-sensitive).

Reserved: This value is ignored on read and MAY<25> be set to arbitrary values on update.

2.2.10.3 WDIGEST_CREDENTIALS

The **WDIGEST_CREDENTIALS** structure defines the format of the Primary:WDigest property within the **supplementalCredentials** attribute. This structure is stored as a property value in a **USER_PROPERTY** structure.

```
typedef struct _WDIGEST_CREDENTIALS {
  unsigned char Reserved1;
  unsigned char Version;
  unsigned char NumberOfHashes;
  unsigned char Reserved2[13];
  unsigned char Hash1[16];
  unsigned char Hash2[16];
  unsigned char Hash3[16];
  unsigned char Hash4[16];
  unsigned char Hash5[16];
  unsigned char Hash6[16];
  unsigned char Hash7[16];
  unsigned char Hash8[16];
  unsigned char Hash9[16];
  unsigned char Hash10[16];
  unsigned char Hash11[16];
```

```
  unsigned char Hash12[16];
  unsigned char Hash13[16];
  unsigned char Hash14[16];
  unsigned char Hash15[16];
  unsigned char Hash16[16];
  unsigned char Hash17[16];
  unsigned char Hash18[16];
  unsigned char Hash19[16];
  unsigned char Hash20[16];
  unsigned char Hash21[16];
  unsigned char Hash22[16];
  unsigned char Hash23[16];
  unsigned char Hash24[16];
  unsigned char Hash25[16];
  unsigned char Hash26[16];
  unsigned char Hash27[16];
  unsigned char Hash28[16];
  unsigned char Hash29[16];
} WDIGEST_CREDENTIALS,
 *PWDIGEST_CREDENTIALS;
```

Reserved1: This value is ignored on read and MAY<26> be set to arbitrary values upon an update to the **supplementalCredentials** attribute.

Version: This value MUST be set to 1.

NumberOfHashes: This value MUST be set to 29 because there are 29 hashes in the array.

Reserved2: This value is ignored on read and SHOULD set to be zero.

For information on the **Hash** fields, see section 3.2.1.8.11.

2.2.10.4 KERB_STORED_CREDENTIAL

The **KERB_STORED_CREDENTIAL** structure is a variable-length structure that defines the format of the Primary:Kerberos property within the **supplementalCredentials** attribute. For information on how this structure is created, see section 3.2.1.8.11.4.

This structure is stored as a property value in a **USER_PROPERTY** structure.

```
typedef struct _KERB_STORED_CREDENTIAL {
  unsigned short Revision;
  unsigned short Flags;
  unsigned short CredentialCount;
  unsigned short OldCredentialCount;
  unsigned short DefaultSaltLength;
  unsigned short DefaultSaltMaximumLength;
  unsigned long DefaultSaltOffset;
  unsigned short Padding;
} KERB_STORED_CREDENTIAL,
 *PKERB_STORED_CREDENTIAL;
```

Revision: This value MUST be set to 3.

Flags: This value MUST be zero and ignored on read.

CredentialCount: This is the count of elements in an array that follows the **KERB_STORED_CREDENTIAL** structure that contains the keys for the current password. This value MUST be set to 2.

OldCredentialCount: This is the count of elements in an array that follows the **KERB_STORED_CREDENTIAL** structure that contains the keys for the previous password. This value MUST be set to 0 or 2.

DefaultSaltLength: The length, in bytes, of a salt value.<27>

This value is in little-endian byte order.

DefaultSaltMaximumLength: The length, in bytes, of the buffer containing the salt value.<28>

This value is in little-endian byte order.

DefaultSaltOffset: An offset, in little-endian byte order, from the beginning of the attribute value (that is, from the beginning of the **Revision** field of **KERB_STORED_CREDENTIAL**), to where the salt value starts.<29>

Padding: This value MUST be set to zero.

2.2.10.5 KERB_KEY_DATA

The **KERB_KEY_DATA** structure holds a cryptographic key. This structure is used in conjunction with **KERB_STORED_CREDENTIAL**. For more information, see section 3.2.1.8.11.4.

```
typedef struct _KERB_KEY_DATA {
  unsigned short Reserved1;
  unsigned short Reserved2;
  unsigned long Reserved3;
  long KeyType;
  unsigned long KeyLength;
  unsigned long KeyOffset;
} KERB_KEY_DATA,
 *PKERB_KEY_DATA;
```

Reserved1: This value MUST be set to zero and ignored on receipt.

Reserved2: This value MUST be set to zero and ignored on receipt.

Reserved3: This value MUST be set to zero and ignored on receipt.

KeyType: Indicates the type of key, stored as a 32-bit, unsigned integer in little-endian byte order. Legal values for this field are in the right-hand column of the following table, along with the associated semantic.

Value	Meaning
1	des-cbc-md5 ([RFC3961] section 6.2.1)
2	dec-cbc-crc ([RFC3961] section 6.2.3)

KeyLength: The length, in bytes, of the key value. The value of this member is stored in little-endian byte order.

KeyOffset: An offset, in little-endian byte order, from the beginning of the property value (that is, from the beginning of the **Revision** field of **KERB_STORED_CREDENTIAL**), to where the key value starts. The key value is the hash value specified per the **KeyType**.

2.2.11 Common Algorithms

2.2.11.1 DES-ECB-LM

This section specifies an algorithm to encrypt and decrypt NT and LM hashes that is used throughout the processing of this protocol. The structure that holds an encrypted hash value is found in section 2.2.3.3, which contains references to the methods that use that structure, and hence specify the encryption key to use for processing.

The base algorithm is the DES [FIPS46-2] in ECB mode [FIPS81]. This section specifies how to generate the 64-bit data blocks and 7-bit keys necessary for [FIPS81] from the hash value and the key specified in the referring sections.

For simplicity, this section specifies just the encryption processing. The processing is the same for encryption and decryption; the only exception is when the DES algorithm is invoked in ECB mode. In this case, the implementer MUST specify whether the operation is encryption or decryption. (For more information, see [FIPS81].)

This protocol provides two types of encryption and decryption keys: an unsigned integer and an array of 16 bytes. The exact key is specified in the message processing or syntax sections that reference this section indirectly through section 2.2.3.3.

First, the way to encrypt the hash value is specified, followed by the way to generate the 7-bit keys.

2.2.11.1.1 Encrypting an NT or LM Hash Value with a Specified Key

This section specifies how to encrypt an NT or LM hash (both 16-byte values).

Split the hash value into two blocks, Block1 and Block2. Block1 is the first 8 bytes of the hash (starting from the left); Block2 is the remaining 8 bytes.

Each block is encrypted with a different 7-byte key, call them Key1 and Key2.

If the specified key is an unsigned integer, see section 2.2.11.1.3 for the way to derive Key1 and Key2.

If the specified key is a 16-byte value, see section 2.2.11.1.4 for the way to derive Key1 and Key2.

Let EncryptedBlock1 be the result of applying the algorithm in section 2.2.11.1.2 over Block1 with Key1.

Let EncryptedBlock2 be the result of applying the algorithm in section 2.2.11.1.2 over Block2 with Key2.

The encrypted hash value is the concatenation of EncryptedBlock1 and EncryptedBlock2.

See section 4.3 for an example.

2.2.11.1.2 Encrypting a 64-Bit Block with a 7-Byte Key

Transform the 7-byte key into an 8-byte key as follows:

1. Let InputKey be the 7-byte key, represented as a zero-base-index array.
2. Let OutputKey be an 8-byte key, represented as a zero-base-index array.
3. Let OutputKey be assigned as follows:

```
OutputKey[0] = InputKey[0] >> 0x01;
OutputKey[1] = ((InputKey[0]&0x01)<<6) | (InputKey[1]>>2);
OutputKey[2] = ((InputKey[1]&0x03)<<5) | (InputKey[2]>>3);
OutputKey[3] = ((InputKey[2]&0x07)<<4) | (InputKey[3]>>4);
OutputKey[4] = ((InputKey[3]&0x0F)<<3) | (InputKey[4]>>5);
OutputKey[5] = ((InputKey[4]&0x1F)<<2) | (InputKey[5]>>6);
OutputKey[6] = ((InputKey[5]&0x3F)<<1) | (InputKey[6]>>7);
OutputKey[7] = InputKey[6] & 0x7F;
```

The 7-byte InputKey is expanded to 8 bytes by inserting a 0-bit after every seventh bit.

```
for( int i=0; i<8; i++ )
{
   OutputKey[i] = (OutputKey[i] << 1) & 0xfe;
}
```

4. Let the least significant bit of each byte of OutputKey be a parity bit. That is, if the sum of the preceding seven bits is odd, the eighth bit is 0; otherwise, the eighth bit is 1. The processing starts at the leftmost bit of OutputKey.

Use [FIPS81] to encrypt the 64-bit block using OutputKey. If the higher-level operation is decryption instead of encryption, this is the point at which an implementer MUST specify the decryption intent to [FIPS81].

2.2.11.1.3 Deriving Key1 and Key2 from a Little-Endian, Unsigned Integer Key

Let I be the little-endian, unsigned integer.

Let I[X] be the Xth byte of I, where I is interpreted as a zero-base-index array of bytes. Note that because I is in little-endian byte order, I[0] is the least significant byte.

Key1 is a concatenation of the following values: I[0], I[1], I[2], I[3], I[0], I[1], I[3].

Key2 is a concatenation of the following values: I[4], I[0], I[1], I[2], I[3], I[0], I[1].

2.2.11.1.4 Deriving Key1 and Key2 from a 16-Byte Key

Let Key1 be the first 7 bytes of the 16-byte key.

Let Key2 be the next 7 bytes of the 16-byte value. For example, consider a zero-base-index array of 16 bytes called KeyArray that contains the 16-byte key. Key2 is comprised of the bytes KeyArray[7] through KeyArray[13] inclusive.

Note A consequence of this derivation is that the fifteenth and sixteenth bytes are ignored.

3 Protocol Details

3.1 Requester Details

3.1.1 Abstract Data Model

There is no normative data model for the requester. The input data may come from any source, and the returned data can be processed by the requester in any manner.

3.1.2 Security Model

The requester MUST create a secure RPC session such that the responder can identify and determine the authorization for the requester. (For more information on secure RPC, see [MS-RPCE].) This requirement exists so that the responder can implement its security model (section 3.2.2).

3.1.2.1 RC4 Cipher Usage

The data MUST be encrypted and decrypted using the **RC4** algorithm. The key, required during runtime by the RC4 algorithm, MUST be the 16-byte key specified by the method using this structure (for examples, see sections 3.2.5.10.2, 3.2.5.10.3, and 3.2.5.10.4). The encrypted portion of the SAMPR_ENCRYPTED_USER_PASSWORD_NEW.Buffer structure MUST be protected in the same way, but the 16-byte key is specified in section 3.1.2.2.

3.1.2.2 MD5 Usage

The key required during runtime by the RC4 encryption algorithm that encrypts and decrypts the protected portion of the SAMPR_ENCRYPTED_USER_PASSWORD_NEW.Buffer is specified by the following pseudocode.

```
CALL MD5Init(md5context)
CALL MD5Update(md5context, SAMPR_USER_PASSWORD_NEW.ClearSalt, 16)
CALL MD5Update(md5context, user-session-key, 16)
CALL MD5Final(md5context)
```

Where:

- MD5Init, MD5Update, and MD5Final are predicates/functions defined in [RFC1321].
- md5Context is a variable of type MD5_CTX, as specified in [RFC1321].
- user-session-key is a 16-byte value obtained from the 16-byte SMB [MS-SMB] session key established by the underlying authentication protocol (either Kerberos [MS-KILE] or NTLM [MS-NLMP]).

3.1.3 Timers

There are no timers used by the protocol.

3.1.4 Initialization

There is no initialization required to use this protocol.

3.1.5 Message Processing Events and Sequencing Rules

To obtain any context handle to the responder, one of the following methods MUST be called initially: **SamrConnect2**, **SamrConnect4**, or **SamrConnect5**. With the *ServerHandle* parameter returned from these methods, it is possible to obtain other context handles and call any associated methods on the handle. See section 4.1 for an example.

Note The following messages do not require a context handle and can be called directly; they also do not return any context handle:

- **SamrGetDomainPasswordInformation**
- **SamrSetDSRMPassword**
- **SamrValidatePassword**
- **SamrOemChangePasswordUser2**
- **SamrUnicodeChangePasswordUser2**

Note A user account MUST be enabled by clearing the UF_ACCOUNTDISABLE bit from the **userAccountControl** attribute before that account will be able to authenticate, as specified in [MS-KILE] section 3.3.5.4.1.

3.1.6 Timer Events

The protocol does not include its own timer events. Information about any transport-level timers is specified in [MS-RPCE].

3.1.7 Other Local Events

None.

3.2 Responder Details

This protocol enables create, read, update, and delete semantics over an **account domain**, as specified in [MS-SECO] section 2.2. Five abstract objects are exposed through this protocol: server, domain, group, alias, and user. User, group, and alias objects may be created and deleted; all objects may be updated and read.

This specification uses the Active Directory data model, as specified in the entire document of [MS-ADTS], for the responder of this protocol. The attribute names specified in this section are normative for the DC configuration. Section 3.2.1 contains a brief overview of that data model that is relevant to this protocol.

Because the behavior of this protocol is very similar between the DC and non-DC configurations, the Active Directory data model is also used for the non-DC configuration. However, when implementing this protocol for the non-DC scenario, the names of attributes in the data model are not normative. For example, it is conceivable that the backing store in a non-DC configuration could be a text file written and read solely by the responder of this protocol.

3.2.1 Abstract Data Model

In the DC configuration, this protocol operates over a directory database that is composed of a set of named objects. The name format is an X.500 name; therefore, the objects are arranged in a hierarchy by name. Each object's name MUST be unique within the directory. In a non-DC

configuration, the name format of X.500 is not normative; this specification assumes the format is X.500 for consistency between the two configurations.

Each object possesses a collection of attributes. Attributes may be multivalued. Each attribute is identified by a value called ldapDisplayName. For example, the X.500 name of the object is a single-valued attribute with the ldapDisplayName: distinguishedName. This specification describes the constraints on the attributes for behaviors relevant to this protocol. For the DC configuration, [MS-ADTS] section 3.1.1.5 contains additional constraints.

Objects are retrieved from the directory database by specifying attribute-value constraints that the object's attributes (and their values) must satisfy. Attribute values are updated by identifying the target object by **distinguishedName** and specifying the new set of attribute-value pairs. Section 3.2.1.3 and section 3.2.1.4 contain a list of the Active Directory attributes and classes relevant to this protocol.

Implementations MUST support creating, reading, updating, and deleting multiple objects, attributes, and attribute values with ACID (atomic, consistent, isolated, and durable) properties. Such an update is referred to as a transaction in this specification.

A user object refers to a **database object** whose **objectClass** attribute is user or derived from user.

A computer object refers to a database object whose **objectClass** attribute is computer or derived from computer.

A group object refers to a database object whose **objectClass** attribute is group or derived from group, and whose **groupType** contains GROUP_TYPE_ACCOUNT_GROUP or GROUP_TYPE_UNIVERSAL_GROUP.

An alias object refers to a database object whose **objectClass** attribute is group or derived from group, and whose **groupType** contains GROUP_TYPE_RESOURCE_GROUP.

Two domains are exposed from a given responder: an account domain and a **built-in domain**; this fact is true for both DC and non-DC configurations. The account domain refers to the object with **objectClass** domainDNS. The built-in domain refers to the object with the **objectClass** builtinDomain. The built-in domain has the characteristic that its **objectSid** value is invariant (S-1-5-32) through all deployments and only contains aliases. There is exactly one built-in domain for every account domain. When opening a domain object, through **SamrOpenDomain (section 3.2.5.1.5)**, a requester selects the domain to open based on the DomainId parameter.

Domain object refers to either the account domain or built-in domain.

Server object refers to the single object in the account domain with the samServer objectClass.

The following sections normatively describe the database constraints and triggers required for the message processing of this protocol.

- Constraints are relationships between attributes that MUST be satisfied for a database update to be successful. The constraints are specified in section 3.2.1.6.
- Triggers are actions that MUST be executed for a database update to be successful. An attribute-scoped trigger is a trigger that is executed when a particular attribute is updated. The attribute-scoped triggers are specified in section 3.2.1.8.

Error statuses (also called error codes) generated by a failure to comply with a constraint are in the NTSTATUS space (a long data type), as specified in [MS-ERREF] section 2.3. Unless specifically called out, error codes are returned to the requester of the protocol and are not handled by any

special processing at the requester; therefore, the exact error code is implementation-specific. Cases in which the requester may handle a specific error code are called out. The set of such error codes is found in section 2.2.1.15.

3.2.1.1 String Handling

The data model for storing an attribute of syntax "string" is a UTF-16 encoded string not including the NULL character. In this protocol, a string is represented within an **RPC_UNICODE_STRING** structure, which is a counted string.

When a string to be stored in the database arrives through this protocol, it MUST be processed such that the database attribute is updated with RPC_UNICODE_STRING.Length bytes of RPC_UNICODE_STRING.Buffer.

When a database attribute is to be returned as an **RPC_UNICODE_STRING** via this protocol, RPC_UNICODE_STRING.Length MUST be the count of bytes stored in the database for that attribute, and RPC_UNICODE_STRING.Buffer MUST contain the database value for that attribute.

In addition, when receiving an **RPC_UNICODE_STRING** or **RPC_STRING**, if the **Length** field is nonzero and the **Buffer** field is NULL, an error MUST be returned.

3.2.1.2 String Matching

When string matching is required by the message processing (for example, when processing a **SamrCreateGroupInDomain** method and the data model checks for uniqueness of the name property), the following string matching rules apply:

1. On a DC configuration, refer to [MS-ADTS] section 7.5 for how strings are compared.
2. When comparing two strings on a non-DC configuration, they MUST be compared in a case-insensitive manner.<30>

3.2.1.3 Attribute Listing

The following attributes are referenced by this protocol (listed by ldapDisplayName). For a normative description of the syntax, see [MS-ADA1], [MS-ADA2], and [MS-ADA3].

- accountExpires
- badPasswordTime
- badPwdCount
- codePage
- countryCode
- dBCSPwd
- description
- displayName
- domainReplica
- forceLogoff

- groupType
- homeDirectory
- homeDrive
- memberOf
- lastLogoff
- lastLogon
- lmPwdHistory
- lockoutObservationWindow
- lockoutDuration
- lockoutThreshold
- lockoutTime
- logonCount
- logonHours
- maxPwdAge
- member
- minPwdAge
- minPwdLength
- mS-DS-CreatorSID
- mS-DS-MachineAccountQuota
- msDS-LockoutObservationWindow
- msDS-LockoutDuration
- msDS-LockoutThreshold
- msDS-MaximumPasswordAge
- msDS-MinimumPasswordAge
- msDS-MinimumPasswordLength
- msDS-PasswordComplexityEnabled
- msDS-PasswordHistoryLength
- msDS-PasswordReversibleEncryptionEnabled
- ntPwdHistory
- nTSecurityDescriptor

- objectClass
- objectSid
- oEMInformation
- primaryGroupID
- profilePath
- pwdHistoryLength
- pwdLastSet
- pwdProperties
- sAMAccountName
- SAMAccountType
- scriptPath
- serverState
- supplementalCredentials
- uASCompat
- unicodePwd
- userAccountControl
- comment
- userParameters
- userWorkstations
- objectClass
- clearTextPassword *

* This attribute is not directly persisted. It has triggers that are applied when an update occurs that, in turn, can update other attributes. As such, it is not found in the Active Directory schema.

3.2.1.4 Object Class List

The following classes are referenced by this protocol (listed by ldapDisplayName). For a normative description of these classes, see [MS-ADSC].

- user
- computer
- domainDNS
- samServer
- builtinDomain

- group

3.2.1.5 Password Settings Attributes for Originating Update Constraints

The following computed attributes are defined for each user object. These attributes are read-only.

Effective-LockoutObservationWindow: A 64-bit value with Delta-Time syntax, indicating the time period in which bad password attempts are counted without resetting the count to zero.

Effective-LockoutDuration: A 64-bit value with Delta-Time syntax, indicating the duration for which an account is locked out before being automatically reset to an unlocked state.

Effective-LockoutThreshold: A 16-bit unsigned integer indicating the number of bad password attempts within an Effective-LockoutObservationWindow that will cause an account to be locked out.

Effective-MaximumPasswordAge: A 64-bit value with Delta-Time syntax, indicating the policy setting for the maximum time allowed before a password reset or change is required.

Effective-MinimumPasswordAge: A 64-bit value with Delta-Time syntax, indicating the policy setting for the minimum time allowed before a password change operation is allowed.

Effective-MinimumPasswordLength: A 16-bit unsigned integer indicating the minimum password-length policy setting.

Effective-PasswordComplexityEnabled: A Boolean value indicating that password complexity rules (as defined in section 3.2.1.7.1) are enabled for the user.

Effective-PasswordHistoryLength: A 16-bit unsigned integer indicating the policy setting for the password history length.

Effective-PasswordReversibleEncryptionEnabled: A Boolean value indicating that the user's cleartext password is to be stored in the **supplementalCredentials** attribute, as defined in section 3.2.1.8.11.

The values for these attributes on user objects are computed according to the following algorithm:

1. If the server is in a DC configuration, and the msDS-ResultantPSO computed attribute (as specified in [MS-ADTS] section 3.1.1.4.5.36) on the user object has value O, values are calculated as follows using attribute values on object O:
    1. Effective-LockoutObservationWindow = msDS-LockoutObservationWindow
    2. Effective-LockoutDuration = msDS-LockoutDuration
    3. Effective-LockoutThreshold = msDS-LockoutThreshold
    4. Effective-MaximumPasswordAge = msDS-MaximumPasswordAge
    5. Effective-MinimumPasswordAge = msDS-MinimumPasswordAge
    6. Effective-MinimumPasswordLength = msDS-MinimumPasswordLength
    7. Effective-PasswordComplexityEnabled = msDS-PasswordComplexityEnabled
    8. Effective-PasswordHistoryLength = msDS-PasswordHistoryLength
    9. Effective-PasswordReversibleEncryptionEnabled = true if either of the following is true:

- The value of msDS-PasswordReversibleEncryptionEnabled is true.
- pwdProperties on the domain object contains DOMAIN_PASSWORD_STORE_CLEARTEXT.

Otherwise, false.

2. Otherwise, values are calculated as follows using attribute values on the domain object:
   1. Effective-LockoutObservationWindow = lockoutObservationWindow on the domain object.
   2. Effective-LockoutDuration = lockoutDuration on the domain object.
   3. Effective-LockoutThreshold = lockoutThreshold on the domain object.
   4. Effective-MaximumPasswordAge = maxPwdAge on the domain object.
   5. Effective-MinimumPasswordAge = minPwdAge on the domain object.
   6. Effective-MinimumPasswordLength = minPwdLength on the domain object.
   7. Effective-PasswordComplexityEnabled = true if pwdProperties on the domain object contains DOMAIN_PASSWORD_COMPLEX; otherwise, false.
   8. Effective-PasswordHistoryLength = pwdHistoryLength on the domain object.
   9. Effective-PasswordReversibleEncryptionEnabled = true if pwdProperties on the domain object contains DOMAIN_PASSWORD_STORE_CLEARTEXT; otherwise, false.

3.2.1.6 Attribute Constraints for Originating Updates

The following attribute constraints MUST be enforced during originating updates to the database.

The term "previous" refers to the value at the beginning of the transaction before any updates occurred. Unless otherwise specified, other attributes referenced for a particular constraint refer to the attribute on the same object as the attribute whose constraint is currently being satisfied. An exception to this rule is for Password Settings Attributes (section 3.2.1.5).

Unless specifically called out, all failure codes are implementation-specific.

A client implementation MUST treat all failure codes as complete failures of the requested operation unless explicitly noted in this section. The possible status codes used for these explicit return codes are found in section 2.2.1.15.

1. lockoutObservationWindow MUST be greater than or equal to lockoutDuration; on error, return a failure code. "Greater than", in this context, means a smaller absolute value since both are negative (see the next two constraints).
2. lockoutObservationWindow MUST be less than or equal to 0; on error, return a failure code.
3. lockoutDuration MUST be less than or equal to 0; on error, return a failure code.
4. maxPwdAge MUST be less than or equal to 0; on error, return a failure code.
5. minPwdAge MUST be less than or equal to 0; on error, return a failure code.
6. minPwdLength MUST be less than or equal to 256 unless UASCompat is nonzero, in which case minPwdLength MUST be less than or equal to 20; on error, return a failure code.

7. pwdHistoryLength MUST be less than or equal to 1024; on error, return a failure code.

8. sAMAccountName MUST contain at least one non-blank character; on error, return a failure code.

9. sAMAccountName MUST NOT end with a '.' (period) character; on error, return a failure code.

10. sAMAccountName MUST NOT contain any of the following characters (shown here as the binary values of UTF-16 encoded characters):

```
characters 0x00 through 0x1F, inclusive, and...
0x22 0x2F 0x5C 0x5B 0x5D 0x3A 0x7C 0x3C
0x3E 0x2B 0x3D 0x3B 0x3F 0x2C 0x2A
```

    On error, return a failure code.

11. sAMAccountName MUST contain less than or equal to 20 characters if the object's objectClass is user; on error, return a failure code.

12. sAMAccountName MUST contain less than or equal to 256 characters if the object's objectClass is group; on error, return a failure code.

13. sAMAccountName MUST be the value "krbtgt" (UTF-16 encoded), if the RID of the objectSid attribute is DOMAIN_USER_RID_KRBTGT; on error, return a failure code.

14. accountExpires MUST be equal to 0 if the RID of the objectSid attribute value is DOMAIN_USER_RID_ADMIN; on error, return a failure code.

15. logonHours MUST conform to the binary structure of **SAMPR_LOGON_HOURS (section 2.2.7.5)**, and SAMPR_LOGON_HOURS.UnitsPerWeek MUST be less than or equal to 10080; on error, return a failure code.

16. userWorkstations MUST conform to the following constraints, with the value interpreted as a UTF-16 encoded string:

    1. The string MUST be composed of substrings separated by a ',' (comma) character; therefore, a substring cannot contain a comma character. Specifically:

        1. If no comma is present, there is one substring, and it is equal to the string itself.

        2. A comma MUST NOT be the first or final character in the value.

        3. If a comma is present, the first substring MUST be the characters starting from the start of the value to the character just preceding the first comma; the final substring MUST be the characters starting just after the final comma to the final character in the string.

    2. Each substring MUST be less than or equal to 256 characters.

    3. Each substring MUST satisfy at least one of the following conditions:

        1. Satisfy the DNS naming syntax for a full DNS host name, as specified in [RFC1123] section 2.1.

        2. Have a length greater than 1 character and less than or equal to 20 characters, not have a leading or trailing blank (0x10) character, and not contain any of the following characters:

```
characters of the value 0x00 through 0x1F, inclusive, and...
```

```
0x22 0x2F 0x5C 0x5B 0x5D 0x3A
0x7C 0x3C 0x3E 0x2B 0x3D 0x3B
0x3F 0x2C 0x2A
```

4. Any processing error or constraint violation MUST return a failure code.

17. primaryGroupId MUST be equal to DOMAIN_GROUP_RID_CONTROLLERS if userAccountControl contains the bit UF_SERVER_TRUST_ACCOUNT; on error, return a failure code.

18. userAccountControl MUST contain only the following bits, as defined in section 2.2.1.13. Note that constraints in this section further limit the possible variations that are legal.

Bits
UF_ACCOUNTDISABLE
UF_HOMEDIR_REQUIRED
UF_PASSWD_NOTREQD
UF_ENCRYPTED_TEXT_PASSWORD_ALLOWED
UF_NORMAL_ACCOUNT
UF_INTERDOMAIN_TRUST_ACCOUNT
UF_WORKSTATION_TRUST_ACCOUNT
UF_SERVER_TRUST_ACCOUNT
UF_DONT_EXPIRE_PASSWD
UF_MNS_LOGON_ACCOUNT
UF_SMARTCARD_REQUIRED
UF_TRUSTED_FOR_DELEGATION
UF_NOT_DELEGATED
UF_USE_DES_KEY_ONLY
UF_DONT_REQUIRE_PREAUTH
UF_TRUSTED_TO_AUTHENTICATE_FOR_DELEGATION
UF_NO_AUTH_DATA_REQUIRED

19. userAccountControl MUST contain one and only one of the following bits, as defined in section 2.2.1.13; on error, return a failure code.

Bits
UF_NORMAL_ACCOUNT
UF_INTERDOMAIN_TRUST_ACCOUNT

Bits
UF_WORKSTATION_TRUST_ACCOUNT
UF_SERVER_TRUST_ACCOUNT

20. userAccountControl MUST NOT contain the UF_ACCOUNTDISABLE bit if the RID of objectSid has the value DOMAIN_USER_RID_ADMIN or DOMAIN_USER_RID_KRBTGT; on error, return a failure code.

21. objectClass MUST be of type computer or derived from computer if userAccountControl contains the following bit: UF_SERVER_TRUST_ACCOUNT.

22. unicodePwd MUST be exactly 16 bytes in length or not present.

23. dbcsPwd MUST be exactly 16 bytes in length or not present.

24. lmPwdHistory MUST have the following binary format:

    1. The length MUST be a multiple of 16 bytes.

    2. If a value is present, the first 16 bytes MUST be equal to the current value of dbcsPwd.

25. ntPwdHistory MUST have the following binary format:

    1. The length MUST be a multiple of 16 bytes.

    2. If a value is present, the first 16 bytes MUST be equal to the current value of unicodePwd.

26. groupType MUST contain only bits specified in section 2.2.1.11.

27. groupType MUST NOT contain GROUP_TYPE_UNIVERSAL if the account domain is in mixed mode.

28. groupType MUST NOT be changed after it has been added if the account domain is in mixed mode.

3.2.1.7 Additional Update Constraints

The following constraints are referenced from other constraints or triggers.

3.2.1.7.1 General Password Policy

This policy is referenced from the dbcsPwd and unicodePwd triggers.

The following constraints MUST be satisfied; on error, the responder MUST return a processing error. For more information on error codes, see section 3.2.5.

1. Minimum Password Length Constraint: If all of the following conditions are true, the following constraint MUST be satisfied:

    1. Conditions:

        1. The **userAccountControl** attribute value contains UF_NORMAL_ACCOUNT.

        2. The **objectSid** attribute value does not have the DOMAIN_USER_KRBTGT value as the RID.

        3. The **userAccountControl** attribute value does NOT contain UF_PASSED_NOTREQD.

4. The Effective-MinimumPasswordLength attribute value (see section 3.2.1.5) is greater than 0.

5. The requesting protocol message is a password change (as compared to a password set).

2. Constraint:

   - At least one of dbcsPwd or unicodePwd MUST be nonzero-length and equal to a value other than the hash of a zero-length string.

2. Minimum Password Age Constraint: If all of the following conditions are true, the following constraint MUST be satisfied:

   1. Conditions:

      1. The **userAccountControl** attribute contains UF_NORMAL_ACCOUNT.

      2. At least one of the **dbcsPwd** or **unicodePwd** attribute values is present and not equal to a hash value of a zero-length string.

   2. Constraint:

      - The **pwdLastSet** attribute MUST be less than the current time plus the value of the Effective-MinimumPasswordAge attribute (see section 3.2.1.5).

3. Password History Length Constraint: If all of the following conditions are true, the following constraints MUST be satisfied:

   1. Conditions:

      1. The **userAccountControl** attribute contains UF_NORMAL_ACCOUNT.

      2. objectSid does not have the DOMAIN_USER_KRBTGT value as the RID.

      3. userAccountControl does NOT contain UF_PASSED_NOTREQD.

      4. minPwdHistory on the account domain object is greater than 0.

      5. The requesting protocol message is a password change (as compared to a password set).

   2. Constraints:

      1. If the **unicodePwd** attribute is being updated, the value of the unicodePwd MUST NOT be present in the first N hashes stored in the ntPwdHistory attribute value, where N is the value of the Effective-PasswordHistoryLength attribute (see section 3.2.1.5). For details on how ntPwdHistory is maintained, see section 3.2.1.9.1.

      2. If the **dbcsPwd** attribute is being updated, the value of the **dbcsPwd** MUST NOT be present in the first N hashes stored in the lmPwdHistory attribute value, where N is the value of the Effective-PasswordHistoryLength attribute (see section 3.2.1.5). For details on how lmPwdHistory is maintained, see section 3.2.1.9.1.

3.2.1.7.2 Cleartext Password Policy

This constraint is referenced when a cleartext password is updated.

The following constraints MUST be satisfied; on error, the responder MUST return a processing error. For more information on error codes, see section 3.2.5.

1. The value MUST be interpreted as a UTF-16 encoded string. If the length of the value is an odd-byte count, ignore the final byte, interpret the remaining characters as a UTF-16 encoded string, and ignore the last constraint (starting with the text, "If the Effective-PasswordComplexityEnabled value...").
2. The value MUST be less than or equal to 256 characters (this constraint is called the "maximum password length constraint").
3. The value MUST satisfy all of the following constraints if all of the following conditions are met:
    1. Conditions:
        1. The userAccountControl attribute value contains UF_NORMAL_ACCOUNT.
        2. objectSid does not have the DOMAIN_USER_KRBTGT value as the RID.
        3. userAccountControl does not contain UF_PASSED_NOTREQD.
    2. Constraints:
        1. The value MUST have a length no smaller than the value of the Effective-MinimumPasswordLength attribute (see section 3.2.1.5). This constraint is called the "minimum password length constraint".
        2. The value MUST NOT contain the sAMAccountName attribute value as a case-insensitive substring if that value contains more than 2 characters.
        3. The value MUST NOT contain any token of the displayName attribute value, where a token is defined as a case-insensitive substring greater than two characters, and delimited by one or more of the following characters:

            ```
            0x20 0x2c 0x2e 0x9 0x2d 0x5f 0x23
            ```

        4. If the Effective-PasswordComplexityEnabled value (see section 3.2.1.5) is set, the password MUST contain characters from at least three of the following five classes:
            1. English Uppercase Letters: characters 0x41 to 0x56, inclusive.
            2. English Lowercase Letters: characters 0x62 to 0x7a, inclusive.
            3. Westernized Arabic Numerals: characters 0x30 to 0x39, inclusive.
            4. Any character from [UNICODE3.1] that is categorized as Lu, LI, Lt, Lm, Lo.
            5. The following characters:

                ```
                0x28 0x60 0x7e 0x21 0x40 0x23 0x24 0x25
                0x5e 0x26 0x2a 0x5f 0x2d 0x2b 0x3d 0x7c
                0x5c 0x7b 0x7d 0x5b 0x5d 0x3a 0x3b 0x22
                0x27 0x3c 0x3e 0x2c 0x2e 0x3f 0x29 0x2f
                ```

3.2.1.8 Attribute Triggers for Originating Updates

The following attribute-scoped triggers MUST be executed during originating updates to the database.

The term "previous" refers to the value at the beginning of the transaction, before any updates occurred. Unless otherwise specified, other attributes referenced for a particular trigger refer to the attribute on the same object as the attribute whose trigger is currently being executed.

3.2.1.8.1 objectClass

1. If the **objectClass** attribute value is user or computer, or derived from either of these classes, all of the following constraints MUST be satisfied:

    1. The **objectSid** attribute MUST be updated per the supplemental trigger specified in section 3.2.1.9.2.

    2. The following attributes MUST be updated with the associated values if no value is present in the database.

Attribute	Value
badPwdCount	0
codePage	0
countryCode	0
badPasswordTime	0
lastLogoff	0
lastLogon	0
pwdLastSet	0
accountExpires	0xFFFFFFF FFFFFFFF
logonCount	0

    3. The **samAccountType** attribute MUST be updated with the value dictated by the new value in the **userAccountControl** attribute. The value of the **userAccountControl** attribute in the database need only contain the specified bit in the table, not test for equality.

userAccountControl	samAccountType
UF_NORMAL_ACCOUNT	SAM_USER_OBJECT
UF_INTERDOMAIN_TRUST_ACCOUNT	SAM_TRUST_ACCOUNT
UF_WORKSTATION_TRUST_ACCOUNT	SAM_MACHINE_ACCOUNT
UF_SERVER_TRUST_ACCOUNT	SAM_MACHINE_ACCOUNT

    4. The **primaryGroupId** attribute MUST be updated with the value dictated by the value in the **userAccountControl** attribute. The value of the **userAccountControl** attribute in the database need only contain the specified bit in the table, not test for equality.

userAccountControl	primaryGroupId
UF_NORMAL_ACCOUNT	DOMAIN_GROUP_RID_USERS

userAccountControl	primaryGroupId
UF_INTERDOMAIN_TRUST_ACCOUNT	DOMAIN_GROUP_RID_USERS
UF_WORKSTATION_TRUST_ACCOUNT	DOMAIN_GROUP_RID_COMPUTERS
UF_SERVER_TRUST_ACCOUNT	DOMAIN_GROUP_RID_CONTROLLERS

5. The **userAccountControl** attribute MUST be updated with the bits specified in the table below using a bitwise OR. The value of the **userAccountControl** attribute in the database need only contain the specified bit in the table, not test for equality.

userAccountControl	userAccountControl bits to augment existing value
UF_NORMAL_ACCOUNT	UF_ACCOUNTDISABLE UF_PASSWD_NOTREQD
UF_WORKSTATION_TRUST_ACCOUNT	UF_PASSWD_NOTREQD

2. If the **objectClass** attribute value is group or is derived from this class, all of the following constraints MUST be satisfied:

   1. The **objectSid** attribute MUST be updated per the supplemental trigger specified in section 3.2.1.9.2.

   2. The **groupType** attribute MUST be updated, if no value is present in the database, with the value: GROUP_TYPE_SECURITY_ACCOUNT.

   3. The **samAccountType** attribute MUST be updated with the value dictated by an exact match with the value in the groupType attribute.

groupType	samAccountType
GROUP_TYPE_SECURITY_ACCOUNT	SAM_GROUP_OBJECT
GROUP_TYPE_ACCOUNT_GROUP	SAM_NON_SECURITY_GROUP_OBJECT
GROUP_TYPE_SECURITY_RESOURCE	SAM_ALIAS_OBJECT
GROUP_TYPE_RESOURCE_GROUP	SAM_NON_SECURITY_ALIAS_OBJECT
GROUP_TYPE_SECURITY_UNIVERSAL	SAM_GROUP_OBJECT
GROUP_TYPE_UNIVERSAL_GROUP	SAM_NON_SECURITY_GROUP_OBJECT

3.2.1.8.2 primaryGroupID

Let G be the group object such that the primaryGroupId value is the RID of the group's **objectSid** attribute.

Let G' be the group object such that the previous primaryGroupId value is the RID of the group's **objectSid** attribute.

If the update to primaryGroupID is a NOT a result of an internal trigger, all of the following constraints MUST be satisfied:

1. The current object MUST be a member of G.

2. The groupType attribute of G MUST be one of the following two values: GROUP_TYPE_SECURITY_ACCOUNT or GROUP_TYPE_SECURITY_RESOURCE.

3. The current object MUST be removed from the member attribute of G'.

3.2.1.8.3 lockoutTime

If the lockoutTime attribute value is 0, badPwdCount MUST be updated to the value of 0.<31>

3.2.1.8.4 sAMAccountName

1. If the **objectSid** attribute has a RID of DOMAIN_RID_USER_KRBTGT and there is already a value present in the **sAMAccountName** attribute, the responder MUST return an error status.

2. If the **sAMAccountName** attribute value is NOT unique with respect to the union of all **sAMAccountName** and **msDS-AdditionalSamAccountName** attribute values for all other objects within the scope of the account and built-in domain, the responder MUST return an error status, per the following conditions:

Condition	Error status
The objectClass attribute of the object whose **sAMAccountName** matches the **sAMAccountName** attribute of the current object is group.	STATUS_GROUP_EXISTS
Otherwise:	STATUS_USER_EXISTS

3.2.1.8.5 clearTextPassword

1. If the **pwdProperties** attribute value on the account domain object contains the DOMAIN_PASSWORD_NO_CLEAR_CHANGE bit, the responder MUST abort the request and return an error status.

2. If the RID of the **objectSid** attribute is DOMAIN_USER_RID_KRBTGT and the requesting protocol is a change-password protocol, the responder MUST abort the request and return an error status.

3. If the RID of the **objectSid** attribute is DOMAIN_USER_RID_KRBTGT and the requesting protocol is a set-password protocol, the value of clearTextPassword MUST be replaced with a randomly generated value that satisfies all criteria in section 3.2.1.7.2.

4. The constraints in section 3.2.1.7.2 MUST be satisfied.

5. The **unicodePwd** attribute MUST be updated with the NT hash of new value.

6. The **dbcsPwd** attribute MUST be updated with the LM hash of new value.

7. On a DC configuration, the **supplementalCredentials** attribute MUST be updated with the cleartext value (see section 3.2.1.8.11 for processing details on how **supplementalCredentials** is updated).

3.2.1.8.6 dbcsPwd

1. The constraints in section 3.2.1.7.1 MUST be satisfied.

2. The new value MUST be encrypted before being persisted. Encryption is accomplished using the algorithm specified in section 2.2.11.1, with the RID (an unsigned integer) as the encryption key.

3. If the requester has access to the ccc2dc7d-a6ad-4a7a-8846-c04e3cc53501 **control access right** on the domain object, **pwdLastSet** MUST be updated to the current time; otherwise, **pwdLastSet** MUST be updated to the value zero, which causes the new password to expire immediately.

4. If the update to this attribute is not from an internal trigger, the **supplementalCredential** attribute MUST be removed.

5. The **lmPwdHistory** attribute MUST be updated with the new **dbcsPwd** attribute value (encrypted with the RID, per constraint 2) per the constraints in section 3.2.1.9.1.

3.2.1.8.7 unicodePwd

1. The constraints in section 3.2.1.7.1 MUST be satisfied.

2. The new value MUST be encrypted before being persisted. Encryption is accomplished using the algorithm specified in section 2.2.11.1, with the RID (an unsigned integer) as the encryption key.

3. If the requester has access to the ccc2dc7d-a6ad-4a7a-8846-c04e3cc53501 control access right on the domain object, **pwdLastSet** MUST be updated to the current time; otherwise **pwdLastSet** MUST be updated to the value zero, which causes the new password to expire immediately.

4. If the update to this attribute is not from an internal trigger, the **supplementalCredential** attribute MUST be removed.

5. The **ntPwdHistory** attribute MUST be updated with the new **unicodePwd** attribute value (encrypted with the RID, per constraint 2) per the constraints in section 3.2.1.9.1.

3.2.1.8.8 pwdLastSet

See the Windows behavior information.<32>

3.2.1.8.9 member

1. If all of the following conditions are true, the subsequent constraint MUST be satisfied.

    1. Conditions:

        1. The value contains a SID-only dsname value.

        2. The dsname value does not resolve to an existing object in the domain NC.

        3. The responder is in a DC configuration, and the domain prefix of the SID value is not equal to any domain SID in the forest; or, the responder is in a non-DC configuration and the value is different than the account domain SID.

    2. Constraint:

        - A new object with the following characteristics MUST be created with the following attributes and values. The dsname value added to the member attribute MUST reference this object.

Attribute	Value
objectClass	fpo
objectSid	The SID value of the new dsname value
distinguishedName	The parent MUST be the well-known object container for foreign principal objects. (More information about this container is specified in [MS-ADTS] section 7.1.1.4.) There is no constraint on the relative distinguished name (RDN) value.
ntSecurityDescriptor	The default security descriptor for fpo objects; the **Owner** and **Group** field of the security descriptor value MUST be the Domain Admins SID from the domain in which the object is created.

2. If the **groupType** is GROUP_TYPE_SECURITY_ACCOUNT, all of the following constraints MUST be satisfied:

    1. If the domain mode is mixed, the member values MUST refer to user objects (or objects derived from user).

    2. If the domain mode is native, the member values MUST satisfy at least one of the following criteria:

        1. The member value refers to a user account.

        2. The member value refers to a group account whose **groupType** is GROUP_TYPE_SECURITY_ACCOUNT.

3. If the **groupType** is GROUP_TYPE_SECURITY_RESOURCE, all of the following constraints MUST be satisfied:

    1. If the domain mode is mixed, the member values MUST either refer to user objects (or objects derived from user), or refer to group objects whose **groupType** is GROUP_TYPE_SECURITY_ACCOUNT.

    2. If the domain mode is native, the constraint shown above is relaxed to include member values that refer to group objects whose **groupType** is GROUP_TYPE_SECURITY_RESOURCE.

4. If the **groupType** contains the GROUP_TYPE_UNIVERSAL_GROUP, each member value MUST satisfy at least one of the following conditions:

    1. The value refers to a user object (or an object derived from user).

    2. The value refers to a group object (or an object derived from group) with a **groupType** attribute that contains GROUP_TYPE_ACCOUNT_GROUP or GROUP_TYPE_UNIVERSAL_GROUP.

3.2.1.8.10 userAccountControl

1. If the UF_LOCKOUT bit (section 2.2.1.13) is set and the **lockoutTime** attribute is nonzero, the **lockoutTime** attribute MUST be updated to a value of zero.

2. The following bits, if set, MUST be silently unset before committing the transaction: UF_LOCKOUT and UF_PASSWORD_EXPIRED.

3. If the UF_SERVER_TRUST_ACCOUNT bit is set, all of the following constraints MUST be satisfied:

1. The **primaryGroupId** attribute MUST be updated to the value DOMAIN_GROUP_RID_CONTROLLERS.

2. If the previous **primaryGroupId** value is NOT DOMAIN_GROUP_RID_COMPUTERS, let G be the group whose **objectSid** value has the RID of the previous **primaryGroupId** on the current object. G's member attribute MUST be updated to add a reference to the current object if not already present; processing errors for this constraint MUST be ignored.

4. If either UF_TRUSTED_TO_AUTHENTICATE_FOR_DELEGATION or UF_TRUSTED_FOR_DELEGATION is set, the requester MUST have the SE_ENABLE_DELEGATION_NAME privilege (specified in the Windows Behavior note of [MS-LSAD] section 3.1.1.2.1). Abort processing and return STATUS_ACCESS_DENIED on error.

5. If any of the following bits are set, the requester MUST have the associated Control Access Right on the **ntSecurityDescriptor** for the account domain object, per an access check. (Information about the access check mechanism is specified in [MS-ADTS] section 5.1.3.3.) If this constraint fails, the responder MUST abort processing and return STATUS_ACCESS_DENIED.

userAccountControlBit	Required Control Access Right
UF_PASSWD_NOTREQD	280f369c-67c7-438e-ae98-1d46f3c6f541
UF_DONT_EXPIRE_PASSWD	ccc2dc7d-a6ad-4a7a-8846-c04e3cc53501
UF_ENCRYPTED_TEXT_PASSWORD_ALLOWED	05c74c5e-4deb-43b4-bd9f-86664c2a7fd5
UF_SERVER_TRUST_ACCOUNT	9923a32a-3607-11d2-b9be-0000f87a36b2

6. If the UF_SMARTCARD_REQUIRED bit is set and is NOT present in the previous value, the **dbcsPwd** and **unicodePwd** attributes MUST be updated with 16 bytes of random bytes, and the **supplementalCredentials** attribute MUST be removed.

7. If the UF_PASSWD_NOTREQD bit is removed from the **userAccountControl** value, the responder MUST abort processing and return an error status if all of the following conditions are true:

   1. **userAccountControl** contains UF_NORMAL_ACCOUNT.

   2. **userAccountControl** does not contain the UF_ACCOUNTDISABLE.

   3. The Effective-MinimumPasswordLength attribute (see section 3.2.1.5) is nonzero.

<33>

3.2.1.8.11 supplementalCredentials

The **supplementalCredentials** attribute is a structured binary value that contains additional cryptographic forms of the cleartext password (and optionally the cleartext password itself) that are stored as property-value pairs.

The structure of **supplementalCredentials** is a **USER_PROPERTIES (section 2.2.10.1)** structure, followed by a variable number of **USER_PROPERTY (section 2.2.10.2)** structures.

When **supplementalCredentials** is updated with a value (which is interpreted as a UTF-16 encoded cleartext password) as a result of a trigger, this value is not stored directly; instead it is processed and the result is stored in **supplementalCredentials** as specified in this section.

Each property name is a UTF-16 encoded string; each value has its own unique binary format. Four properties are in **supplementalCredentials** and listed in this table:

Property name (normative)	Property value semantic	Property value format specification section
Packages	A list of the credential types that are stored as properties in **supplementalCredentials**.	3.2.1.8.11.2
Primary:WDigest	Cryptographic hashes of the cleartext password for the Digest authentication protocol.	3.2.1.8.11.3
Primary:Kerberos	Cryptographic hashes of the cleartext password for the Kerberos authentication protocol.	3.2.1.8.11.4
Primary:CLEARTEXT	The cleartext password.	3.2.1.8.11.5

3.2.1.8.11.1 Processing

Section 3.2.1.8.11.1.1 describes how to update the **USER_PROPERTIES** structure when properties are added or removed.

Section 3.2.1.8.11.1.2 describes how to update a **USER_PROPERTY** structure given a property-value pair.

3.2.1.8.11.1.1 USER_PROPERTIES Processing

When a new property-value pair is added (as a result of an update, for example), the **PropertyCount** field of the **USER_PROPERTIES** structure MUST be incremented by one, and the property structure (a **USER_PROPERTY** structure) MUST be added to the variable-length array of **USER_PROPERTY** structures that follow **USER_PROPERTIES**. The order of the USER_PROPERTY entries is not important.

When a property-value pair is removed and the property-value is present in the **USER_PROPERTIES** structure, the **PropertyCount** field of the **USER_PROPERTIES** structure MUST be decremented by one, and the property structure (a **USER_PROPERTY** structure) MUST be removed from the variable-length array of **USER_PROPERTY** structures that follow USER_PROPERTIES.

If the property-value is not present on removal, then no change to **USER_PROPERTIES** is required.

3.2.1.8.11.1.2 USER_PROPERTY Processing

This section describes how to structure a given property-value pair in a **USER_PROPERTY** structure.

- The **PropertyLength** field MUST be set to the size, in bytes, of the property name. The property name "WDigest", for example, has a PropertyLength of 14.
- The **ValueLength** field MUST be set to the size, in bytes, of the value of the property after hexadecimal-encoding the value per the specification in section 2.2.10.2.
- The property name MUST follow the **Reserved** field of the **USER_PROPERTY** structure.
- The hex-encoded value MUST follow the property name.

3.2.1.8.11.2 Packages Property

The property value is a UTF-16 encoded string. The string itself is composed of a set of sub-strings separated by a NULL Unicode character value, as defined in [UNICODE3.1]. The final character does not need to be a NULL Unicode character. Each substring is the name of a credential type stored as a property in the supplementalCredentials value.

When an update occurs, if a credential-type property (that is, a property that represents a credential-type) is successfully computed, this value MUST be updated with the associated credential name. The following table shows the legal values of names to be used as strings in the property value of the "Packages" property along with their associated credential type.

Credential type property	Name
Primary:WDigest	WDigest
Primary:Kerberos	Kerberos
Primary:CLEARTEXT	CLEARTEXT

3.2.1.8.11.3 Primary:WDigest Property

The WDigest property contains pre-calculated hash forms that are used in the digest authentication protocols ([RFC2617]). A normative description of the hashes used by the protocol is specified in [RFC2617] section 3.2.2.2.

When an update to supplementalCredentials occurs, the responder MUST create a WDIGEST_CREDENTIALS-structured value using the hash-computation mechanisms in section 3.2.1.8.11.3.1. This value MUST then be placed in a **USER_PROPERTY** structure along with the property name "Primary:WDigest". Finally, the resulting USER_PROPERTY-structured value MUST be added to the list of properties within **supplementalCredentials** per section 3.2.1.8.11.1.1.

3.2.1.8.11.3.1 WDIGEST_CREDENTIALS Construction

The following notation is used to describe how the hash values value are constructed. All strings are converted from UTF-16 encoding to ISO 8859-1 Latin I code page ([MSFT-LATIN1]) prior to the hashing.

Notation	Description
MD5(x, y, z)	An MD5 hash of the values x, y, z in that order.
MD5(x, y)	An MD5 hash of the values x, y in that order.
UPPER(x)	The uppercase version of the string per the Unicode standard ([UNICODE3.1]).
LOWER(x)	The lowercase version of the string per the Unicode standard ([UNICODE3.1]).
sAMAccountName	The **sAMAccountName** attribute value.
NETBIOSDomainName	The name attribute of the account domain object.
DNSDomainName	The DNS name of the domain.

Hash1: MD5(sAMAccountName, NETBIOSDomainName, password)

Hash2: MD5(UPPER(sAMAccountName), UPPER(NETBIOSDomainName), password)

Hash3: MD5(LOWER(sAMAccountName), LOWER(NETBIOSDomainName), password)

Hash4: MD5(sAMAccountName, UPPER(NETBIOSDomainName), password)

Hash5: MD5(sAMAccountName, LOWER(NETBIOSDomainName), password)

Hash6: MD5(UPPER(sAMAccountName), LOWER(NETBIOSDomainName), password)

Hash7: MD5(LOWER(sAMAccountName), UPPER(NETBIOSDomainName), password)

Hash8: MD5(sAMAccountName, DNSDomainName, password)

Hash9: MD5(UPPER(sAMAccountName), UPPER(DNSDomainName), password)

Hash10: MD5(LOWER(sAMAccountName), LOWER(DNSDomainName), password)

Hash11: MD5(sAMAccountName, UPPER(DNSDomainName), password)

Hash12: MD5(sAMAccountName, LOWER(DNSDomainName), password)

Hash13: MD5(UPPER(sAMAccountName), LOWER(DNSDomainName), password)

Hash14: MD5(LOWER(sAMAccountName), UPPER(DNSDomainName), password)

Hash15: MD5(userPrincipalName, password)

Hash16: MD5(UPPER(userPrincipalName), password)

Hash17: MD5(LOWER(userPrincipalName), password)

Hash18: MD5(NETBIOSDomainName\sAMAccountName, password)

Hash19: MD5(UPPER(NETBIOSDomainName\sAMAccountName), password)

Hash20: MD5(LOWER(NETBIOSDomainName\sAMAccountName), password)

Hash21: MD5(sAMAccountName, "Digest", password)

Hash22: MD5(UPPER(sAMAccountName), "DIGEST", password)

Hash23: MD5(LOWER(sAMAccountName), "digest", password)

Hash24: MD5(userPrincipalName, "Digest", password)

Hash25: MD5(UPPER(userPrincipalName), "DIGEST", password)

Hash26: MD5(LOWER(userPrincipalName), "digest", password)

Hash27: MD5(NETBIOSDomainName\sAMAccountName, "Digest", password)

Hash28: MD5(UPPER(NETBIOSDomainName\sAMAccountName), "DIGEST", password)

Hash29: MD5(LOWER(NETBIOSDomainName\sAMAccountName), "digest", password)

3.2.1.8.11.4 Primary:Kerberos Property

When an update to **supplementalCredentials** occurs, the responder MUST create a KERB_STORED_CREDENTIAL-structured value as specified below. This value MUST then be placed in a **USER_PROPERTY** structure along with the property name "Primary:Kerberos". Finally, the resulting USER_PROPERTY-structured value MUST be added to the list of properties within **supplementalCredentials** per section 3.2.1.8.11.1.1.

KERB_STORED_CREDENTIAL is a variable-length structure starting with a **KERB_STORED_CREDENTIAL** structure, followed by two or four **KERB_KEY_DATA** structures, followed by a salt value and two or four key values. The salt and key values are referenced from the **KERB_STORED_CREDENTIAL** and **KERB_KEY_DATA** structures.

The following diagram depicts the layout. The first eight fields are from the **KERB_STORED_CREDENTIAL** structure.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Revision																Flags															
CredentialCount																OldCredentialCount															
DefaultSaltLength																DefaultSaltMaximumLength															
DefaultSaltOffset																Padding															
KERB_KEY_DATA (20 bytes)																															
KERB_KEY_DATA (20 bytes)																															
KERB_KEY_DATA (20 bytes)(optional)																															
KERB_KEY_DATA (20 bytes)(optional)																															
DefaultSalt value (variable length)																															
Key1 value (variable length)																															
Key2 value (variable length)																															
Key3 value (variable length) (optional)																															
Key4 value (variable length) (optional)																															

Revision, Flags, DefaultSaltLength, DefaultSaltMaximumLength, and DefaultSaltOffset MUST be set per the specification in section 2.2.10.4. DefaultSaltOffset, for example, is the offset of the "DefaultSalt value" section from the start of the Revision field in the preceding diagram.<34>

The responder MUST calculate two hash forms of the cleartext password, as specified in [RFC3961] sections 6.2.1 and 6.2.3. Call these values Key1 and Key2.

The first two **KERB_KEY_DATA** MUST be set to hold Key1 and Key2. Key1 and Key2 MUST be added to the end of the structure (as depicted above).

If there are existing **KERB_KEY_DATA** elements in the property prior to the current update, these elements MUST be copied into the third and fourth **KERB_KEY_DATA** elements. Call the associated key values of these **KERB_KEY_DATA** structures Key3 and Key4. Key3 and Key4 MUST be added to the end of the structure (as depicted above).

If there are no existing **KERB_KEY_DATA** elements in the property prior to the current update, the resulting **KERB_STORED_CREDENTIAL** in the third and fourth optional **KERB_KEY_DATA**

elements are excluded from the resulting value (and Key3 and Key4, in the preceding diagram, are also excluded).

3.2.1.8.11.5 Primary:CLEARTEXT Property

This credential type is the cleartext password. The value format is the UTF-16 encoded cleartext password.

Storage of the cleartext password for an object is configured when the Effective-PasswordReversibleEncryptionEnabled value (section 3.2.1.5) is set, or the current object's **userAccountControl** contains the USER_ENCRYPTED_TEXT_PASSWORD_ALLOWED bit.

If during a **clearTextPassword** attribute update, there is a Primary:CLEARTEXT property present in **supplementalCredentials** and storage of the cleartext password is not configured, the Primary:CLEARTEXT property MUST be removed, and the Packages property within **supplementalCredentials** MUST be updated to not contain the "CLEARTEXT" string.

If during a password set or change operation, there is a Primary:CLEARTEXT property present in **supplementalCredentials** and storage of the cleartext password is configured, the Primary:CLEARTEXT property MUST be updated (or added if not present), and the Packages property with **supplementalCredentials** MUST be updated to contain the "CLEARTEXT" string, if not already present.

3.2.1.9 Additional Update Triggers

The following triggers are referenced from other constraints or triggers.

3.2.1.9.1 Password History Update

The following constraints MUST be satisfied for ntPwdHistory and lmPwdHistory. The term "history attribute" refers to one or the other in the following constraints, and the term "associated password" refers to dbcsPwd when the history attribute is lmPwdHistory, and unicodePwd when the history attribute is ntPwdHistory.

Let Password-History-Length be the value of the Effective-PasswordHistoryLength attribute (see section 3.2.1.5). If the target object being updated is the krbtgt account (that is, the objectSid value has the RID value of DOMAIN_USER_RID_KRBTGT), and Password-History-Length is less than 3, the value of 3 MUST be used for Password-History-Length.

1. If the Password-History-Length is greater than 0 and the history attribute is zero length, the history attribute MUST be updated with the previous associated password if the old associated password's length is nonzero.

2. If the Password-History-Length is zero, the history attribute MUST be updated with a zero-length value.

3. If the Password-History-Length is nonzero, the associated password value MUST be placed at the beginning of the history attribute, and existing values MUST be shifted by 16 bytes to the right. If the size of the attribute exceeds Password-History-Length * 16, the attribute value MUST be truncated to not exceed Password-History-Length * 16 bytes.

3.2.1.9.2 objectSid Value Generation

This section is referenced by object creation triggers to update the **objectSid** attribute with a SID value. The SID value is generated by first generating a 32-bit unsigned integer value (the RID) and then concatenating that value with the **account domain SID**.

The key part of this section is how the RID is generated, because it must be unique for all time and space for a given domain. For all algorithms, once the RID is generated, the SID value is generated as specified in the previous sentence, and the **objectSid** attribute is updated with that value.

The simplest RID-generation algorithm is to maintain a counter and increment the counter for each RID that is issued. This algorithm is entirely sufficient for the non–domain controller case for this protocol. In a distributed environment, where any domain controller may be creating a security principal and therefore needs to assign a RID to that principal, the algorithm becomes more complicated. Many schemes are possible, up to and including a distributed counter, as described in [LAMPORT].

The RID-generation algorithm is different between a DC and non-DC configuration.

The following specifications present the constraints that must be satisfied when generating a RID. Generating RIDs in a monotonically increasing manner when possible (in addition to satisfying the constraints) is one implementation choice, but is not required.

3.2.1.9.2.1 DC Configuration

The following steps are used to generate a unique RID on a DC configuration.

1. The responder MUST generate a 32-bit integer value subject to all of the following constraints:

    1. The value MUST be within the range of the **RIDAllocationPool** attribute value, called the Rid-Range, as stored on the object referenced in the **RIDSetReferences** attribute as stored on the configured computer object for the host responder. The lower bound of the Rid-Range is the first 32-bit integer (in little-endian byte order) of the **RIDAllocationPool** attribute value. The upper bound of the Rid-Range is the second 32-bit integer (in little-endian byte order).

    2. Any value chosen from the Rid-Range that is used for an **objectSid** value that is successfully committed in a transaction MUST NOT ever be used again for **objectSid** generation within the current domain.

2. If the constraints in step 1 cannot be satisfied because the **RIDAllocationPool** attribute does not exist, or because all possible RIDs within the Rid-Range have been consumed, the responder MAY obtain a (new) value for **RIDAllocationPool** and attempt to generate a 32-bit value per the constraints in step 1. The responder MAY also return an error code if the constraints in step 1 cannot be satisfied.<35>

3.2.1.9.2.2 Non-DC Configuration

The following steps are used to generate a unique RID on a non-DC configuration.

1. The responder MUST generate a 32-bit integer value subject to all of the following constraints:

    1. The value MUST be greater than 1000.

    2. Any value chosen by this algorithm that is successfully committed in a transaction MUST NOT ever be used again for **objectSid** generation within the current domain.

2. If the constraints in step 1 cannot be satisfied, the responder MUST abort processing and return an error status.

3.2.2 Security Model

For methods that accept a context handle, the security model is a handle-based security model. A requester obtains a handle with a requester-specified access for that handle. The handle can then be used for operations that require the granted access. The access is encoded in a 32-bit value (an access mask). Note that some methods MUST enforce additional security requirements based on the input.

The security model assumes that whenever a context handle is presented to a method, the identity of the requester is the same as the identity of the requester that originally opened the handle.<36>

3.2.2.1 Standard Handle-Based Access Checks

The following tables specify the required access for the RPC methods that enforce required access on a handle parameter.

- **SamrCloseHandle**

Information level	Required access
N/A	None checked

- **SamrLookupDomainInSamServer**

Information level	Required access
N/A	SAM_SERVER_LOOKUP_DOMAIN

- **SamrEnumerateDomainsInSamServer**

Information level	Required access
N/A	SAM_SERVER_ENUMERATE_DOMAINS

- **SamrOpenDomain**

Information level	Required access
N/A	SAM_SERVER_LOOKUP_DOMAIN

- **SamrQueryInformationDomain**

Information level	Required access
DomainPasswordInformation	DOMAIN_READ_PASSWORD_PARAMETERS
DomainLockoutInformation:	DOMAIN_READ_PASSWORD_PARAMETERS
DomainGeneralInformation	DOMAIN_READ_OTHER_PARAMETERS
DomainLogoffInformation	DOMAIN_READ_OTHER_PARAMETERS
DomainOemInformation	DOMAIN_READ_OTHER_PARAMETERS
DomainNameInformation	DOMAIN_READ_OTHER_PARAMETERS

Information level	Required access
DomainServerRoleInformation	DOMAIN_READ_OTHER_PARAMETERS
DomainReplicationInformation	DOMAIN_READ_OTHER_PARAMETERS
DomainModifiedInformation	DOMAIN_READ_OTHER_PARAMETERS
DomainStateInformation	DOMAIN_READ_OTHER_PARAMETERS
DomainUasInformation	DOMAIN_READ_OTHER_PARAMETERS
DomainModifiedInformation2	DOMAIN_READ_OTHER_PARAMETERS
DomainGeneralInformation2	DOMAIN_READ_PASSWORD_PARAMETERS | DOMAIN_READ_OTHER_PARAMETERS

- **SamrSetInformationDomain**

Information level	Required access
DomainPasswordInformation	DOMAIN_WRITE_PASSWORD_PARAMS
DomainLockoutInformation	DOMAIN_WRITE_PASSWORD_PARAMS
DomainLogoffInformation	DOMAIN_WRITE_OTHER_PARAMETERS
DomainOemInformation	DOMAIN_WRITE_OTHER_PARAMETERS
DomainUasInformation	DOMAIN_WRITE_OTHER_PARAMETERS
DomainReplicationInformation	DOMAIN_ADMINISTER_SERVER
DomainStateInformation	DOMAIN_ADMINISTER_SERVER
DomainServerRoleInformation	DOMAIN_ADMINISTER_SERVER

- **SamrCreateGroupInDomain**

Information level	Required access
N/A	DOMAIN_CREATE_GROUP

- **SamrEnumerateGroupsInDomain**

Information level	Required access
N/A	DOMAIN_LIST_ACCOUNTS

- **SamrEnumerateUsersInDomain**

Information level	Required access
N/A	DOMAIN_LIST_ACCOUNTS

- **SamrCreateAliasInDomain**

Information level	Required access
N/A	DOMAIN_CREATE_ALIAS

- **SamrEnumerateAliasesInDomain**

Information level	Required access
N/A	DOMAIN_LIST_ACCOUNTS

- **SamrGetAliasMembership**

Information level	Required access
N/A	DOMAIN_GET_ALIAS_MEMBERSHIP

- **SamrLookupNamesInDomain**

Information level	Required access
N/A	DOMAIN_LOOKUP

- **SamrLookupIdsInDomain**

Information level	Required access
N/A	DOMAIN_LOOKUP

- **SamrOpenGroup**

Information level	Required access
N/A	DOMAIN_LOOKUP

- **SamrQueryInformationGroup**

Information level	Required access
GroupGeneralInformation	GROUP_READ_INFORMATION
GroupNameInformation	GROUP_READ_INFORMATION
GroupAttributeInformation	GROUP_READ_INFORMATION
GroupAdminCommentInformation	GROUP_READ_INFORMATION
GroupReplicationInformation	GROUP_READ_INFORMATION

- **SamrSetInformationGroup**

Information level	Required access
GroupNameInformation	GROUP_WRITE_ACCOUNT

Information level	Required access
GroupAttributeInformation	GROUP_WRITE_ACCOUNT
GroupAdminCommentInformation	GROUP_WRITE_ACCOUNT

- **SamrAddMemberToGroup**

Information level	Required access
N/A	GROUP_ADD_MEMBER

- **SamrDeleteGroup**

Information level	Required access
N/A	DELETE

- **SamrRemoveMemberFromGroup**

Information level	Required access
N/A	GROUP_REMOVE_MEMBER

- **SamrGetMembersInGroup**

Information level	Required access
N/A	GROUP_LIST_MEMBERS

- **SamrSetMemberAttributesOfGroup**

Information level	Required access
N/A	GROUP_ADD_MEMBER

- **SamrOpenAlias**

Information level	Required access
N/A	DOMAIN_LOOKUP

- **SamrQueryInformationAlias**

Information level	Required access
AliasGeneralInformation	ALIAS_READ_INFORMATION
AliasNameInformation	ALIAS_READ_INFORMATION
AliasAdminCommentInformation	ALIAS_READ_INFORMATION
AliasReplicationInformation	ALIAS_READ_INFORMATION

- **SamrSetInformationAlias**

Information level	Required access
AliasNameInformation	ALIAS_WRITE_ACCOUNT
AliasAdminCommentInformation	ALIAS_WRITE_ACCOUNT

- **SamrDeleteAlias**

Information level	Required access
N/A	DELETE

- **SamrGetMembersInAlias**

Information level	Required access
N/A	ALIAS_LIST_MEMBERS

- **SamrOpenUser**

Information level	Required access
N/A	DOMAIN_LOOKUP

- **SamrDeleteUser**

Information level	Required access
N/A	DELETE

- **SamrChangePasswordUser**

Information level	Required access
N/A	USER_CHANGE_PASSWORD

- **SamrGetGroupsForUser**

Information level	Required access
N/A	USER_LIST_GROUPS

- **SamrQueryDisplayInformation3**

Information level	Required access
N/A	DOMAIN_LIST_ACCOUNTS

- **SamrGetDisplayEnumerationIndex2**

Information level	Required access
N/A	DOMAIN_LIST_ACCOUNTS

- **SamrRemoveMemberFromForeignDomain**

Information level	Required access
N/A	DOMAIN_LOOKUP

- **SamrAddMultipleMembersToAlias**

Information level	Required access
N/A	ALIAS_ADD_MEMBER

- **SamrRemoveMultipleMembersFromAlias**

Information level	Required access
N/A	ALIAS_REMOVE_MEMBER

- **SamrRidToSid**

Information level	Required access
N/A	None checked

3.2.2.2 AD Access Checks in DC Configuration

Unless otherwise specified, the create, update, delete, and read access checks enforced by the MS-ADTS data model (specified in [MS-ADTS] section 5.1.3) are not enforced during the message processing of this protocol.

3.2.3 Timers

This protocol does not introduce any timers. Information about any transport-level timers is specified in [MS-RPCE].

3.2.4 Initialization

This section covers the default users and groups that the responder MUST have and the default access control on the data manipulated by this protocol.

3.2.4.1 Default Access

Information about the default access control (expressed in the default security descriptor) on user, group, alias, domain, and **server** objects is specified in [MS-ADTS] section 3.1.1.2. This is significant because this responder MUST use the security descriptor from the [MS-ADTS] data model to determine if the requester has access to perform the requested operation. If, for example, a requester opens a domain object with **SamrOpenDomain** requesting DOMAIN_READ_PASSWORD_PROPERTIES, **SamrOpenDomain** uses the [MS-ADTS] data model security descriptor to determine if the requester has access to read password-related properties. For

more information related to this example, see the message processing section of **SamrOpenDomain** (section 3.2.5.1.5).

3.2.4.2 Default Accounts

The following accounts MUST be present in a responder's database.

Non-DC configuration, user accounts:

Name	Domain	Rid	userAccountControl	Revision introduced
Administrator	Account	500	UF_NORMAL_ACCOUNT | UF_DONT_EXPIRE_PASSWORD	Windows NT 4.0 and earlier
Guest	Account	501	UF_NORMAL_ACCOUNT | UF_ACCOUNTDISABLE | UF_DONT_EXPIRE_PASSWORD	Windows NT 4.0 and earlier

Non-DC configuration, alias accounts:

Name	Domain	Rid	Member	Revision introduced
Administrators	Built-in	544	Administrator	Windows NT 4.0 and earlier
Users	Built-in	545		Windows NT 4.0 and earlier
Guests	Built-in	546	Guest	Windows NT 4.0 and earlier
Power Users	Built-in	547		Windows NT 4.0 and earlier
Print Operators	Built-in	550		Windows NT 4.0 and earlier
Backup Operators	Built-in	551		Windows NT 4.0 and earlier
Replicator	Built-in	552		Windows NT 4.0 and earlier
Remote Desktop Users	Built-in	555		Windows XP
Network Configuration Operators	Built-in	556		Windows XP
Performance Monitor Users	Built-in	558		Windows Server 2003
Performance Log Users	Built-in	559		Windows Server 2003

Name	Domain	Rid	Member	Revision introduced
Distributed COM Users	Built-in	562		Windows Server 2003 SP1
IIS_IUSRS	Built-in	568	IUSR	Windows Vista
Cryptographic Operators	Built-in	569		Windows Vista
Event Log Readers	Built-in	573		Windows Vista

DC configuration, user accounts:

Name	Domain	Rid	userAccountControl	Revision introduced
Administrator	Account	500	UF_NORMAL_ACCOUNT | UF_DONT_EXPIRE_PASSWORD	Windows NT 4.0 and earlier
Guest	Account	501	UF_NORMAL_ACCOUNT | UF_ACCOUNTDISABLE | UF_DONT_EXPIRE_PASSWORD	Windows NT 4.0 and earlier
krbtgt	Account	502	UF_NORMAL_ACCOUNT | UF_ACCOUNTDISABLE	Windows 2000

DC configuration, universal group accounts (only on root-domain):

Name	Domain	Rid	Member	Revision introduced
Schema Admins	Account	518	Administrator	Windows 2000
Enterprise Admins	Account	519	Administrator	Windows 2000
Enterprise Read-only Domain Controllers	Account	498		Windows Server 2008

DC configuration, group accounts:

Name	Domain	Rid	Member	Revision introduced
Domain Admins	Account	512	Administrator	Windows NT 4.0 and earlier
Domain Users	Account	513		Windows NT 4.0 and earlier
Domain Guests	Account	514	Guest	Windows NT 4.0 and earlier
Domain Computers	Account	515		Windows NT 4.0 and earlier
Domain Controllers	Account	516		Windows NT 4.0 and earlier

Name	Domain	Rid	Member	Revision introduced
Group Policy Creator Owners	Account	520	Administrator	Windows XP
Read-only Domain Controllers	Account	521		Windows Server 2008

DC configuration, alias accounts:

Name	Domain	Rid	Member	Revision introduced
Administrators	Built-in	544	Domain Admins, Administrator, Enterprise Admins	Windows NT 4.0 and earlier
Users	Built-in	545	Domain Users	Windows NT 4.0 and earlier
Guests	Built-in	546	Domain Guests, Guest	Windows NT 4.0 and earlier
Account Operators	Built-in	548		Windows NT 4.0 and earlier
System Operators	Built-in	549		Windows NT 4.0 and earlier
Print Operators	Built-in	550		Windows NT 4.0 and earlier
Backup Operators	Built-in	551		Windows NT 4.0 and earlier
Replicator	Built-in	552		Windows NT 4.0 and earlier
Cert Publishers	Account	517		Windows 2000
RAS and IAS Servers	Account	553		Windows 2000
Pre-Windows 2000 Compatible Access	Built-in	554	Everyone, Anonymous Logon, Authenticated Users<37>	Windows 2000
Remote Desktop Users	Built-in	555		Windows Server 2003
Network Configuration Operators	Built-in	556		Windows Server 2003
Incoming Forest Trust Builders	Built-in	557		Windows Server 2003

Name	Domain	Rid	Member	Revision introduced
Performance Monitor Users	Built-in	558		Windows Server 2003
Performance Log Users	Built-in	559		Windows Server 2003
Windows Authorization Access Group	Built-in	560	Enterprise Domain Controllers	Windows Server 2003
Terminal Server License Servers	Built-in	561		Windows Server 2003
Distributed COM Users	Built-in	562		Windows Server 2003 SP1
IIS_IUSRS	Built-in	568	IUSR	Windows Vista
Cryptographic Operators	Built-in	569		Windows Vista
Allowed RODC Password Replication Group	Account	571		Windows Vista
Denied RODC Password Replication Group	Account	572	Group Policy Creator Owners, Domain Admins, Cert Publishers, Domain Controllers, Krbtgt, Enterprise Admins, Schema Admins, Read-only Domain Controllers	Windows Vista
Event Log Readers	Built-in	573		Windows Vista
Certificate Service DCOM Access	Built-in	574		Windows Vista SP1 and Windows Server 2008

3.2.5 Message Processing Events and Sequencing Rules

This section specifies the methods of the protocol along with their processing.

The return value space of all methods is the **NTSTATUS** type, specified in [MS-ERREF] section 2.3. Unless specifically called out, error codes are returned to the requester of the protocol and are not handled by any special processing at the requester; therefore, the exact error code is implementation-specific. Cases in which the requester may handle a specific error code are called out. The set of such error codes are found in section 2.2.1.15.

Methods in RPC Opnum Order

Method	Description
SamrConnect	Returns a handle to a server object. Opnum: 0
SamrCloseHandle	Closes any context handle obtained from this RPC interface. Opnum: 1
SamrSetSecurityObject	Sets the access control on a server, domain, user, group, or alias object. Opnum: 2
SamrQuerySecurityObject	Queries the access control on a server, domain, user, group, or alias object. Opnum: 3
Opnum4NotUsedOnWire	Reserved for local use. Opnum: 4
SamrLookupDomainInSamServer	Obtains the SID of a domain object. Opnum: 5
SamrEnumerateDomainsInSamServer	Obtains a listing of all domains hosted by the responder side. Opnum: 6
SamrOpenDomain	Obtains a handle to a domain object. Opnum: 7
SamrQueryInformationDomain	Obtains attributes from a domain object. Opnum: 8
SamrSetInformationDomain	Updates attributes on a domain object. Opnum: 9
SamrCreateGroupInDomain	Creates a group object within a domain. Opnum: 10
SamrEnumerateGroupsInDomain	Enumerates all groups. Opnum: 11
SamrCreateUserInDomain	Creates a user. Opnum: 12
SamrEnumerateUsersInDomain	Enumerates all users. Opnum: 13
SamrCreateAliasInDomain	Creates an alias. Opnum: 14
SamrEnumerateAliasesInDomain	Enumerates all aliases. Opnum: 15
SamrGetAliasMembership	Obtains the union of all aliases of which a given set of

Method	Description
	SIDs are a member. Opnum: 16
SamrLookupNamesInDomain	Translates a set of account names into a set of RIDs. Opnum: 17
SamrLookupIdsInDomain	Translates a set of RIDs into account names. Opnum: 18
SamrOpenGroup	Obtains a handle to a group. Opnum: 19
SamrQueryInformationGroup	Obtains attributes from a group object. Opnum: 20
SamrSetInformationGroup	Updates attributes on a group object. Opnum: 21
SamrAddMemberToGroup	Adds a member to a group. Opnum: 22
SamrDeleteGroup	Removes a group object. Opnum: 23
SamrRemoveMemberFromGroup	Removes a member from a group. Opnum: 24
SamrGetMembersInGroup	Reads the members of a group. Opnum: 25
SamrSetMemberAttributesOfGroup	Sets the attributes of a member relationship. Opnum: 26
SamrOpenAlias	Obtains a handle to an alias. Opnum: 27
SamrQueryInformationAlias	Obtains attributes from an alias object. Opnum: 28
SamrSetInformationAlias	Updates attributes on an alias object. Opnum: 29
SamrDeleteAlias	Removes an alias object. Opnum: 30
SamrAddMemberToAlias	Adds a member to an alias. Opnum: 31
SamrRemoveMemberFromAlias	Removes a member from an alias. Opnum: 32
SamrGetMembersInAlias	Obtains the membership list of an alias. Opnum: 33

Method	Description
SamrOpenUser	Obtains a handle to a user. Opnum: 34
SamrDeleteUser	Removes a user object. Opnum: 35
SamrQueryInformationUser	Obtains attributes from a user object. Opnum: 36
SamrSetInformationUser	Updates attributes on a user object. Opnum: 37
SamrChangePasswordUser	Changes the password of a user object. Opnum: 38
SamrGetGroupsForUser	Obtains a list of groups of which a user is a member. Opnum: 39
SamrQueryDisplayInformation	Obtains a list of accounts in name-sorted order. Opnum: 40
SamrGetDisplayEnumerationIndex	Obtains an index into an account-name–sorted list of accounts. Opnum: 41
Opnum42NotUsedOnWire	Reserved for local use. Opnum: 42
Opnum43NotUsedOnWire	Reserved for local use. Opnum: 43
SamrGetUserDomainPasswordInformation	Obtains select password policy information. Opnum: 44
SamrRemoveMemberFromForeignDomain	Removes a member from all aliases. Opnum: 45
SamrQueryInformationDomain2	Obtains attributes from a domain object. Opnum: 46
SamrQueryInformationUser2	Obtains attributes from a user object. Opnum: 47
SamrQueryDisplayInformation2	Obtains a listing of accounts in name-sorted order. Opnum: 48
SamrGetDisplayEnumerationIndex2	Obtains an index into an account-name–sorted list of accounts. Opnum: 49
SamrCreateUser2InDomain	Creates a user. Opnum: 50

Method	Description
SamrQueryDisplayInformation3	Obtains a listing of accounts in name-sorted order. Opnum: 51
SamrAddMultipleMembersToAlias	Adds multiple members to an alias. Opnum: 52
SamrRemoveMultipleMembersFromAlias	Removes multiple members from an alias. Opnum: 53
SamrOemChangePasswordUser2	Changes a user's password. Opnum: 54
SamrUnicodeChangePasswordUser2	Changes a user account's password. Opnum: 55
SamrGetDomainPasswordInformation	Obtains select password policy information, Opnum: 56
SamrConnect2	Obtains a handle to a server object. Opnum: 57
SamrSetInformationUser2	Updates attributes on a user object. Opnum: 58
Opnum59NotUsedOnWire	Reserved for local use. Opnum: 59
Opnum60NotUsedOnWire	Reserved for local use. Opnum: 60
Opnum61NotUsedOnWire	Reserved for local use. Opnum: 61
SamrConnect4	Obtains a handle to a server object. Opnum: 62
Opnum63NotUsedOnWire	Reserved for local use. Opnum: 63
SamrConnect5	Obtains a handle to a server object. Opnum: 64
SamrRidToSid	Obtains the SID of an account. Opnum: 65
SamrSetDSRMPassword	Sets a local recovery password. Opnum: 66
SamrValidatePassword	Validates an application password against the locally stored policy. Opnum: 67
Opnum68NotUsedOnWire	Reserved for local use.

Method	Description
	Opnum: 68
Opnum69NotUsedOnWire	Reserved for local use. Opnum: 69

In the table above, the phrase "Reserved for local use" means that the client MUST NOT send the opnum, and the server behavior is undefined<38> since it does not affect interoperability.

All methods MUST NOT throw exceptions.

3.2.5.1 Open Pattern

These methods enable a requester to obtain an RPC context handle to an existing object.

See section 1.7.2 for details on how to choose between SamrConnect variations.

On success, each of these methods returns a handle that references a database object in the responder's implementation.

For a description of the "open" pattern of methods, see section 1.3.

3.2.5.1.1 SamrConnect5 (Opnum 64)

The **SamrConnect5** method obtains a handle to a server object.

```
long SamrConnect5(
  [in, unique, string] PSAMPR_SERVER_NAME ServerName,
  [in] unsigned long DesiredAccess,
  [in] unsigned long InVersion,
  [in, switch_is(InVersion)] SAMPR_REVISION_INFO* InRevisionInfo,
  [out] unsigned long* OutVersion,
  [out, switch_is(*OutVersion)] SAMPR_REVISION_INFO* OutRevisionInfo,
  [out] SAMPR_HANDLE* ServerHandle
);
```

ServerName: The null-terminated NETBIOS name of the responder; this parameter MAY<39> be ignored on receipt.

DesiredAccess: An **ACCESS_MASK** that indicates the access requested for *ServerHandle* on output. For a listing of possible values, see section 2.2.1.3.

InVersion: Indicates which field of the *InRevisionInfo* union is used.

InRevisionInfo: Revision information. For details, see the definition of the **SAMPR_REVISION_INFO_V1** structure, which is contained in the **SAMPR_REVISION_INFO** union.

OutVersion: Indicates which field of the *OutRevisionInfo* union is used.

OutRevisionInfo: Revision information. For details, see the definition of the **SAMPR_REVISION_INFO_V1** structure, which is contained in the **SAMPR_REVISION_INFO** union.

ServerHandle: An RPC context handle, as specified in section 2.2.3.2.

Upon receiving this message, the responder MUST process the data from the message subject to the following constraints:

1. The responder MUST translate the following bits in *DesiredAccess* per the following table. Translate means to remove the "Incoming Bit" and replace with the "Translated Bits".

Incoming bit	Translated bits
GENERIC_READ	SAM_SERVER_READ
GENERIC_WRITE	SAM_SERVER_WRITE
GENERIC_EXECUTE	SAM_SERVER_EXECUTE
GENERIC_ALL	SAM_SERVER_ALL_ACCESS

2. Let S be the server object in the account domain.

3. Let GrantedAccess be the union of all bits in the *DesiredAccess* column in the following table, where the requester has the specified access (shown in the Access Mask column) on the **ntSecurityDescriptor** on S. [MS-ADTS] section 5.1.3.3.3 specifies how to determine the requester's access.

DesiredAccess	Access mask
SAM_SERVER_CONNECT	ACTRL_DS_READ_PROP
SAM_SERVER_SHUTDOWN	ACTRL_DS_WRITE_PROP
SAM_SERVER_INITIALIZE	ACTRL_DS_WRITE_PROP
SAM_SERVER_CREATE_DOMAIN	ACTRL_DS_WRITE_PROP
SAM_SERVER_ENUMERATE_DOMAINS	ACTRL_DS_READ_PROP
SAM_SERVER_LOOKUP_DOMAIN	ACTRL_DS_READ_PROP
ACCESS_SYSTEM_SECURITY	ACCESS_SYSTEM_SECURITY
WRITE_OWNER	WRITE_OWNER
WRITE_DAC	WRITE_DAC
DELETE	DELETE

4. If GrantedAccess is 0, the responder MUST return STATUS_ACCESS_DENIED.

5. If *DesiredAccess* contains the MAXIMUM_ALLOWED bit, the responder MUST create and return a context handle via GroupHandle with GrantedAccess associated with the responder-side implementation of the context handle.

6. If *DesiredAccess* does not contain the MAXIMUM_ALLOWED bit, the following constraint MUST be satisfied:

   - If *DesiredAccess* contains bits not in GrantedAccess, the responder MUST return STATUS_ACCESS_DENIED. Otherwise, the responder MUST create and return a context handle via GroupHandle with *DesiredAccess* associated with the responder-side implementation of the context handle.

7. The responder SHOULD set *OutVersion* to 1 and OutRevisionInfo.Revision to the current version of the responder. The remaining fields of *OutRevisionInfo* SHOULD be set to zero.<40>

8. If any processing error occurred, the responder MUST return that error. Otherwise, the responder MUST return STATUS_SUCCESS.

3.2.5.1.2 SamrConnect4 (Opnum 62)

The **SamrConnect4** method obtains a handle to a server object.

```
long SamrConnect4(
  [in, unique, string] PSAMPR_SERVER_NAME ServerName,
  [out] SAMPR_HANDLE* ServerHandle,
  [in] unsigned long ClientRevision,
  [in] unsigned long DesiredAccess
);
```

ServerName: The null-terminated NETBIOS name of the responder; this parameter MAY<41> be ignored on receipt.

ServerHandle: An RPC context handle, as specified in section 2.2.3.2.

ClientRevision: Indicates the revision (for this protocol) of the requester. See the **Revision** field of **SAMPR_REVISION_INFO_V1 (section 2.2.3.14)** for possible values.

DesiredAccess: An **ACCESS_MASK** that indicates the access requested for *ServerHandle* on output. See section 2.2.1.3 for a listing of possible values.

The responder MUST behave as with a call to **SamrConnect5** with the following parameter values:

Parameter name	Parameter value
ServerName	SamrConnect4.ServerName
DesiredAccess	SamrConnect4.DesiredAccess
InVersion	1
InRevisionInfo	SAMPR_REVISION_INFO_V1.Revision = {SamrConnect4.ClientRevision} SAMPR_REVISION_INFO_V1.SupportedFeatures = {0}
OutVersion	Output ignored
OutRevisionInfo	Output ignored
ServerHandle	SamrConnect4.ServerHandle

3.2.5.1.3 SamrConnect2 (Opnum 57)

The **SamrConnect2** method returns a handle to a server object.

```
long SamrConnect2(
  [in, unique, string] PSAMPR_SERVER_NAME ServerName,
  [out] SAMPR_HANDLE* ServerHandle,
  [in] unsigned long DesiredAccess
```

```
);
```

ServerName: The null-terminated NETBIOS name of the responder; this parameter MAY<42> be ignored on receipt.

ServerHandle: An RPC context handle, as specified in section 2.2.3.2.

DesiredAccess: An **ACCESS_MASK** that indicates the access requested for *ServerHandle* on output. See section 2.2.1.3 for a listing of possible values.

The responder MUST behave as with a call to **SamrConnect5** with the following parameter values:

Parameter name	Parameter value
ServerName	SamrConnect2.ServerName
DesiredAccess	SamrConnect2.DesiredAccess
InVersion	1
InRevisionInfo	SAMPR_REVISION_INFO_V1.Revision = {0} SAMPR_REVISION_INFO_V1.SupportedFeatures = {0}
OutVersion	Output ignored
OutRevisionInfo	Output ignored
ServerHandle	SamrConnect2.ServerHandle

3.2.5.1.4 SamrConnect (Opnum 0)

The **SamrConnect** method returns a handle to a server object.

```
long SamrConnect(
  [in, unique] PSAMPR_SERVER_NAME ServerName,
  [out] SAMPR_HANDLE* ServerHandle,
  [in] unsigned long DesiredAccess
);
```

ServerName: The first character of the NETBIOS name of the responder; this parameter MAY<43> be ignored on receipt.

ServerHandle: An RPC context handle, as specified in section 2.2.3.2.

DesiredAccess: An **ACCESS_MASK** that indicates the access requested for *ServerHandle* upon output. See section 2.2.1.3 for a listing of possible values.

The responder MUST behave as with a call to **SamrConnect5** with the following parameter values:

Parameter name	Parameter value
ServerName	SamrConnect.ServerName
DesiredAccess	SamrConnect.DesiredAccess

Parameter name	Parameter value
InVersion	1
InRevisionInfo	SAMPR_REVISION_INFO_V1.Revision = {0} SAMPR_REVISION_INFO_V1.SupportedFeatures = {0}
OutVersion	Output ignored
OutRevisionInfo	Output ignored
ServerHandle	SamrConnect.ServerHandle

3.2.5.1.5 SamrOpenDomain (Opnum 7)

The **SamrOpenDomain** method obtains a handle to a domain object, given a SID.

```
long SamrOpenDomain(
  [in] SAMPR_HANDLE ServerHandle,
  [in] unsigned long DesiredAccess,
  [in] PRPC_SID DomainId,
  [out] SAMPR_HANDLE* DomainHandle
);
```

ServerHandle: An RPC context handle, as specified in section 2.2.3.2, representing a server object.

DesiredAccess: An **ACCESS_MASK**. See section 2.2.1.4 for a list of domain access values.

DomainId: A SID value of a domain hosted by the responder side of this protocol.

DomainHandle: An RPC context handle, as specified in section 2.2.3.2.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

Upon receiving this message, the responder MUST process the data from the message subject to the following constraints:

1. *DomainHandle* MUST have the required access specified in section 3.2.2.1.
2. The responder MUST translate the following bits in *DesiredAccess* per the following table. Translate means to remove the "Incoming bit" and replace with the "Translated bits" as follows:

Incoming bit	Translated bits
GENERIC_READ	DOMAIN_READ
GENERIC_WRITE	DOMAIN_WRITE
GENERIC_EXECUTE	DOMAIN_EXECUTE
GENERIC_ALL	DOMAIN_ALL_ACCESS

3. Let D be the domain object whose objectSid is *DomainId*. If no such object exists, the responder MUST return an error code.

4. Let GrantedAccess be the union of all bits in the "DesiredAccess" column in the following table where the requester has the specified access (shown in the "Access mask" column) on the **ntSecurityDescriptor** on D. A missing value in the "Object ACE type" column means that the access mask applies to the entire object. [MS-ADTS] section 5.1.3.3.3 specifies how to determine the requester's access.

DesiredAccess	Access mask	Object ACE type
DOMAIN_READ_PASSWORD_PARAMETERS	ACTRL_DS_READ_PROP	c7407360-20bf-11d0-a768-00aa006e0529
DOMAIN_WRITE_PASSWORD_PARAMS	ACTRL_DS_WRITE_PROP	c7407360-20bf-11d0-a768-00aa006e0529
DOMAIN_READ_OTHER_PARAMETERS	ACTRL_DS_READ_PROP	b8119fd0-04f6-4762-ab7a-4986c76b3f9a
DOMAIN_WRITE_OTHER_PARAMETERS	ACTRL_DS_WRITE_PROP	b8119fd0-04f6-4762-ab7a-4986c76b3f9a
DOMAIN_CREATE_USER	Always grant, if requested or MAXIMUM_ALLOWED present	
DOMAIN_CREATE_GROUP	Always grant, if requested or MAXIMUM_ALLOWED present	
DOMAIN_CREATE_ALIAS	Always grant, if requested or MAXIMUM_ALLOWED present	
DOMAIN_LIST_ACCOUNTS	ACTRL_DS_LIST	
DOMAIN_LOOKUP	ACTRL_DS_LIST	
DOMAIN_ADMINISTER_SERVER	ACTRL_DS_CONTROL_ACCESS	ab721a52-1e2f-11d0-9819-00aa0040529b
ACCESS_SYSTEM_SECURITY	ACCESS_SYSTEM_SECURITY	
WRITE_OWNER	WRITE_OWNER	
WRITE_DAC	WRITE_DAC	

DesiredAccess	Access mask	Object ACE type
DELETE	DELETE	

5. If GrantedAccess is 0, the responder MUST return STATUS_ACCESS_DENIED.

6. If *DesiredAccess* contains the MAXIMUM_ALLOWED bit, the responder MUST create and return a context handle via *DomainHandle* with GrantedAccess associated with the responder-side implementation of the context handle.

7. If *DesiredAccess* does not contain the MAXIMUM_ALLOWED bit, the following constraint MUST be satisfied:

   - If *DesiredAccess* contains bits that are not in GrantedAccess, the responder MUST return STATUS_ACCESS_DENIED. Otherwise, the responder MUST create and return a context handle via *DomainHandle* with *DesiredAccess* associated with the responder-side implementation of the context handle.

8. If any processing error occurred, the responder MUST return that error. Otherwise, the responder MUST return STATUS_SUCCESS to the requester.

3.2.5.1.6 Common Processing for Group, Alias, and User

This section specifies the message processing for **SamrOpenGroup (section 3.2.5.1.7)**, **SamrOpenAlias (section 3.2.5.1.8)**, and **SamrOpenUser (section 3.2.5.1.9)**. Each one of these methods specifies the following "input" parameters for this common processing:

- Target-Rid: A RID input parameter from the message.
- Target-Object-Type: The intended object type to be opened.
- Generic-Access-Mask-Mapping-Table: A mapping from a generic access (for example, GENERIC_READ) to a specific mapping (for example, DOMAIN_READ for domain objects).
- Desired-Access-Mapping-Table: A table that maps access masks specific to this protocol to object ACE values. An example access mask specific to this protocol is USER_READ (section 2.2.1.7).
- Output-Handle: An RPC context handle returned to the requester that represents the object that is requested to be opened.

Upon receiving this message, the responder MUST process the data from the message subject to the following constraints:

1. *DomainHandle* (the input parameter from the method) MUST have the required access specified in section 3.2.2.1.

2. The responder MUST translate the bits in *DesiredAccess* per the Generic-Access-Mask-Mapping-Table.

3. Let A be the database object, in the domain referenced by *DomainHandle*, whose **objectSid's** RID is Target-Rid, and whose database object type is Target-Object-Type. If no such object exists, the responder MUST return an error code.

4. Let *GrantedAccess* be the union of all bits in the "DesiredAccess" column in the Desired-Access-Mapping-Table, where the requester has the specified access (shown in the "Access mask"

column) on the **ntSecurityDescriptor** on Target-Object. A missing value in the "Object ACE type" column means that the access mask applies to the entire object. [MS-ADTS] section 5.1.3.3.3 specifies how to determine the requester's access.

5. If *GrantedAccess* is 0, the responder MUST return STATUS_ACCESS_DENIED.

6. If *DesiredAccess* contains the MAXIMUM_ALLOWED bit, the responder MUST create and return a context handle via Output-Handle with *GrantedAccess* associated with the responder-side implementation of the context handle.

7. If *DesiredAccess* does not contain the MAXIMUM_ALLOWED bit, the following constraint MUST be satisfied:

   - If *DesiredAccess* contains bits not in *GrantedAccess*, the responder MUST return STATUS_ACCESS_DENIED. Otherwise, the responder MUST create and return a context handle via Output-Handle with *DesiredAccess* associated with the responder-side implementation of the context handle.

8. If any processing error occurred, the responder MUST return that error. Otherwise, the responder MUST return STATUS_SUCCESS to the requester.

3.2.5.1.7 SamrOpenGroup (Opnum 19)

The **SamrOpenGroup** method obtains a handle to a group, given a RID.

```
long SamrOpenGroup(
  [in] SAMPR_HANDLE DomainHandle,
  [in] unsigned long DesiredAccess,
  [in] unsigned long GroupId,
  [out] SAMPR_HANDLE* GroupHandle
);
```

DomainHandle: An RPC context handle, as specified in section 2.2.3.2, representing a domain object.

DesiredAccess: An **ACCESS_MASK** that indicates the requested access for the returned handle. See section 2.2.1.5 for a list of group access values.

GroupId: A RID of a group.

GroupHandle: An RPC context handle, as specified in section 2.2.3.2.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

Upon receiving this message, the responder MUST process the data from the message per the constraints in section 3.2.5.1.6 with the following values:

- Target-Rid: *GroupId*
- Target-Object-Type: a group object (that is, a database with the objectClass group, or derived from group) and groupType containing GROUP_TYPE_ACCOUNT_GROUP or GROUP_TYPE_UNIVERSAL_GROUP.
- Generic-Access-Mask-Mapping-Table:

Incoming bit	Translated bits
GENERIC_READ	GROUP_READ
GENERIC_WRITE	GROUP_WRITE
GENERIC_EXECUTE	GROUP_EXECUTE
GENERIC_ALL	GROUP_ALL_ACCESS

- Desired-Access-Mapping-Table:

DesiredAccess	Access mask	Object ACE type
GROUP_READ_INFORMATION	ACTRL_DS_READ_PROP	59ba2f42-79a2-11d0-9020-00c04fc2d3cf
GROUP_WRITE_ACCOUNT	ACTRL_DS_WRITE_PROP	59ba2f42-79a2-11d0-9020-00c04fc2d3cf
GROUP_ADD_MEMBER	ACTRL_DS_WRITE_PROP	bf9679c0-0de6-11d0-a285-00aa003049e2
GROUP_REMOVE_MEMBER	ACTRL_DS_WRITE_PROP	bf9679c0-0de6-11d0-a285-00aa003049e2
GROUP_LIST_MEMBERS	ACTRL_DS_READ_PROP	bf9679c0-0de6-11d0-a285-00aa003049e2
ACCESS_SYSTEM_SECURITY	ACCESS_SYSTEM_SECURITY	
WRITE_OWNER	WRITE_OWNER	
WRITE_DAC	WRITE_DAC	
DELETE	DELETE	

- Output-Handle: *GroupHandle*

3.2.5.1.8 SamrOpenAlias (Opnum 27)

The **SamrOpenAlias** method obtains a handle to an alias, given a RID.

```
long SamrOpenAlias(
  [in] SAMPR_HANDLE DomainHandle,
  [in] unsigned long DesiredAccess,
  [in] unsigned long AliasId,
  [out] SAMPR_HANDLE* AliasHandle
);
```

DomainHandle: An RPC context handle, as specified in section 2.2.3.2, representing a domain object.

DesiredAccess: An **ACCESS_MASK** that indicates the requested access for the returned handle. See section 2.2.1.6 for a list of alias access values.

AliasId: A RID of an alias.

AliasHandle: An RPC context handle, as specified in section 2.2.3.2.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

Upon receiving this message, the responder MUST process the data from the message per the constraints in section 3.2.5.1.6 with the following values:

- Target-Rid: *AliasId*
- Target-Object-Type: A group object (that is, a database with the objectClass group, or derived from group) and groupType containing GROUP_TYPE_RESOURCE_GROUP.
- Generic-Access-Mask-Mapping-Table:

Incoming bit	Translated bits
GENERIC_READ	ALIAS_READ
GENERIC_WRITE	ALIAS_WRITE
GENERIC_EXECUTE	ALIAS_EXECUTE
GENERIC_ALL	ALIAS_ALL_ACCESS

- Desired-Access-Mapping-Table:

DesiredAccess	Access mask	Object ACE type
ALIAS_READ_INFORMATION	ACTRL_DS_READ_PROP	59ba2f42-79a2-11d0-9020-00c04fc2d3cf
ALIAS_WRITE_ACCOUNT	ACTRL_DS_WRITE_PROP	59ba2f42-79a2-11d0-9020-00c04fc2d3cf
ALIAS_ADD_MEMBER	ACTRL_DS_WRITE_PROP	bf9679c0-0de6-11d0-a285-00aa003049e2
ALIAS_REMOVE_MEMBER	ACTRL_DS_WRITE_PROP	bf9679c0-0de6-11d0-a285-00aa003049e2
ALIAS_LIST_MEMBERS	ACTRL_DS_READ_PROP	bf9679c0-0de6-11d0-a285-00aa003049e2
ACCESS_SYSTEM_SECURITY	ACCESS_SYSTEM_SECURITY	

DesiredAccess	Access mask	Object ACE type
WRITE_OWNER	WRITE_OWNER	
WRITE_DAC	WRITE_DAC	
DELETE	DELETE	

- Output-Handle: *AliasHandle*

3.2.5.1.9 SamrOpenUser (Opnum 34)

The **SamrOpenUser** method obtains a handle to a user, given a RID.

```
long SamrOpenUser(
  [in] SAMPR_HANDLE DomainHandle,
  [in] unsigned long DesiredAccess,
  [in] unsigned long UserId,
  [out] SAMPR_HANDLE* UserHandle
);
```

DomainHandle: An RPC context handle, as specified in section 2.2.3.2, representing a domain object.

DesiredAccess: An **ACCESS_MASK** that indicates the requested access for the returned handle. See section 2.2.1.7 for a list of user access values.

UserId: A RID of a user account.

UserHandle: An RPC context handle, as specified in section 2.2.3.2.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

Upon receiving this message, the responder MUST process the data from the message per the constraints in section 3.2.5.1.6 with the following values:

- Target-Rid: *UserId*
- Target-Object-Type: A user object (that is, a database with the objectClass user, or derived from user).
- Generic-Access-Mask-Mapping-Table:

Incoming bit	Translated bits
GENERIC_READ	USER_READ
GENERIC_WRITE	USER_WRITE
GENERIC_EXECUTE	USER_EXECUTE

Incoming bit	Translated bits
GENERIC_ALL	USER_ALL_ACCESS

- Desired-Access-Mapping-Table:

DesiredAccess	Access mask	Object ACE type
USER_READ_GENERAL	ACTRL_DS_READ_PROP	59ba2f42-79a2-11d0-9020-00c04fc2d3cf
USER_READ_PREFERENCES	ACTRL_DS_READ_PROP	59ba2f42-79a2-11d0-9020-00c04fc2d3cf
USER_READ_LOGON	ACTRL_DS_READ_PROP	5f202010-79a5-11d0-9020-00c04fc2d4cf
USER_READ_ACCOUNT	ACTRL_DS_READ_PROP	4c164200-20c0-11d0-a768-00aa006e0529
USER_WRITE_PREFERENCES	ACTRL_DS_WRITE_PROP	59ba2f42-79a2-11d0-9020-00c04fc2d3cf
USER_WRITE_ACCOUNT	ACTRL_DS_WRITE_PROP	59ba2f42-79a2-11d0-9020-00c04fc2d3cf
USER_WRITE_ACCOUNT	ACTRL_DS_WRITE_PROP	5f202010-79a5-11d0-9020-00c04fc2d4cf
USER_WRITE_ACCOUNT	ACTRL_DS_WRITE_PROP	4c164200-20c0-11d0-a768-00aa006e0529
USER_CHANGE_PASSWORD	ACTRL_DS_CONTROL_ACCESS	ab721a53-1e2f-11d0-9819-00aa0040529b
USER_FORCE_PASSWORD_CHANGE	ACTRL_DS_CONTROL_ACCESS	00299570-246d-11d0-a768-00aa006e0529
USER_LIST_GROUPS	ACTRL_DS_READ_PROP	bf967991-0de6-11d0-a285-00aa003049e2
USER_READ_GROUP_INFORMATION	ACTRL_DS_READ_PROP	
USER_WRITE_GROUP_INFORMATION	ACTRL_DS_WRITE_PROP	
ACCESS_SYSTEM_SECURITY	ACCESS_SYSTEM_SECURITY	
WRITE_OWNER	WRITE_OWNER	
WRITE_DAC	WRITE_DAC	

DesiredAccess	Access mask	Object ACE type
DELETE	DELETE	

- Output-Handle: *UserHandle*

3.2.5.2 Enumerate Pattern

These methods enable a requester to obtain a listing of all objects of a certain type. With the exception of **SamrEnumerateDomainsInSamServer**, which requires a server handle, these methods require a domain handle from the "open" pattern of methods (section 3.2.5.1).

For a description of the "enumerate" pattern of methods, see section 1.3.

3.2.5.2.1 SamrEnumerateDomainsInSamServer (Opnum 6)

The **SamrEnumerateDomainsInSamServer** method obtains a listing of all domains hosted by the responder side of this protocol.

```
long SamrEnumerateDomainsInSamServer(
  [in] SAMPR_HANDLE ServerHandle,
  [in, out] unsigned long* EnumerationContext,
  [out] PSAMPR_ENUMERATION_BUFFER* Buffer,
  [in] unsigned long PreferedMaximumLength,
  [out] unsigned long* CountReturned
);
```

ServerHandle: An RPC context handle, as specified in section 2.2.3.2, representing a server object.

EnumerationContext: This value is a cookie that the responder can use to continue an enumeration on a subsequent call. It is an opaque value to the requester.

Buffer: A listing of domain information, as described in section 2.2.3.9.

PreferedMaximumLength: The requested maximum number of bytes to return in *Buffer*.

CountReturned: The count of domain elements returned in *Buffer*.

This method asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

On receiving this message, the responder MUST process the data from the message subject to the following constraints:

1. *ServerHandle* MUST have the required access specified in section 3.2.2.1.
2. The responder MUST enable a requester to obtain a listing, without duplicates, of the following two values: the name attribute of the account domain object and the name attribute of the built-in domain object.

3. *EnumerationContext* MUST be used to allow the requester implementation to pass back to the responder, on a subsequent call, information on the last database object that was returned using *EnumerationContext*.<44>

4. Responders SHOULD<45> validate that *EnumerationContext* is an expected value for the responder's implementation.

5. The responder SHOULD<46> fill Buffer.Buffer with as many entries as possible, such that not more than *PreferedMaximumLength* bytes are returned in Buffer.Buffer. If the responder returns more than *PreferedMaximumLength* bytes, the difference between *PreferedMaximumLength* and the actual number of bytes returned MUST be less than the maximum size, in bytes, of one entry in the array Buffer.Buffer.

6. Each element of Buffer.Buffer MUST represent one database object that matches the criteria from item 2 above, and MUST be filled as follows:

    1. Buffer.Buffer.Name is the name attribute value of the database object.

    2. Buffer.Buffer.RelativeId is 0.

7. On output, *CountReturned* MUST equal Buffer.EntriesRead.

8. STATUS_MORE_ENTRIES MUST be returned if the responder returns less than all of the database objects in Buffer.Buffer because of the *PreferedMaximumLength* restriction described above. Note that this return value is not an error status.

9. If there are no entries or Buffer.Buffer contains all matching database objects that remain, the responder MUST return STATUS_SUCCESS.

3.2.5.2.2 Common Processing for Enumeration of Users, Groups, and Aliases

This section specifies message processing that is common for **SamrEnumerateGroupsInDomain**, **SamrEnumerateAliasesInDomain**, and **SamrEnumerateUsersInDomain**. The explanation of each method specifies a filter that is used to identify which objects to return to the requester; this filter is referred to in this section by the term Enumerate-Filter.

Let the term "session" refer to a set of sequential enumerate method calls made by a requester, starting with an *EnumerationContext* parameter of value 0 and ending with an enumerate method that returns STATUS_SUCCESS. The methods must be the same type; for example, a session is a sequence of **SamrEnumerateGroupsInDomain** method calls, not a **SamrEnumerateGroupsInDomain** method call followed by a **SamrEnumerateUsersInDomain** method call.

Finally, a non-normative description of *EnumerationContext* is helpful to understand the processing of this parameter. This parameter is used as a "cookie" by the responder in order to communicate to itself, between method calls within a session, which accounts have already been returned to the requester.

As an example, recall that *EnumerationContext* is a 32-bit value. Because of this fact, a possible choice of cookie could be the RID of the last account that was returned. Upon receiving a nonzero cookie, the responder can determine the next account that needs to be returned. Note that this example depends on the responder returning the accounts in RID sort order; however, this method has no constraint about sort order.

Upon receiving this message, the responder MUST process the data from the message, subject to the following constraints:

1. Domain MUST have the required access specified in section 3.2.2.1.

2. The responder MUST enable a requester to obtain a listing, without duplicates, of all database objects that satisfy the criteria of Enumerate-Filter.

3. The responder SHOULD<47> use *EnumerationContext* to allow the requester implementation to pass back to the responder, on a subsequent call, information on the last database object that was returned using *EnumerationContext*.

   If an object that satisfies Enumerate-Filter is added between successive Enumerate method calls in a session, and said object has a RID that is greater than the RIDs of all objects returned in previous calls, the responder MUST return said object before the Enumeration is complete.

   If an object that satisfies Enumerate-Filter is deleted between successive Enumerate method calls in a session, and said object has not already been returned by a previous method call in the session, the responder MUST NOT return said object before the Enumeration is complete.

4. The responder SHOULD<48> validate that *EnumerationContext* is an expected value for the responder's implementation.

5. The responder SHOULD<49> fill Buffer.Buffer with as many entries as possible, such that not more than *PreferedMaximumLength* bytes are returned in Buffer.Buffer. If the responder returns more than *PreferedMaximumLength* bytes, the difference between *PreferedMaximumLength* and the actual number of bytes returned MUST be less than the maximum size, in bytes, of one entry in the array Buffer.Buffer.

6. Each element of Buffer.Buffer MUST represent one database object that matches the Enumerate-Filter and MUST be set as follows:

   1. Buffer.Buffer.Name is the **sAMAccountName** attribute value of the database object.

   2. Buffer.Buffer.RelativeId is the RID of the **objectSid** attribute of the database object.

7. On output, *CountReturned* MUST equal Buffer.EntriesRead.

8. STATUS_MORE_ENTRIES MUST be returned if the responder returns less than all of the database objects in Buffer.Buffer because of the *PreferedMaximumLength* restriction described above. Note that this return value is not an error status.

9. If there are more database objects than are returned in Buffer.Buffer, the responder MUST return STATUS_MORE_ENTRIES. Note that this return value is not an error status.

10. If there are no entries, or if Buffer.Buffer contains all matching database objects that remain, the responder MUST return STATUS_SUCCESS.

3.2.5.2.3 SamrEnumerateGroupsInDomain (Opnum 11)

The **SamrEnumerateGroupsInDomain** method enumerates all groups.

```
long SamrEnumerateGroupsInDomain(
  [in] SAMPR_HANDLE DomainHandle,
  [in, out] unsigned long* EnumerationContext,
  [out] PSAMPR_ENUMERATION_BUFFER* Buffer,
  [in] unsigned long PreferedMaximumLength,
  [out] unsigned long* CountReturned
);
```

DomainHandle: An RPC context handle, as specified in section 2.2.3.2, representing a domain object.

EnumerationContext: This value is a cookie that the responder can use to continue an enumeration on a subsequent call. It is an opaque value to the requester.

Buffer: A list of group information, as specified in section 2.2.3.9.

PreferedMaximumLength: The requested maximum number of bytes to return in *Buffer*.

CountReturned: The count of domain elements returned in *Buffer*.

This method asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

This method MUST be processed per the specifications in section 3.2.5.2.2 using the following object selection filter:

1. The **objectClass** attribute value MUST be group or derived from group.
2. The **groupType** attribute value MUST be one of GROUP_TYPE_SECURITY_UNIVERSAL or GROUP_TYPE_SECURITY_ACCOUNT.
3. The **objectSid** attribute value MUST have the domain prefix of the domain referenced by *DomainHandle*.

3.2.5.2.4 SamrEnumerateAliasesInDomain (Opnum 15)

The **SamrEnumerateAliasesInDomain** method enumerates all aliases.

```
long SamrEnumerateAliasesInDomain(
  [in] SAMPR_HANDLE DomainHandle,
  [in, out] unsigned long* EnumerationContext,
  [out] PSAMPR_ENUMERATION_BUFFER* Buffer,
  [in] unsigned long PreferedMaximumLength,
  [out] unsigned long* CountReturned
);
```

DomainHandle: An RPC context handle, as specified in section 2.2.3.2, representing a domain object.

EnumerationContext: This value is a cookie that the responder can use to continue an enumeration on a subsequent call. It is an opaque value to the requester.

Buffer: A list of alias information, as specified in section 2.2.3.9.

PreferedMaximumLength: The requested maximum number of bytes to return in *Buffer*.

CountReturned: The count of domain elements returned in *Buffer*.

This method asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

This method MUST be processed per the specifications in section 3.2.5.2.2 using the following object selection filter:

1. The **objectClass** attribute value MUST be group or derived from group.
2. The **groupType** attribute value MUST be GROUP_TYPE_SECURITY_RESOURCE.
3. The **objectSid** attribute value MUST have the domain prefix of the domain referenced by *DomainHandle*.

3.2.5.2.5 SamrEnumerateUsersInDomain (Opnum 13)

The **SamrEnumerateUsersInDomain** method enumerates all users.

```
long SamrEnumerateUsersInDomain(
  [in] SAMPR_HANDLE DomainHandle,
  [in, out] unsigned long* EnumerationContext,
  [in] unsigned long UserAccountControl,
  [out] PSAMPR_ENUMERATION_BUFFER* Buffer,
  [in] unsigned long PreferedMaximumLength,
  [out] unsigned long* CountReturned
);
```

DomainHandle: An RPC context handle, as specified in section 2.2.3.2, representing a domain object.

EnumerationContext: This value is a cookie that the responder can use to continue an enumeration on a subsequent call. It is an opaque value to the requester.

UserAccountControl: A filter value to be used on the **userAccountControl** attribute.

Buffer: A list of user information, as specified in section 2.2.3.9.

PreferedMaximumLength: The requested maximum number of bytes to return in *Buffer*.

CountReturned: The count of domain elements returned in *Buffer*.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

This method MUST be processed per the specifications in section 3.2.5.2.2, using the following object selection filter:

1. The **objectClass** attribute value MUST be user or derived from user.
2. The **userAccountControl** attribute value MUST contain all the bits in the method parameter *UserAccountControl*.
3. The **objectSid** attribute value MUST have the domain prefix of the domain referenced by *DomainHandle*.

In addition, all of the following constraints MUST be satisfied before the constraints of section 3.2.5.2.2 are satisfied:

1. If *DomainHandle* is a reference to the account domain and the configuration is DC, the requester MUST have the 91d67418-0135-4acc-8d79-c08e857cfbec Control Access Right on the domain's **ntSecurityDescriptor** attribute value.

2. The responder MUST ignore the UF_LOCKOUT and UF_PASSWORD_EXPIRED bits in the *UserAccountControl* parameter.

3.2.5.3 Selective Enumerate Pattern

The requester use-pattern for these methods is a call to **SamrGetDisplayEnumerationIndex2**, followed by a call to **SamrQueryDisplayInformation3**, passing in the state returned by **SamrGetDisplayEnumerationIndex2**. This state is used as an index to indicate the account at which **SamrQueryDisplayInformation3** must start its enumeration.

These methods require a domain handle from the "open" pattern of methods (section 3.2.5.1).

See section 1.7.2 for details on how to choose between **SamrQueryDisplayInformation** and **SamrGetDisplayEnumerationIndex** variations.

See section 1.3 for a description of the "selective enumerate" pattern of methods.

3.2.5.3.1 SamrQueryDisplayInformation3 (Opnum 51)

The **SamrQueryDisplayInformation3** method obtains a listing of accounts in name-sorted order, starting at a specified index.

```
long SamrQueryDisplayInformation3(
  [in] SAMPR_HANDLE DomainHandle,
  [in] DOMAIN_DISPLAY_INFORMATION DisplayInformationClass,
  [in] unsigned long Index,
  [in] unsigned long EntryCount,
  [in] unsigned long PreferredMaximumLength,
  [out] unsigned long* TotalAvailable,
  [out] unsigned long* TotalReturned,
  [out, switch_is(DisplayInformationClass)]
    PSAMPR_DISPLAY_INFO_BUFFER Buffer
);
```

DomainHandle: An RPC context handle, as specified in section 2.2.3.2, representing a domain object.

DisplayInformationClass: An enumeration (see section 2.2.8.12) that indicates the type of accounts, as well as the type of attributes on the accounts, to return via the *Buffer* parameter.

Index: A cursor into an account-name–sorted list of accounts.

EntryCount: The number of accounts that the requester is requesting on output.

PreferredMaximumLength: The requested maximum number of bytes to return in *Buffer*; this value overrides *EntryCount* if this value is reached before *EntryCount* is reached.

TotalAvailable: The number of bytes required to see a complete listing of accounts specified by the *DisplayInformationClass* parameter.

TotalReturned: The number of bytes returned.<50>

Buffer: The accounts that are returned.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

Upon receiving this message, the responder MUST process the data from the message subject to the following constraints:

1. *DomainHandle* MUST have the required access specified in section 3.2.2.1.

2. This method MUST return a set of database objects, sorted by their **sAMAccountName** attribute value, in the domain referenced by *DomainHandle* that match the following criteria for the given *DisplayInformationClass*:

DisplayInformationClass	Database object criteria
DomainDisplayUser	All user objects (or those derived from user) with **userAccountControl** containing the UF_NORMAL_ACCOUNT bit.
DomainDisplayMachine	All user objects (or those derived from user) with userAccountControl containing the UF_WORKSTATION_TRUST_ACCOUNT or UF_SERVER_TRUST_ACCOUNT bit.
DomainDisplayGroup	All group objects (or those derived from group) with groupType equal to GROUP_TYPE_SECURITY_UNIVERSAL or GROUP_TYPE_SECURITY_ACCOUNT.
DomainDisplayOemUser	See DomainDisplayUser.
DomainDisplayOemGroup	See DomainDisplayGroup.

3. Let L be a list of accounts, sorted by **sAMAccountName**, that match the above criteria. If the *Index* parameter is nonzero, the responder MUST return objects starting from the position in L implied by the implementation-specific cookie (carried in the *Index* parameter). If the *Index* parameter is zero, the responder MUST start at the beginning of L. If the implementation-specific cookie refers to an object that has been deleted since the time in which the cookie was created, the responder MUST return objects, if any, starting from the next position in L.

4. For each candidate object to return, the responder MUST fill an element in the *Buffer* output parameter per the following table:

Element field	Value
Index	Any unsigned integer such that there are no duplicates in the set of values returned in *Buffer*; that is, each element has a unique *Index*. There is no requirement on the ordering of Index values.<51>
Rid	RID of the **objectSid** attribute.
AccountControl	**userAccountControl** attribute value.
AccountName	**sAMAccountName** attribute value.
AdminComment	**description** attribute value.
FullName	**displayName** attribute value.

Element field	Value
Attributes	See section 3.2.5.14.7 for a message processing specification.

A call with *DisplayInformationClass* set to DomainDisplayOemUser or DomainDisplayOemGroup MUST behave identically to a call with *DisplayInformationClass* set to DomainDisplayUser or DomainDisplayGroup, respectively. The only exception to this rule is that the **RPC_UNICODE_STRING** structures in the non-Oem cases of *DisplayInformationClass* MUST be translated to **RPC_STRING** structures using the OEM code page.

5. If a processing error occurs, the responder MUST return that error. Otherwise, the responder MUST return STATUS_SUCCESS.

3.2.5.3.2 SamrQueryDisplayInformation2 (Opnum 48)

The **SamrQueryDisplayInformation2** method obtains a list of accounts in name-sorted order, starting at a specified index.

```
long SamrQueryDisplayInformation2(
  [in] SAMPR_HANDLE DomainHandle,
  [in] DOMAIN_DISPLAY_INFORMATION DisplayInformationClass,
  [in] unsigned long Index,
  [in] unsigned long EntryCount,
  [in] unsigned long PreferredMaximumLength,
  [out] unsigned long* TotalAvailable,
  [out] unsigned long* TotalReturned,
  [out, switch_is(DisplayInformationClass)]
    PSAMPR_DISPLAY_INFO_BUFFER Buffer
);
```

See the description of **SamrQueryDisplayInformation3 (section 3.2.5.3.1)** for details, because the method-interface arguments and message processing are identical.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

The responder MUST behave as with a call to **SamrQueryDisplayInformation3**, with the following parameter values:

Parameter name	Parameter value
DomainHandle	SamrQueryDisplayInformation2.DomainHandle
DisplayInformationClass	SamrQueryDisplayInformation2.DisplayInformationClass
Index	SamrQueryDisplayInformation2.Index
EntryCount	SamrQueryDisplayInformation2.EntryCount
PreferredMaximumLength	SamrQueryDisplayInformation2.PreferredMaximumLength
TotalAvailable	SamrQueryDisplayInformation2.TotalAvailable
TotalReturned	SamrQueryDisplayInformation2.TotalReturned

Parameter name	Parameter value
Buffer	SamrQueryDisplayInformation2.Buffer

3.2.5.3.3 SamrQueryDisplayInformation (Opnum 40)

The **SamrQueryDisplayInformation** method obtains a list of accounts in name-sorted order, starting at a specified index.

```
long SamrQueryDisplayInformation(
  [in] SAMPR_HANDLE DomainHandle,
  [in] DOMAIN_DISPLAY_INFORMATION DisplayInformationClass,
  [in] unsigned long Index,
  [in] unsigned long EntryCount,
  [in] unsigned long PreferredMaximumLength,
  [out] unsigned long* TotalAvailable,
  [out] unsigned long* TotalReturned,
  [out, switch_is(DisplayInformationClass)]
    PSAMPR_DISPLAY_INFO_BUFFER Buffer
);
```

See the description of **SamrQueryDisplayInformation3 (section 3.2.5.3.1)** for details, because the method interface arguments and message processing are identical.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

The responder MUST behave as with a call to **SamrQueryDisplayInformation3**, with the following parameter values:

Parameter name	Parameter value
DomainHandle	SamrQueryDisplayInformation.DomainHandle
DisplayInformationClass	SamrQueryDisplayInformation.DisplayInformationClass
Index	SamrQueryDisplayInformation.Index
EntryCount	SamrQueryDisplayInformation.EntryCount
PreferredMaximumLength	SamrQueryDisplayInformation.PreferredMaximumLength
TotalAvailable	SamrQueryDisplayInformation.TotalAvailable
TotalReturned	SamrQueryDisplayInformation.TotalReturned
Buffer	SamrQueryDisplayInformation.Buffer

3.2.5.3.4 SamrGetDisplayEnumerationIndex2 (Opnum 49)

The **SamrGetDisplayEnumerationIndex2** method obtains an index into an account-name–sorted list of accounts, such that the index is the position in the list of the accounts whose account name best matches a requester-provided string.

```
long SamrGetDisplayEnumerationIndex2(
  [in] SAMPR_HANDLE DomainHandle,
  [in] DOMAIN_DISPLAY_INFORMATION DisplayInformationClass,
  [in] PRPC_UNICODE_STRING Prefix,
  [out] unsigned long* Index
);
```

DomainHandle: An RPC context handle, as specified in section 2.2.3.2, representing a domain object.

DisplayInformationClass: An enumeration indicating which set of objects to return an index into (for a subsequent **SamrQueryDisplayInformation3** method call).

Prefix: A string matched against the account name to find a starting point for an enumeration. The *Prefix* parameter enables the requester to obtain a listing of an account from **SamrQueryDisplayInformation3** such that the accounts are returned in alphabetical order with respect to their account name, starting with the account name that most closely matches *Prefix*. See below for details.

Index: A value to use as input to **SamrQueryDisplayInformation3** in order to control the accounts that are returned from that method.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

Upon receiving this message, the responder MUST process the data from the message subject to the following constraints:

1. *DomainHandle* MUST have the required access specified in section 3.2.2.1.
2. If *DisplayInformationClass* is not one of the following values, the responder MUST return an error code: DomainDisplayUser,DomainDisplayMachine, DomainDisplayGroup.
3. The output parameter called *Index* MUST be returned as an index into a one-based-indexed list of database objects sorted by their **sAMAccountName** attribute value. The index is the position of the element that just precedes the element whose **sAMAccountName** generates the longest substring match starting at the beginning of the string with the *Prefix* input parameter. If no such element exists, *Index* MUST be returned as 0.
4. The list of directory objects MUST correspond to *DisplayInformationClass* as follows:

DisplayInformationClass	Database object criteria
DomainDisplayUser	All user objects (or those derived from user) with **userAccountControl** containing the UF_NORMAL_ACCOUNT bit.
DomainDisplayMachine	All user objects (or those derived from user) with **userAccountControl** containing the UF_WORKSTATION_TRUST_ACCOUNT or UF_SERVER_TRUST_ACCOUNT bit.
DomainDisplayGroup	All group objects.

5. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.

3.2.5.3.5 SamrGetDisplayEnumerationIndex (Opnum 41)

The **SamrGetDisplayEnumerationIndex** method obtains an index into an account-name–sorted list of accounts.

```
long SamrGetDisplayEnumerationIndex(
  [in] SAMPR_HANDLE DomainHandle,
  [in] DOMAIN_DISPLAY_INFORMATION DisplayInformationClass,
  [in] PRPC_UNICODE_STRING Prefix,
  [out] unsigned long* Index
);
```

See the description of **SamrGetDisplayEnumerationIndex2 (section 3.2.5.3.4)** for details, because the method-interface arguments and processing are identical.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

The responder MUST behave as with a call to **SamrGetDisplayEnumerationIndex2**, with the following parameter values:

Parameter name	Parameter value
DomainHandle	SamrGetDisplayEnumerationIndex.DomainHandle
DisplayInformationClass	SamrGetDisplayEnumerationIndex.DisplayInformationClass
Prefix	SamrGetDisplayEnumerationIndex.Prefix
Index	SamrGetDisplayEnumerationIndex.Index

3.2.5.4 Create Pattern

These methods enable a requester to create a group, alias, or user object. These methods require a domain handle from the "open" pattern of methods (section 3.2.5.1).

See section 1.7.2 for details on how to choose between the **SamrCreateUserInDomain** and **SamrCreateUser2InDomain** variations.

See section 1.3 for a description of the "create" pattern of methods.

3.2.5.4.1 Common Processing for Group and Alias Creation

This section specifies message processing that is common for **SamrCreateAliasInDomain** and **SamrCreateGroupInDomain**. The explanation of each method specifies a **groupType** attribute to use during group and alias creation, and a section containing valid access mask values; these values are referred to in this section by the terms Provided-Group-Type and Provided-Access-Mask-Section.

Upon receiving this message, the responder MUST process the data from the message, subject to the following constraints:

1. *DomainHandle* MUST have the required access specified in section 3.2.2.1.

2. If *DomainHandle* refers to the built-in domain, the responder MUST abort the request and return a failure code.

3. All updates caused by this request MUST be performed in the same transaction.

4. On successful completion of this method, a new database object MUST be created (subsequent constraints specify attributes for this new object).

5. The following database attribute MUST be updated from the values provided in the message per the following table:

Database attribute	Message input
sAMAccountName	Name

6. The **distinguishedName** database attribute MUST be updated with a value that conforms to the constraints as specified in section 3.2.5.14.1.

7. The **objectClass** database attribute MUST be updated with the value group.

8. The **groupType** database attribute MUST be updated with the value Provided-Group-Type.

9. The security model for object creation specified in [MS-ADTS] section 5.1.3 MUST be adhered to.

10. Granted access MUST be set to *DesiredAccess* if *DesiredAccess* contains only valid access masks, per Provided-Access-Mask-Section and section 2.2.1.1 (common Access Masks); otherwise, the request MUST be aborted and STATUS_ACCESS_DENIED MUST be returned.

11. If *DesiredAccess* contains the ACCESS_SYSTEM_SECURITY bit, and the requester does NOT have the SE_SECURITY_NAME privilege (see the Windows Behavior note of [MS-LSAD] section 3.1.1.2.1 for details), the responder MUST abort processing and return STATUS_ACCESS_DENIED.

12. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.

3.2.5.4.2 SamrCreateGroupInDomain (Opnum 10)

The **SamrCreateGroupInDomain** method creates a group object within a domain.

```
long SamrCreateGroupInDomain(
  [in] SAMPR_HANDLE DomainHandle,
  [in] PRPC_UNICODE_STRING Name,
  [in] unsigned long DesiredAccess,
  [out] SAMPR_HANDLE* GroupHandle,
  [out] unsigned long* RelativeId
);
```

DomainHandle: An RPC context handle, as specified in section 2.2.3.2, representing a domain object.

Name: The value to use as the name of the group. See below for details on how this value maps to the data model.

DesiredAccess: The access requested on the *GroupHandle* on output. See section 2.2.1.5 for a listing of possible values.

GroupHandle: An RPC context handle, as specified in section 2.2.3.2.

RelativeId: The RID of the newly created group.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

This method MUST be processed per the specifications in section 3.2.5.4.1, using a group type of GROUP_TYPE_SECURITY_ACCOUNT and using access mask values from section 2.2.1.5.

3.2.5.4.3 SamrCreateAliasInDomain (Opnum 14)

The **SamrCreateAliasInDomain** method creates an alias.

```
long SamrCreateAliasInDomain(
  [in] SAMPR_HANDLE DomainHandle,
  [in] PRPC_UNICODE_STRING AccountName,
  [in] unsigned long DesiredAccess,
  [out] SAMPR_HANDLE* AliasHandle,
  [out] unsigned long* RelativeId
);
```

DomainHandle: An RPC context handle, as specified in section 2.2.3.2, representing a domain object.

AccountName: The value to use as the name of the alias. See below for details on how this value maps to the data model.

DesiredAccess: The access requested on the *AliasHandle* on output. See section 2.2.1.6 for a listing of possible values.

AliasHandle: An RPC context handle, as specified in section 2.2.3.2.

RelativeId: The RID of the newly created alias.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

This method MUST be processed per the specifications in section 3.2.5.4.1, using a group type of GROUP_TYPE_SECURITY_RESOURCE and using access mask values from section 2.2.1.6.

3.2.5.4.4 SamrCreateUser2InDomain (Opnum 50)

The **SamrCreateUser2InDomain** method creates a user.

```
long SamrCreateUser2InDomain(
  [in] SAMPR_HANDLE DomainHandle,
  [in] PRPC_UNICODE_STRING Name,
  [in] unsigned long AccountType,
  [in] unsigned long DesiredAccess,
```

```
  [out] SAMPR_HANDLE* UserHandle,
  [out] unsigned long* GrantedAccess,
  [out] unsigned long* RelativeId
);
```

DomainHandle: An RPC context handle, as specified in section 2.2.3.2, representing a domain object.

Name: The value to use as the name of the user. See the message processing shown below for details on how this value maps to the data model.

AccountType: A 32-bit value indicating the type of account to create. See the message processing shown below for possible values.

DesiredAccess: The access requested on the *UserHandle* on output. See section 2.2.1.7 for a listing of possible values.

UserHandle: An RPC context handle, as specified in section 2.2.3.2.

GrantedAccess: The access granted on *UserHandle*.

RelativeId: The RID of the newly created user.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

Upon receiving this message, the responder MUST process the data from the message subject to the following constraints:

1. *DomainHandle* MUST have the required access specified in section 3.2.2.1.
2. If *DomainHandle* refers to the built-in domain, the responder MUST abort the request and return a failure code.
3. The *AccountType* parameter from the message MUST be equal to exactly one value from the following list. If there is no match, an error status MUST be returned.
   - USER_NORMAL_ACCOUNT
   - USER_WORKSTATION_TRUST_ACCOUNT
   - USER_SERVER_TRUST_ACCOUNT
4. All updates caused by this request MUST be performed in the same transaction.
5. On successful completion of this method, a new database object MUST be created (subsequent constraints specify attributes for this new object).
6. The following database attribute MUST be updated from the values provided in the message per the following table:

Database attribute	Message input
sAMAccountName	Name

7. The **distinguishedName** attribute MUST be updated with a value that conforms to the constraints as specified in section 3.2.5.14.1. Let the term Container-Object be the object with the **distinguishedName** of the suffix chosen in section 3.2.5.14.1 for the new object. For a computer object, for example, Container-Object is, by default, the object with the **distinguishedName** CN=Computers,< DN of account domain object >.

8. The **objectClass** database attribute MUST be updated with a value determined as follows:

    1. If the *AccountType* parameter is USER_WORKSTATION_TRUST_ACCOUNT or USER_SERVER_TRUST_ACCOUNT, use computer.

    2. Otherwise, use user.

9. The **userAccountControl** attribute MUST be updated with a value per the following table. *AccountType* is the *AccountType* parameter from the message.

AccountType	userAccountControl
USER_NORMAL_ACCOUNT	UF_NORMAL_ACCOUNT | UF_ACCOUNTDISABLE
USER_WORKSTATION_TRUST_ACCOUNT	UF_WORKSTATION_TRUST_ACCOUNT | UF_ACCOUNTDISABLE*
USER_SERVER_TRUST_ACCOUNT	UF_SERVER_TRUST_ACCOUNT | UF_ACCOUNTDISABLE

    * If the *AccountType* parameter is USER_WORKSTATION_TRUST_ACCOUNT, the requester does not have the ACTRL_DS_CREATE_CHILD access correct on the Container-Object object, and the requester has the SE_MACHINE_ACCOUNT_NAME privilege, the **userAccountControl** attribute MUST be updated only with the UF_WORKSTATION_TRUST_ACCOUNT value.

10. The security model for object creation specified in [MS-ADTS] section 5.1.3 MUST NOT be adhered to.

11. If the requester does not have the ACTRL_DS_CREATE_CHILD access right on the Container-Object object and the *AccountType* parameter is USER_WORKSTATION_TRUST_ACCOUNT, then:

    1. On a DC configuration:

        1. If the identity of the responder does not have the SE_MACHINE_ACCOUNT_NAME privilege (see the Windows Behavior note of [MS-LSAD] section 3.1.1.2.1 for details), return a processing error.

        2. If the requester has the SE_MACHINE_ACCOUNT_NAME privilege, then:

            1. Let CallerSid be the SID representing the identity of the requester.

            2. The number of computer objects in the domain with msDS-creatorSID equal to CallerSid MUST be less than the value of ms-DS-MachineAccountQuota on the account domain object. On error, abort and return a failure code.

            3. msDS-creatorSID MUST be set to CallerSid.

            4. The owner and group of the default security descriptor MUST be the Domain Admins SID for the domain in which the account is created.

    2. On a non-DC configuration:

        - The responder MUST abort processing and return STATUS_ACCESS_DENIED.

12. The return parameter of *GrantedAccess* MUST be set to *DesiredAccess* if *DesiredAccess* contains only valid access masks for the user object (per section 2.2.1.7); otherwise the request MUST be aborted and STATUS_ACCESS_DENIED MUST be returned. Additionally, on a DC configuration, if the creation occurred because of a privilege (see step 11.a), the returned *GrantedAccess* MUST be restricted by the intersection of *DesiredAccess* and the following bits:

    - DELETE
    - USER_WRITE
    - USER_FORCE_PASSWORD_CHANGE

13. If *DesiredAccess* contains the ACCESS_SYSTEM_SECURITY bit and the requester does NOT have the SE_SECURITY_NAME privilege (see [MS-LSAD] section 3.1.1.2.1 for details), the responder MUST abort processing and return STATUS_ACCESS_DENIED.

14. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.

3.2.5.4.5 SamrCreateUserInDomain (Opnum 12)

The **SamrCreateUserInDomain** method creates a user.

```
long SamrCreateUserInDomain(
  [in] SAMPR_HANDLE DomainHandle,
  [in] PRPC_UNICODE_STRING Name,
  [in] unsigned long DesiredAccess,
  [out] SAMPR_HANDLE* UserHandle,
  [out] unsigned long* RelativeId
);
```

DomainHandle: An RPC context handle, as specified in section 2.2.3.2, representing a domain object.

Name: The value to use as the name of the user. See the message processing shown below for details on how this value maps to the data model.

DesiredAccess: The access requested on the *UserHandle* on output. See section 2.2.1.7 for a listing of possible values.

UserHandle: An RPC context handle, as specified in section 2.2.3.2.

RelativeId: The RID of the newly created user.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

The responder MUST behave as with a call to **SamrCreateUser2InDomain** with the following parameter values:

Parameter name	Parameter value
DomainHandle	SamrCreateUserInDomain.DomainHandle
Name	SamrCreateUserInDomain.Name

Parameter name	Parameter value
AccountType	USER_NORMAL_ACCOUNT
DesiredAccess	SamrCreateUserInDomain.DesiredAccess
UserHandle	SamrCreateUserInDomain.UserHandle
RelativeId	SamrCreateUserInDomain.RelativeId

3.2.5.5 Query Pattern

These methods enable a requester to read attributes about a domain, group, alias, or user object.

A requester must first obtain a handle to the object through an "open" or "create" method. See sections 3.2.5.1 and 3.2.5.4.

See section 1.7.2 for details on how to choose between **SamrQueryInformationUser**, **SamrQueryInformationUser2**, **SamrQueryInformationDomain**, and **SamrQueryInformationDomain2** variations.

See section 1.3 for a description of the "query" pattern of methods.

3.2.5.5.1 SamrQueryInformationDomain2 (Opnum 46)

The **SamrQueryInformationDomain2** method obtains attributes from a domain object.

```
long SamrQueryInformationDomain2(
  [in] SAMPR_HANDLE DomainHandle,
  [in] DOMAIN_INFORMATION_CLASS DomainInformationClass,
  [out, switch_is(DomainInformationClass)]
    PSAMPR_DOMAIN_INFO_BUFFER* Buffer
);
```

DomainHandle: An RPC context handle, as specified in section 2.2.3.2, representing a domain object.

DomainInformationClass: An enumeration indicating which attributes to return. See section 2.2.4.16 for a listing of possible values.

Buffer: The requested attributes on output. See section 2.2.4.17 for structure details.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

Upon receiving this message, the responder MUST process the data from the message subject to the following constraints.

1. DomainHandle MUST have the required access specified in section 3.2.2.1.
2. The following information levels MUST be processed by setting the appropriate output field name to the associated database attribute, per section 3.2.5.14.8. Processing is completed by returning 0 on success. If any database errors occur, the responder MUST abort processing and return a database processing error.

DomainInformationClass
DomainPasswordInformation
DomainLockoutInformation
DomainLogoffInformation
DomainOemInformation
DomainNameInformation
DomainModifiedInformation
DomainModifiedInformation2

3. If *DomainInformationClass* does not meet the criteria of constraint 2, the constraints associated with the *DomainInformationClass* input value in the following subsections MUST be satisfied; if there is no subsection for the *DomainInformationClass* value, an error MUST be returned to the requester.

3.2.5.5.1.1 DomainGeneralInformation

1. The responder MUST use the database attribute value on the directory object referred to by *DomainHandle* to set the *Buffer* fields per the table in section 3.2.5.14.8.

2. The **Buffer.General.DomainServerState** field MUST be set to DomainStateEnabled.

3. If the **fsmoRoleOwner** attribute value of the account domain object is equal to the **distinguishedName** attribute value of the responder's computer object, the **Buffer.General.DomainServerRole** field MUST be set to DomainServerRolePrimary; otherwise the value MUST be set to DomainServerRoleBackup.

4. **Buffer.General.UasCompatibilityRequired** MUST be set to 1 if the **uasCompat** database attribute value on the domain object is nonzero.

5. The **Buffer.General.UserCount** field SHOULD<52> be the count of objects with the objectClass user (or derived from user). The accuracy is bounded by the requirement that if there is at least one user object, the value MUST be greater than 0.

6. The **Buffer.General.GroupCount** field SHOULD<53> be the count of objects with the objectClass group (or derived from group) and a **groupType** attribute value of GROUP_TYPE_SECURITY_ACCOUNT. The accuracy is bounded by the requirement that if there is at least one such group object, the value MUST be greater than 0.

7. The **Buffer.General.AliasCount** field SHOULD<54> be the count of objects with the objectClass group (or derived from group) and a groupType attribute value of GROUP_TYPE_SECURITY_RESOURCE. The accuracy is bounded by the requirement that if there is at least one alias object, the value MUST be greater than 0.

8. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.

3.2.5.5.1.2 DomainServerRoleInformation

1. If the **fsmoRoleOwner** attribute value of the account domain object is equal to the **distinguishedName** attribute value of the responder's computer object, the

Buffer.Role.DomainServerRole field MUST be set to DomainServerRolePrimary; otherwise the value MUST be set to DomainServerRoleBackup.

2. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.

3.2.5.5.1.3 DomainStateInformation

1. The responder MUST set **Buffer.State.DomainServerState** to DomainServerEnabled.

2. Return STATUS_SUCCESS if no processing errors have occurred.

3.2.5.5.1.4 DomainGeneralInformation2

The responder MUST process this call as two calls to **SamrQueryInformationDomain** with the information levels of DomainGeneralInformation and DomainLockoutTime, but all in the same transaction. The output fields MUST be set as follows:

Message output field	Value
Buffer.General2.I1	**SAMPR_DOMAIN_GENERAL_INFORMATION**
Buffer.General2.LockoutDuration	**SAMPR_DOMAIN_LOCKOUT_INFORMATION**.LockoutDuration
Buffer.General2.LockoutObservationWindow	**SAMPR_DOMAIN_LOCKOUT_INFORMATION**.LockoutObservationWindow
Buffer.General2.LockoutThreshold	**SAMPR_DOMAIN_LOCKOUT_INFORMATION**.LockoutThreshold

3.2.5.5.2 SamrQueryInformationDomain (Opnum 8)

The **SamrQueryInformationDomain** method obtains attributes from a domain object.

```
long SamrQueryInformationDomain(
  [in] SAMPR_HANDLE DomainHandle,
  [in] DOMAIN_INFORMATION_CLASS DomainInformationClass,
  [out, switch_is(DomainInformationClass)]
    PSAMPR_DOMAIN_INFO_BUFFER* Buffer
);
```

See the description of **SamrQueryInformationDomain2 (section 3.2.5.5.1)** for details, because the method interface arguments and message processing are identical.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

The responder MUST behave as with a call to **SamrQueryInformationDomain2**, with the following parameter values:

Parameter name	Parameter value
DomainHandle	SamrQueryInformationDomain.DomainHandle

Parameter name	Parameter value
DomainInformationClass	SamrQueryInformationDomain.DomainInformationClass
Buffer	SamrQueryInformationDomain.Buffer

3.2.5.5.3 SamrQueryInformationGroup (Opnum 20)

The **SamrQueryInformationGroup** method obtains attributes from a group object.

```
long SamrQueryInformationGroup(
  [in] SAMPR_HANDLE GroupHandle,
  [in] GROUP_INFORMATION_CLASS GroupInformationClass,
  [out, switch_is(GroupInformationClass)]
    PSAMPR_GROUP_INFO_BUFFER* Buffer
);
```

GroupHandle: An RPC context handle, as specified in section 2.2.3.2, representing a group object.

GroupInformationClass: An enumeration indicating which attributes to return. See section 2.2.5.6 for a listing of possible values.

Buffer: The requested attributes on output. See section 2.2.5.7 for structure details.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

Upon receiving this message, the responder MUST process the data from the message subject to the following constraints:

1. *GroupHandle* MUST have the required access specified in section 3.2.2.1.
2. The following information levels MUST be processed by setting the appropriate output field name to the associated database attribute, per section 3.2.5.14.9. Processing is completed by returning 0 on success. If any database errors occur, the responder MUST abort processing and return a database processing error.

GroupInformationClass
GroupGeneralInformation
GroupNameInformation
GroupAttributeInformation
GroupAdminCommentInformation

3. If *GroupInformationClass* does not meet the criteria of constraint 2, the constraints associated with the *GroupInformationClass* input value in the following subsections MUST be satisfied; if there is no subsection for the *GroupInformationClass* value, an error MUST be returned to the requester.

3.2.5.5.3.1 GroupReplicationInformation

This information level is an anomaly in that it sets the *Buffer* fields for General, whereas in the union structure of **SAMP_GROUP_INFO_BUFFER (section 2.2.5.7)** the information level is associated with a different field (named **DoNotUse**).

1. The responder MUST use the database attribute value on the directory object referred to by *GroupHandle* to set the outgoing method parameters per the following table:

Message output	Database attribute
Buffer.General.Name	sAMAccountName
Buffer.General.Attributes	See section 3.2.5.14.7 for a message processing specification.
Buffer.General.AdminComment	description
Buffer.General.MemberCount	0

2. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.

3.2.5.5.4 SamrQueryInformationAlias (Opnum 28)

The **SamrQueryInformationAlias** method obtains attributes from an alias object.

```
long SamrQueryInformationAlias(
  [in] SAMPR_HANDLE AliasHandle,
  [in] ALIAS_INFORMATION_CLASS AliasInformationClass,
  [out, switch_is(AliasInformationClass)]
    PSAMPR_ALIAS_INFO_BUFFER* Buffer
);
```

AliasHandle: An RPC context handle, as specified in section 2.2.3.2, representing an alias object.

AliasInformationClass: An enumeration indicating which attributes to return. See section 2.2.6.5 for a listing of possible values.

Buffer: The requested attributes on output. See section 2.2.6.6 for structure details.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

Upon receiving this message, the responder MUST process the data from the message subject to the following constraints:

1. *AliasHandle* MUST have the required access specified in section 3.2.2.1.

2. The following information levels MUST be processed by setting the appropriate output field name to the associated database attribute, per section 3.2.5.14.10. Processing is completed by returning 0 on success. If any database errors occur, the responder MUST abort processing and return a database processing error. If the presented information level is not in the following table, the responder MUST return an error.

AliasInformationClass
AliasGeneralInformation
AliasNameInformation
AliasAdminCommentInformation

3.2.5.5.5 SamrQueryInformationUser2 (Opnum 47)

The **SamrQueryInformationUser2** method obtains attributes from a user object.

```
long SamrQueryInformationUser2(
  [in] SAMPR_HANDLE UserHandle,
  [in] USER_INFORMATION_CLASS UserInformationClass,
  [out, switch_is(UserInformationClass)]
    PSAMPR_USER_INFO_BUFFER* Buffer
);
```

UserHandle: An RPC context handle, as specified in section 2.2.3.2, representing a user object.

UserInformationClass: An enumeration indicating which attributes to return. See section 2.2.7.29 for a list of possible values.

Buffer: The requested attributes on output. See section 2.2.7.30 for structure details.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

Upon receiving this message, the responder MUST process the data from the message subject to the following constraints:

1. *DomainHandle* MUST have the required access specified in section 3.2.2.1.
2. If *UserInformationClass* is UserAllInformation, the constraints in the following "UserAllInformation" section (section 3.2.5.5.5.2) MUST be satisfied. Otherwise, the constraints in the following "Common Processing" section (section 3.2.5.5.5.1) MUST be satisfied.

3.2.5.5.5.1 Common Processing

1. *UserHandle* MUST have the Required Access per the following table; on error, the responder MUST return STATUS_ACCESS_DENIED. If there is no match on Information Level, the responder MUST return an error.

Information level	Required access
UserAccountInformation	USER_READ_GENERAL | USER_READ_PREFERENCES | USER_READ_LOGON | USER_READ_ACCOUNT
UserGeneralInformation	USER_READ_GENERAL
UserPrimaryGroupInformation	USER_READ_GENERAL
UserNameInformation	USER_READ_GENERAL

Information level	Required access
UserAccountNameInformation	USER_READ_GENERAL
UserFullNameInformation	USER_READ_GENERAL
UserAdminCommentInformation	USER_READ_GENERAL
UserPreferencesInformation	USER_READ_PREFERENCES | USER_READ_GENERAL
UserLogonInformation	USER_READ_GENERAL | USER_READ_PREFERENCES | USER_READ_LOGON | USER_READ_ACCOUNT
UserLogonHoursInformation	USER_READ_LOGON
UserHomeInformation	USER_READ_LOGON
UserScriptInformation	USER_READ_LOGON
UserProfileInformation	USER_READ_LOGON
UserWorkStationsInformation	USER_READ_LOGON
UserControlInformation	USER_READ_ACCOUNT
UserExpiresInformation	USER_READ_ACCOUNT
UserParametersInformation	USER_READ_ACCOUNT (*)

(*) In the DC configuration, this handle-based check MUST be relaxed if the requester has ACTRL_DS_READ_PROP access on the **userParameters** attribute (GUID bf967a6d-0de6-11d0-a285-00aa003049e2).

2. The message processing MUST be similar to a **SamrQueryInformationUser2** call with the *UserInformationClass* parameter set to UserAllInformation; that is, similar in the manner in which the fields are set from database attributes, but different in that the only processing errors that are propagated to the requester are those errors related to the fields specifically requested. On return, the requested fields MUST be set to the value of the field with the same name in the **SAMPR_USER_ALL_INFORMATION** structure.

   The following table shows an example for an information level of UserGeneralInformation.

Information level: UserGeneralInformation	
Field of the Buffer parameter	**Field value (from SAMPR_USER_ALL)**
General.UserName	UserName
General.FullName	FullName
General.PrimaryGroupId	PrimaryGroupId
General.AdminComment	AdminComment
General.UserComment	UserComment

3.2.5.5.5.2 UserAllInformation

1. The responder MUST set the fields of **Buffer.All** based on the access granted in the *UserHandle*. The following table normatively specifies the value that the responder MUST set in the **Buffer.All.WhichFields** field. If *UserHandle* does not have any of the Access Granted bits from this table, the responder MUST return STATUS_ACCESS_DENIED.

Access Granted	WhichFields
USER_READ_GENERAL	USER_ALL_USERNAME USER_ALL_FULLNAME USER_ALL_USERID USER_ALL_PRIMARYGROUPID USER_ALL_ADMINCOMMENT USER_ALL_USERCOMMENT
USER_READ_LOGON	USER_ALL_HOMEDIRECTORY USER_ALL_HOMEDIRECTORYDRIVE USER_ALL_SCRIPTPATH USER_ALL_PROFILEPATH USER_ALL_WORKSTATIONS USER_ALL_LASTLOGON USER_ALL_LASTLOGOFF USER_ALL_LOGONHOURS USER_ALL_BADPASSWORDCOUNT USER_ALL_LOGONCOUNT USER_ALL_PASSWORDCANCHANGE USER_ALL_PASSWORDMUSTCHANGE
USER_READ_ACCOUNT	USER_ALL_PASSWORDLASTSET USER_ALL_ACCOUNTEXPIRES USER_ALL_USERACCOUNTCONTROL USER_ALL_PARAMETERS
USER_READ_PREFERENCES	USER_ALL_COUNTRYCODE USER_ALL_CODEPAGE

2. Using the tables in sections 2.2.1.8 and 3.2.5.14.11, the responder MUST set the appropriate fields in the *Buffer* parameter. The first table (section 2.2.1.8) lists the WhichFields-to-field-name mapping, and the second table (section 3.2.5.14.11) specifies the field-name-to-database-attribute mapping.

3. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.

3.2.5.5.6 SamrQueryInformationUser (Opnum 36)

The **SamrQueryInformationUser** method obtains attributes from a user object.

```
long SamrQueryInformationUser(
  [in] SAMPR_HANDLE UserHandle,
```

```
  [in] USER_INFORMATION_CLASS UserInformationClass,
  [out, switch_is(UserInformationClass)]
    PSAMPR_USER_INFO_BUFFER* Buffer
);
```

See the description of **SamrQueryInformationUser2 (section 3.2.5.5.5)** for details, because the method interface arguments and message processing are identical.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

The responder MUST behave as with a call to **SamrQueryInformationUser2**, with the following parameter values:

Parameter name	Parameter value
UserHandle	SamrQueryInformationUser.UserHandle
UserInformationClass	SamrQueryInformationUser.UserInformationClass
Buffer	SamrQueryInformationUser.Buffer

3.2.5.6 Set Pattern

These methods enable a requester to set attributes on a domain, group, alias, or user object.

A requester must first obtain a handle to the object through an "open" or "create" method. See sections 3.2.5.1 and 3.2.5.4.

See section 1.7.2 for details on how to choose between **SamrSetInformationUser** and **SamrSetInformationUser2**.

See section 1.3 for a description of the "set" pattern of methods.

3.2.5.6.1 SamrSetInformationDomain (Opnum 9)

The **SamrSetInformationDomain** method updates attributes on a domain object.

```
long SamrSetInformationDomain(
  [in] SAMPR_HANDLE DomainHandle,
  [in] DOMAIN_INFORMATION_CLASS DomainInformationClass,
  [in, switch_is(DomainInformationClass)]
    PSAMPR_DOMAIN_INFO_BUFFER DomainInformation
);
```

DomainHandle: An RPC context handle, as specified in section 2.2.3.2, representing a domain object.

DomainInformationClass: An enumeration indicating which attributes to update. See section 2.2.4.16 for a list of possible values.

DomainInformation: The requested attributes and values to update. See section 2.2.4.17 for structure details.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

Upon receiving this message, the responder MUST process the data from the message subject to the following constraints.

1. *DomainHandle* MUST have the required access specified in section 3.2.2.1.
2. The following information levels MUST be processed by setting the database attribute on the domain object associated with *DomainHandle* to the associated input field-name value using the mapping in section 3.2.5.14.8. All updates MUST be performed in the same transaction. If any database errors occur, the responder MUST abort processing and return a database processing error; otherwise return STATUS_SUCCESS.

DomainInformationClass
DomainPasswordInformation
DomainLockoutInformation
DomainLogoffInformation
DomainOemInformation
DomainReplicationInformation

3. If *DomainInformationClass* does not meet the criteria of constraint 2, the constraints associated with the *DomainInformationClass* input value in the following subsections MUST be satisfied. If there is no subsection for the *DomainInformationClass* value, an error MUST be returned to the requester.

3.2.5.6.1.1 DomainServerRoleInformation

1. If the responder is not a DC, an error status MUST be returned.
2. If *DomainHandle* refers to the built-in domain, the responder MUST abort and return STATUS_SUCCESS.
3. If **DomainInformation.Role.DomainServerRole** is not equal to DomainServerRolePrimary, a status of 0 MUST be returned.
4. The operation to request the PDC role MUST be performed; any resulting processing errors MUST be returned. Otherwise, return STATUS_SUCCESS.

3.2.5.6.1.2 DomainStateInformation

The responder MUST return STATUS_SUCCESS.

3.2.5.6.2 SamrSetInformationGroup (Opnum 21)

The **SamrSetInformationGroup** method updates attributes on a group object.

```
long SamrSetInformationGroup(
  [in] SAMPR_HANDLE GroupHandle,
  [in] GROUP_INFORMATION_CLASS GroupInformationClass,
```

```
  [in, switch_is(GroupInformationClass)]
    PSAMPR_GROUP_INFO_BUFFER Buffer
);
```

GroupHandle: An RPC context handle, as specified in section 2.2.3.2, representing a group object.

GroupInformationClass: An enumeration indicating which attributes to update. See section 2.2.5.6 for a listing of possible values.

Buffer: The requested attributes and values to update. See section 2.2.5.7 for structure details.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

Upon receiving this message, the responder MUST process the data from the message subject to the following constraints:

1. *GroupHandle* MUST have the required access specified in section 3.2.2.1.
2. The following information levels MUST be processed by setting the database attribute on the group object associated with *GroupHandle* to the associated input field-name value using the mapping in section 3.2.5.14.9. All updates MUST be performed in the same transaction. If any database errors occur, the responder MUST abort processing and return a database processing error; otherwise, return STATUS_SUCCESS.

GroupInformationClass
GroupNameInformation
GroupAttributeInformation
GroupAdminCommentInformation

3. If *GroupInformationClass* does not meet the criteria of constraint 2, the responder MUST return an error code.

3.2.5.6.3 SamrSetInformationAlias (Opnum 29)

The **SamrSetInformationAlias** method updates attributes on an alias object.

```
long SamrSetInformationAlias(
  [in] SAMPR_HANDLE AliasHandle,
  [in] ALIAS_INFORMATION_CLASS AliasInformationClass,
  [in, switch_is(AliasInformationClass)]
    PSAMPR_ALIAS_INFO_BUFFER Buffer
);
```

AliasHandle: An RPC context handle, as specified in section 2.2.3.2, representing an alias object.

AliasInformationClass: An enumeration indicating which attributes to update. See section 2.2.6.5 for a listing of possible values.

Buffer: The requested attributes and values to update. See section 2.2.6.6 for structure details.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

Upon receiving this message, the responder MUST process the data from the message subject to the following constraints:

1. *AliasHandle* MUST have the required access specified in section 3.2.2.1.
2. The following information levels MUST be processed by setting the database attribute on the alias object associated with *AliasHandle* to the associated input field-name value using the mapping in section 3.2.5.14.10. All updates MUST be performed in the same transaction. If any database errors occur, the responder MUST abort processing and return a database processing error; otherwise, return STATUS_SUCCESS.

AliasInformationClass
AliasNameInformation
AliasAdminInformation

3. If *AliasInformationClass* does not meet the criteria of constraint 2, the responder MUST return an error code.

3.2.5.6.4 SamrSetInformationUser2 (Opnum 58)

The **SamrSetInformationUser2** method updates attributes on a user object.

```
long SamrSetInformationUser2(
  [in] SAMPR_HANDLE UserHandle,
  [in] USER_INFORMATION_CLASS UserInformationClass,
  [in, switch_is(UserInformationClass)]
    PSAMPR_USER_INFO_BUFFER Buffer
);
```

UserHandle: An RPC context handle, as specified in section 2.2.3.2, representing a user object.

UserInformationClass: An enumeration indicating which attributes to update. See section 2.2.7.29 for a listing of possible values.

Buffer: The requested attributes and values to update. See section 2.2.7.30 for structure details.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

Upon receiving this message, the responder MUST process the data from the message subject to the following constraints:

1. *UserHandle* MUST have the required access specified in section 3.2.2.1.

2. The constraints in the following sections MUST be satisfied based on the *UserInformationClass* parameter. If there is no match in the table, the constraints of section 3.2.5.6.4.1 MUST be used.

UserInformationClass	Constraint section
UserAllInformation	3.2.5.6.4.3
UserInternal4Information	3.2.5.6.4.4
UserInternal4InformationNew	3.2.5.6.4.5

3.2.5.6.4.1 Common Processing

1. If the value of *UserInformationClass* is present in the following table, the message MUST be processed exactly as a call to **SamrSetInformationUser2** with *UserInformationClass* set to UserAllInformation and Buffer of type **SAMPR_USER_ALL_INFORMATION**.

UserInformationClass value
UserPreferencesInformation
UserParametersInformation
UserNameInformation
UserAccountNameInformation
UserFullNameInformation
UserPrimaryGroupInformation
UserHomeInformation
UserScriptInformation
UserProfileInformation
UserAdminCommentInformation
UserWorkStationsInformation
UserControlInformation
UserInternal1Information

All **SAMPR_USER_ALL_INFORMATION** fields with the same name as the fields in the incoming structure must be set with the same value. Furthermore, the **WhichFields** field MUST be updated per the table in section 2.2.1.8. All **SAMPR_USER_ALL_INFORMATION** fields not covered MUST be zero.

As an example, the following table shows how a request for UserPreferencesInformation MUST be handled.

Source UserInformationClass value: UserPreferencesInformation	
Target: SAMPR_USER_ALL_INFORMATION	
Field name	**Value**
WhichFields	USER_ALL_USERCOMMENT | USER_ALL_COUNTRYCODE | USER_ALL_CODEPAGE
UserComment	Preferences.UserComment
CountryCode	Preferences.CountryCode
CodePage	Preferences.CodePage

A request for Internal1Information is a slight exception and thus is shown explicitly in the following table.

Source UserInformationClass value: UserInternal1Information	
Target: SAMPR_USER_ALL_INFORMATION	
Field name	**Value**
WhichFields	USER_ALL_NTPASSWORDPRESENT | USER_ALL_LMPASSWORDPRESENT | USER_ALL_PASSWORDEXPIRED
NtPasswordPresent	Internal1.NtPasswordPresent
LmPasswordPresent	Internal1.LmPasswordPresent
PasswordExpired	Internal1.PasswordExpired
LmOwfPassword.Length	0x10
LmOwfPassword.MaximumLength	0x10
LmOwfPassword.Buffer	Internal1.LmOwfPassword
NtOwfPassword.Length	0x10
NtOwfPassword.MaximumLength	0x10
NtOwfPassword.Buffer	Internal1.NtOwfPassword

2. If the value of *UserInformationClass* is UserInternal5InformationNew, the message MUST be processed exactly as a call to **SamrSetInformationUser2** with *UserInformationClass* set to UserInternal4InformationNew and Buffer of type **SAMPR_USER_INTERNAL4_INFORMATION_NEW** with the fields set per the table below. All **SAMPR_USER_INTERNAL4_INFORMATION_NEW** fields not covered by the table MUST be zero.

Source UserInformationClass value: UserInternal5InformationNew	
Target: SAMPR_USER_INTERNAL4_INFORMATION_NEW	
Field name	**Value**
I1.WhichFields	USER_ALL_NTPASSWORDPRESENT | USER_ALL_LMPASSWORDPRESENT | USER_ALL_PASSWORDEXPIRED
I1.PasswordExpired	Internal5.PasswordExpired
UserPassword	Internal5.UserPassword

3. If the value of *UserInformationClass* is UserInternal5Information, the message MUST be processed exactly as a call to **SamrSetInformationUser2** with *UserInformationClass* set to UserInternal4Information and Buffer of type **SAMPR_USER_INTERNAL4_INFORMATION** with the fields set per the table below. All **SAMPR_USER_INTERNAL4_INFORMATION** fields not covered by the table MUST be zero.

Source UserInformationClass value: UserInternal5Information	
Target: SAMPR_USER_INTERNAL4_INFORMATION	
Field name	**Value**
I1.WhichFields	USER_ALL_NTPASSWORDPRESENT | USER_ALL_LMPASSWORDPRESENT | USER_ALL_PASSWORDEXPIRED
I1.PasswordExpired	Internal5.PasswordExpired
UserPassword	Internal5.UserPassword

4. If the value of *UserInformationClass* was not found in the previous three constraints, the responder MUST return an error.

3.2.5.6.4.2 UserAllInformation (Common)

The responder MUST process the message subject to the following constraints on the **SAMPR_USER_ALL_INFORMATION** message parameter:

1. If the **WhichFields** field is 0 or contains any of the following bits, the responder MUST abort and return an error.

Bit
USER_ALL_USERID
USER_ALL_PASSWORDCANCHANGE
USER_ALL_PASSWORDMUSTCHANGE
USER_ALL_UNDEFINED_MASK
USER_ALL_LASTLOGON

Bit
USER_ALL_LASTLOGOFF
USER_ALL_BADPASSWORDCOUNT
USER_ALL_LOGONCOUNT
USER_ALL_PASSWORDLASTSET
USER_ALL_SECURITYDESCRIPTOR
USER_ALL_PRIVATEDATA

2. The *UserHandle* MUST be granted the following access based on the value of the **WhichFields** field.

WhichFields	Required access
USER_ALL_USERNAME	USER_WRITE_ACCOUNT
USER_ALL_FULLNAME	USER_WRITE_ACCOUNT
USER_ALL_PRIMARYGROUPID	USER_WRITE_ACCOUNT
USER_ALL_HOMEDIRECTORY	USER_WRITE_ACCOUNT
USER_ALL_HOMEDIRECTORYDRIVE	USER_WRITE_ACCOUNT
USER_ALL_SCRIPTPATH	USER_WRITE_ACCOUNT
USER_ALL_PROFILEPATH	USER_WRITE_ACCOUNT
USER_ALL_ADMINCOMMENT	USER_WRITE_ACCOUNT
USER_ALL_WORKSTATIONS	USER_WRITE_ACCOUNT
USER_ALL_LOGONHOURS	USER_WRITE_ACCOUNT
USER_ALL_ACCOUNTEXPIRES	USER_WRITE_ACCOUNT
USER_ALL_USERACCOUNTCONTROL	USER_WRITE_ACCOUNT
USER_ALL_PARAMETERS	USER_WRITE_ACCOUNT
USER_ALL_USERCOMMENT	USER_WRITE_PREFERENCES
USER_ALL_COUNTRYCODE	USER_WRITE_PREFERENCES
USER_ALL_CODEPAGE	USER_WRITE_PREFERENCES
USER_ALL_NTPASSWORDPRESENT	USER_FORCE_PASSWORD_CHANGE
USER_ALL_LMPASSWORDPRESENT	USER_FORCE_PASSWORD_CHANGE
USER_ALL_PASSWORDEXPIRED	USER_FORCE_PASSWORD_CHANGE

3. The responder MUST update the corresponding database attributes for each bit that is present in the **WhichFields** field. In addition, the responder MUST enforce that the requester has

ACTRL_DS_READ_PROP access to the database attribute being updated, per the *UserHandle* passed into the method. Section 2.2.1.8 specifies a WhichFields-to-field mapping, and section 3.2.5.14.11 specifies a field-to-database-attribute mapping.

4. If the USER_ALL_USERACCOUNTCONTROL bit is present in the **WhichFields** field, the responder MUST:

    1. Enforce that the requester has ACTRL_DS_READ_PROP access to the database attribute of **userAccountControl**, per the *UserHandle* passed into the method.

    2. Translate the bits per the table in section 3.2.5.14.2. If a bit does not translate, abort with a processing error.

    3. Update the **userAccountControl** attribute in the database.

5. If the USER_ALL_PASSWORDEXPIRED flag is present in the **WhichFields** field, the responder MUST:

    1. If **Buffer.All.PasswordExpired** is nonzero, then:

        - Update the **pwdLastSet** with a value of 0.

    2. If **Buffer.All.PasswordExpired** is 0 and the value of the current time minus the **pwdLastSet** attribute is greater than the Effective-MaximumPasswordAge (see section 3.2.1.5), then:

        - Update the **pwdLastSet** attribute with a value of the current time.

    3. Enforce that this update to **pwdLastSet** MUST take precedence over any other writes to this attribute during the message processing and associated triggers.

3.2.5.6.4.3 UserAllInformation

The responder MUST process the message subject to the following constraints:

1. All updates MUST be done in the same transaction.

2. The responder MUST satisfy the constraints listed in UserAllInformation (Common) (section 3.2.5.6.4.2).

3. If the USER_ALL_NTPASSWORDPRESENT flag is present in the **WhichFields** field, the responder MUST:

    1. If **Buffer.All.NtPasswordPresent** is true:

        - Update the **unicodePwd** attribute with the (decrypted) value of **Buffer.All.NtOwfPassword.Buffer**.

    2. If **Buffer.All.NtPasswordPresent** is false:

        - Update the **unicodePwd** attribute with the NT hash of a zero-length string.

4. If the USER_ALL_LMPASSWORDPRESENT flag is present in the **WhichFields** field, the responder MUST:

    1. If **Buffer.All.LmPasswordPresent** is true, update the **dbcsPwd** attribute with the (decrypted) value of **Buffer.All.LmOwfPassword.Buffer**.

2. If **Buffer.All.NtPasswordPresent** is false, update **dbcsPwd** attribute with the LM hash of a zero-length string.

5. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.

3.2.5.6.4.4 UserInternal4Information

The responder MUST process the message subject to the following constraints:

1. All updates MUST be done in the same transaction.

2. The responder MUST satisfy the constraints listed in section 3.2.5.6.4.2.

3. If the USER_ALL_NTPASSWORDPRESENT or USER_ALL_LMPASSWORDPRESENT flag is present in the **WhichFields** field, the responder MUST update the **clearTextPassword** attribute with the (decrypted) value of **SAMPR_USER_INTERNAL4_INFORMATION.UserPassword**, using the decryption key of the 16-byte SMB [MS-SMB] session key established by the underlying authentication protocol (Kerberos or NTLM).

4. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.

3.2.5.6.4.5 UserInternal4InformationNew

The responder MUST process the message subject to the following constraints:

1. All updates MUST be done in the same transaction.

2. The responder MUST satisfy the constraints listed in section 3.2.5.6.4.2.

3. If the USER_ALL_NTPASSWORDPRESENT or USER_ALL_LMPASSWORDPRESENT flag is present in the **WhichFields** field, the responder MUST update the **clearTextPassword** attribute with the (decrypted) value of **SAMPR_USER_INTERNAL4_INFORMATION_NEW.UserPassword**.

4. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.

3.2.5.6.5 SamrSetInformationUser (Opnum 37)

The **SamrSetInformationUser** method updates attributes on a user object.

```
long SamrSetInformationUser(
  [in] SAMPR_HANDLE UserHandle,
  [in] USER_INFORMATION_CLASS UserInformationClass,
  [in, switch_is(UserInformationClass)]
    PSAMPR_USER_INFO_BUFFER Buffer
);
```

See the description of **SamrSetInformationUser2 (section 3.2.5.6.4)** for details, because the method interface arguments and message processing are identical.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

The responder MUST behave as with a call to **SamrSetInformationUser2**, with the following parameter values:

Parameter name	Parameter value
UserHandle	SamrSetInformationUser.UserHandle
UserInformationClass	SamrSetInformationUser.UserInformationClass
Buffer	SamrSetInformationUser.Buffer

3.2.5.7 Delete Pattern

These methods enable a requester to delete a group, alias, or user object.

A requester must first obtain a handle to the object through an "open" or "create" method. See sections 3.2.5.1 and 3.2.5.4.

See section 1.3 for a description of the "delete" pattern of methods.

3.2.5.7.1 SamrDeleteGroup (Opnum 23)

The **SamrDeleteGroup** method removes a group object.

```
long SamrDeleteGroup(
  [in, out] SAMPR_HANDLE* GroupHandle
);
```

GroupHandle: An RPC context handle, as specified in section 2.2.3.2, representing a group object.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

Upon receiving this message, the responder MUST process the data from the message subject to the following constraints:

1. *GroupHandle* MUST have the required access specified in section 3.2.2.1.
2. All database operations MUST occur in a single transaction.
3. Let G be the group referenced by the *GroupHandle*.
4. If the RID of G's **objectSid** attribute is less than or equal to 1000, an error MUST be returned.
5. If G has any values in the member attribute, an error MUST be returned.
6. If any user in the same domain as G has, as its **primaryGroupId** attribute, the RID of G's **objectSid** attribute, an error MUST be returned.
7. In the DC configuration, if G is a parent to another object, the requester MUST have the ACTRL_DS_DELETE_TREE access per G's **ntSecurityDescriptor**; if not, an error MUST be returned.
8. G MUST be removed from the database.

9. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.

3.2.5.7.2 SamrDeleteAlias (Opnum 30)

The **SamrDeleteAlias** method removes an alias object.

```
long SamrDeleteAlias(
  [in, out] SAMPR_HANDLE* AliasHandle
);
```

AliasHandle: An RPC context handle, as specified in section 2.2.3.2, representing an alias object.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

Upon receiving this message, the responder MUST process the data from the message subject to the following constraints:

1. *AliasHandle* MUST have the required access specified in section 3.2.2.1.
2. All database operations MUST occur in a single transaction.
3. Let A be the alias object referenced by *AliasHandle*.
4. If the RID of A's **objectSid** attribute value is less than or equal to 1000, an error MUST be returned.
5. In the DC configuration, if the alias object is a parent to another object, the requester MUST have the ACTRL_DS_DELETE_TREE access per A's **ntSecurityDescriptor**.
6. The database object to which *AliasHandle* refers MUST be removed from the database.
7. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.

3.2.5.7.3 SamrDeleteUser (Opnum 35)

The **SamrDeleteUser** method removes a user object.

```
long SamrDeleteUser(
  [in, out] SAMPR_HANDLE* UserHandle
);
```

UserHandle: An RPC context handle, as specified in section 2.2.3.2, representing a user object.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

Upon receiving this message, the responder MUST process the data from the message subject to the following constraints:

1. *UserHandle* MUST have the required access specified in section 3.2.2.1.
2. Let U be the object referenced by *UserHandle*.
3. All database operations MUST occur in a single transaction.
4. If the RID of U's **objectSid** attribute value is less than or equal to 1000, an error MUST be returned.
5. In the DC configuration, if U is a parent to another object, the requester MUST have the ACTRL_DS_DELETE_TREE access per U's **ntSecurityDescriptor**.
6. U MUST be removed from the database.
7. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.

3.2.5.8 Membership Pattern

These methods enable a requester to set and query the membership of a group or alias.

A requester must first obtain a handle to the group or alias object through an "open" or "create" method. See sections 3.2.5.1 and 3.2.5.4.

See section 1.3 for a description of the "membership" pattern of methods.

3.2.5.8.1 SamrAddMemberToGroup (Opnum 22)

The **SamrAddMemberToGroup** method adds a member to a group.

```
long SamrAddMemberToGroup(
  [in] SAMPR_HANDLE GroupHandle,
  [in] unsigned long MemberId,
  [in] unsigned long Attributes
);
```

GroupHandle: An RPC context handle, as specified in section 2.2.3.2, representing a group object.

MemberId: A RID representing an account to add to the group's membership list.

Attributes: The characteristics of the membership relationship. See section 2.2.1.10 for legal values and semantics.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

Upon receiving this message, the responder MUST process the data from the message subject to the following constraints:

1. *GroupHandle* MUST have the required access specified in section 3.2.2.1.
2. All database operations MUST occur in a single transaction.
3. Let G be the group referenced by *GroupHandle*.

4. Let TargetSid be the SID composed by making the MemberId a suffix to the domain prefix of G's objectSid.

5. If there is no object whose **objectSid** attribute is TargetSid, the responder MUST return STATUS_NO_SUCH_USER.

6. If G's member attribute already has as a dsname value that references the object whose **objectSid** is TargetSid, the responder MUST return an error.

7. G's member attribute MUST be updated to add a dsname value that references the object with the **objectSid** value TargetSid.

8. The message processing specified in section 3.2.5.14.7 for the *Attributes* parameter MUST be adhered to.

9. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.

3.2.5.8.2 SamrRemoveMemberFromGroup (Opnum 24)

The **SamrRemoveMemberFromGroup** method removes a member from a group.

```
long SamrRemoveMemberFromGroup(
  [in] SAMPR_HANDLE GroupHandle,
  [in] unsigned long MemberId
);
```

GroupHandle: An RPC context handle, as specified in section 2.2.3.2, representing a group object.

MemberId: A RID representing an account to remove from the group's membership list.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

Upon receiving this message, the responder MUST process the data from the message subject to the following constraints:

1. *GroupHandle* MUST have the required access specified in section 3.2.2.1.

2. All database operations MUST occur in a single transaction.

3. Let G be the group referenced by the *GroupHandle*.

4. Let TargetSid be the SID composed by making the MemberId a suffix to the domain prefix of G's objectSid.

5. If G's member attribute does not have a dsname value that references the object whose **objectSid** is TargetSid, the responder MUST return an error.

6. G's member attribute MUST be updated to remove a dsname value that references the object with the **objectSid** value TargetSid.

7. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.

3.2.5.8.3 SamrGetMembersInGroup (Opnum 25)

The **SamrGetMembersInGroup** method reads the members of a group.

```
long SamrGetMembersInGroup(
  [in] SAMPR_HANDLE GroupHandle,
  [out] PSAMPR_GET_MEMBERS_BUFFER* Members
);
```

GroupHandle: An RPC context handle, as specified in section 2.2.3.2, representing a group object.

Members: A structure containing an array of RIDs, as well as an array of attribute values.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

Upon receiving this message, the responder MUST process the data from the message subject to the following constraints:

1. *GroupHandle* MUST have the required access specified in section 3.2.2.1.
2. Let G be the group object referenced by *GroupHandle*.
3. Let M be the set of values of G's member attribute such that the groupType of the object referenced by each value is GROUP_TYPE_SECURITY_ACCOUNT.
4. On output:
    1. **Members.MemberCount** MUST be equal to the number of values in M.
    2. The **Members.Members** array MUST contain the RelativeId's of the **objectSid** attribute values for all objects in set M.
    3. For each element, see section 3.2.5.14.7 for a message processing specification of the **Attributes** field.
5. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.

3.2.5.8.4 SamrAddMemberToAlias (Opnum 31)

The **SamrAddMemberToAlias** method adds a member to an alias.

```
long SamrAddMemberToAlias(
  [in] SAMPR_HANDLE AliasHandle,
  [in] PRPC_SID MemberId
);
```

AliasHandle: An RPC context handle, as specified in section 2.2.3.2, representing an alias object.

MemberId: The SID of an account to add to the alias.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

Upon receiving this message, the responder MUST process the data from the message subject to the following constraints:

1. *AliasHandle* MUST have the required access specified in section 3.2.2.1.
2. All database operations MUST occur in a single transaction.
3. Let A be the alias referenced by *AliasHandle*.
4. If the domain prefix of MemberId is the same domain prefix as the account domain and there is no object whose **objectSid** attribute is MemberId, the responder MUST return an error.
5. If A's member attribute already has a dsname value that references the object whose **objectSid** is MemberId, the responder MUST return an error.
6. A's member attribute MUST be updated to add a dsname value that references the object with the **objectSid** value MemberId.
7. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.

3.2.5.8.5 SamrRemoveMemberFromAlias (Opnum 32)

The **SamrRemoveMemberFromAlias** method removes a member from an alias.

```
long SamrRemoveMemberFromAlias(
  [in] SAMPR_HANDLE AliasHandle,
  [in] PRPC_SID MemberId
);
```

AliasHandle: An RPC context handle, as specified in section 2.2.3.2, representing an alias object.

MemberId: The SID of an account to remove from the alias.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

Upon receiving this message, the responder MUST process the data from the message subject to the following constraints:

1. *AliasHandle* MUST have the required access specified in section 3.2.2.1.
2. All database operations MUST occur in a single transaction.
3. Let A be the alias object referenced by the *AliasHandle*.
4. If A's member attribute does not have a dsname value that references the object whose **objectSid** is MemberId, the responder MUST return an error.

5. A's member attribute MUST be updated to remove a dsname value that references the object with the **objectSid** value MemberId.

6. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.

3.2.5.8.6 SamrGetMembersInAlias (Opnum 33)

The **SamrGetMembersInAlias** method obtains the membership list of an alias.

```
long SamrGetMembersInAlias(
  [in] SAMPR_HANDLE AliasHandle,
  [out] PSAMPR_PSID_ARRAY Members
);
```

AliasHandle: An RPC context handle, as specified in section 2.2.3.2, representing an alias object.

Members: A structure containing an array of SIDs that represent the membership list of the alias referenced by *AliasHandle*.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

Upon receiving this message, the responder MUST process the data from the message subject to the following constraints:

1. *AliasHandle* MUST have the required access specified in section 3.2.2.1.

2. On output, **Members.Count** MUST be equal to the number of values in the member attribute, and **Members.Sids** MUST have **Member.Count** number of elements. Each element MUST contain the **objectSid** value of the object referenced in the member attribute.

3. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.

3.2.5.8.7 SamrRemoveMemberFromForeignDomain (Opnum 45)

The **SamrRemoveMemberFromForeignDomain** method removes a member from all aliases.

```
long SamrRemoveMemberFromForeignDomain(
  [in] SAMPR_HANDLE DomainHandle,
  [in] PRPC_SID MemberSid
);
```

DomainHandle: An RPC context handle, as specified in section 2.2.3.2, representing a domain object.

MemberSid: The SID to remove from the membership.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

Upon receiving this message, the responder MUST process the data from the message subject to the following constraints:

1. *DomainHandle* MUST have the required access specified in section 3.2.2.1.
2. All database operations MUST occur in a single transaction.
3. For all alias objects in the domain referenced by *DomainHandle*, the responder MUST remove any member value that references the object with the **objectSid** attribute value of MemberSid.
4. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.

3.2.5.8.8 SamrAddMultipleMembersToAlias (Opnum 52)

The **SamrAddMultipleMembersToAlias** method adds multiple members to an alias.

```
long SamrAddMultipleMembersToAlias(
  [in] SAMPR_HANDLE AliasHandle,
  [in] PSAMPR_PSID_ARRAY MembersBuffer
);
```

AliasHandle: An RPC context handle, as specified in section 2.2.3.2, representing an alias object.

MembersBuffer: A structure containing a list of SIDs to add as members to the alias.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

The responder MUST behave as with N successive message calls to **SamrAddMemberToAlias**, once for each SID value in *MembersBuffer*, where *MembersBuffer* contains N elements. The responder MUST ignore the processing error of a member value already being present in the member attribute and abort the request on any other processing error.

3.2.5.8.9 SamrRemoveMultipleMembersFromAlias (Opnum 53)

The **SamrRemoveMultipleMembersFromAlias** method removes multiple members from an alias.

```
long SamrRemoveMultipleMembersFromAlias(
  [in] SAMPR_HANDLE AliasHandle,
  [in] PSAMPR_PSID_ARRAY MembersBuffer
);
```

AliasHandle: An RPC context handle, as specified in section 2.2.3.2, representing an alias object.

MembersBuffer: A structure containing a list of SIDs to remove from the alias's membership list.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

The responder MUST behave as with N successive message calls to **SamrRemoveMemberFromAlias**, once for each SID value in *MembersBuffer*, where *MembersBuffer* contains N elements. The responder MUST ignore the processing error triggered by a value not existing in the member attribute's values, and abort the request on any other processing error.

3.2.5.9 Membership-Of Pattern

These methods enable a requester to obtain the group membership of a user or the alias membership of a set of SIDs. In mixed mode domains, these methods are useful in approximating the authorization data associated with an authentication request for a given user. However, in native mode domains, these methods are not accurate because the authorization building process is more complex than what these methods enable. This means that in native mode domains, these methods MUST NOT be used to approximate the authorization data for a given user accessing a resource.

A requester must first obtain a handle to the user or domain, depending on the method.

See section 1.3 for a description of the "membership-of" pattern of methods.

3.2.5.9.1 SamrGetGroupsForUser (Opnum 39)

The **SamrGetGroupsForUser** method obtains a listing of groups that a user is a member of.

```
long SamrGetGroupsForUser(
  [in] SAMPR_HANDLE UserHandle,
  [out] PSAMPR_GET_GROUPS_BUFFER* Groups
);
```

UserHandle: An RPC context handle, as specified in section 2.2.3.2, representing a user object.

Groups: An array of RIDs of the groups that the user referenced by *UserHandle* is a member of.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

Upon receiving this message, the responder MUST process the data from the message subject to the following constraints:

1. *UserHandle* MUST have the required access specified in section 3.2.2.1.
2. The responder MUST determine the union of all database objects with class group and groupType GROUP_TYPE_SECURITY_ACCOUNT whose member value contains the SID of the user referenced by *UserHandle*.
3. The returned **Groups.MemberCount** MUST be set to the cardinality that the union determined from step 2.
4. For each group in the union determined from step 2, the responder MUST set a corresponding element in **Groups.Group** as follows:
    1. RelativeId MUST contain the RID of the SID of the dsname member value.
    2. Set the Attributes field according to the message processing rules in section 3.2.5.14.7.

3.2.5.9.2 SamrGetAliasMembership (Opnum 16)

The **SamrGetAliasMembership** method obtains the union of all aliases that a given set of SIDs is a member of.

```
long SamrGetAliasMembership(
  [in] SAMPR_HANDLE DomainHandle,
  [in] PSAMPR_PSID_ARRAY SidArray,
  [out] PSAMPR_ULONG_ARRAY Membership
);
```

DomainHandle: An RPC context handle, as specified in section 2.2.3.2, representing a domain object.

SidArray: A list of SIDs.

Membership: The union of all aliases (represented by RIDs) that all SIDs in SidArray are a member of.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

Upon receiving this message, the responder MUST process the data from the message subject to the following constraints:

1. *DomainHandle* MUST have the required access specified in section 3.2.2.1.
2. For each SID value in SidArray, the responder MUST determine the union of all database objects in the domain referenced by *DomainHandle* with class group and groupType GROUP_TYPE_SECURITY_RESOURCE whose member value contains the SID.
3. The returned Membership parameter MUST contain the RIDs of the **objectSid** attribute of the union of all groups found by constraint 2.

3.2.5.10 Change Password Pattern

The "change password" methods enable a requester to change the password of a user object. All these methods require that the requester has knowledge of the current password in order for the message to be processed successfully.

It is important to note that **SamrChangePasswordUser** requires a handle to a user object (obtained through an "open" or "create" method, sections 3.2.5.1 and 3.2.5.4) and therefore requires an authentication connect. **SamrUnicodeChangePasswordUser2** and **SamrOemChangePasswordUser2** do not require any handle, and can be sent directly to the targeted responder using no security or by authenticating as anonymous. This characteristic allows end users, whose passwords have expired and therefore cannot logon, to change their passwords without an authenticated connection. See section 1.3 for a description of the "change password" pattern of methods.

In the following descriptions, when a value is said to be "presented by the requester", that value is provided by the requester side of the protocol. In a canonical password-change scenario, an end user enters his or her old and new passwords into a password-change application. That application acts as a requester for this method.

To encrypt password data, these methods use the fact that the requester (an end user in the canonical scenario) and the responder (a DC in the canonical scenario) share a common secret: the user's existing password. The LM and/or NT hash (specified in the following sections) of the existing password's cleartext value is used as an encryption key. Because the DC stores the existing password as well, the DC is able to decrypt the data sent by the requester. Of course, if the end user did not enter the correct existing password, the decryption does not result in meaningful data, and an error is returned.

SamrUnicodeChangePasswordUser2 is preferred if the Unicode-encoded cleartext password is available to the requester.

3.2.5.10.1 SamrChangePasswordUser (Opnum 38)

The **SamrChangePasswordUser** method changes the password of a user object.

```
long SamrChangePasswordUser(
  [in] SAMPR_HANDLE UserHandle,
  [in] unsigned char LmPresent,
  [in, unique] PENCRYPTED_LM_OWF_PASSWORD LmOldEncryptedWithLmNew,
  [in, unique] PENCRYPTED_LM_OWF_PASSWORD LmNewEncryptedWithLmOld,
  [in] unsigned char NtPresent,
  [in, unique] PENCRYPTED_NT_OWF_PASSWORD NtOldEncryptedWithNtNew,
  [in, unique] PENCRYPTED_NT_OWF_PASSWORD NtNewEncryptedWithNtOld,
  [in] unsigned char NtCrossEncryptionPresent,
  [in, unique] PENCRYPTED_NT_OWF_PASSWORD NtNewEncryptedWithLmNew,
  [in] unsigned char LmCrossEncryptionPresent,
  [in, unique] PENCRYPTED_LM_OWF_PASSWORD LmNewEncryptedWithNtNew
);
```

UserHandle: An RPC context handle, as specified in section 2.2.3.2, representing a user object.

LmPresent: If this parameter is zero, the *LmOldEncryptedWithLmNew* and *LmNewEncryptedWithLmOld* fields MUST be ignored by the responder; otherwise these fields MUST be processed.

LmOldEncryptedWithLmNew: The LM hash of the target user's existing password (as presented by the requester) encrypted per the specification of **ENCRYPTED_LM_OWF_PASSWORD (section 2.2.3.3)**, where the key is the LM hash of the new password for the target user (as presented by the requester in the *LmNewEncryptedWithLmOld* parameter).

LmNewEncryptedWithLmOld: The LM hash of the target user's new password (as presented by the requester) encrypted per the specification of **ENCRYPTED_LM_OWF_PASSWORD**, where the key is the LM hash of the existing password for the target user (as presented by the requester in the *LmOldEncryptedWithLmNew* parameter).

NtPresent: If this parameter is zero, *NtOldEncryptedWithNtNew* and *NtNewEncryptedWithNtOld* MUST be ignored by the responder; otherwise these fields MUST be processed.

NtOldEncryptedWithNtNew: The NT hash of the target user's existing password (as presented by the requester) encrypted per the specification of **ENCRYPTED_NT_OWF_PASSWORD** (section 2.2.3.3), where the key is the NT hash of the new password for the target user (as presented by the requester).

NtNewEncryptedWithNtOld: The NT hash of the target user's new password (as presented by the requester) encrypted per the specification of **ENCRYPTED_NT_OWF_PASSWORD**, where the key is the NT hash of the existing password for the target user (as presented by the requester).

NtCrossEncryptionPresent: If this parameter is zero, *NtNewEncryptedWithLmNew* MUST be ignored; otherwise, this field MUST be processed.

NtNewEncryptedWithLmNew: The NT hash of the target user's new password (as presented by the requester) encrypted per the specification of **ENCRYPTED_NT_OWF_PASSWORD**, where the key is the LM hash of the new password for the target user (as presented by the requester).

LmCrossEncryptionPresent: If this parameter is zero, *LmNewEncryptedWithNtNew* MUST be ignored; otherwise, this field MUST be processed.

LmNewEncryptedWithNtNew: The LM hash of the target user's new password (as presented by the requester) encrypted per the specification of **ENCRYPTED_LM_OWF_PASSWORD**, where the key is the NT hash of the new password for the target user (as presented by the requester).

The processing for this method is quite complex. To aid comprehension, a brief, non-normative description of the method's intent follows.

The method requires that the requester presents both the NT and LM hash of the new password (and will fail otherwise). However, because the old password might not be stored in either the NT or LM hash format on the receiver, and thus the new hash values cannot be decrypted using the old hash values, the method allows for the new NT and LM hashes to be "cross-encrypted" using the new LM or NT hash value (instead of the old values). As such, there are three combinations that can lead to successful processing, as listed below.

1. *NtPresent* is nonzero, *LmPresent* is nonzero, and both the LM and NT hashes are present in the database. No "cross-encryption" is required. The cross-encryption–related parameters are ignored.

2. *LmPresent* is nonzero, *NtCrossEncryptionPresent* is nonzero, and the LM hash is present in the database. This combination is used when the NT hash is not stored at the responder; the requester can send the NT hash encrypted with the new LM hash instead. The NT-hash–related parameters are ignored.

3. *NtPresent* is nonzero, *LmCrossEncryptionPresent* is nonzero, and the NT hash is present in the database. This combination is used when the LM hash is not stored at the responder; the requester can send the LM hash encrypted with the new NT hash instead. The LM-hash–related parameters are ignored.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

Upon receiving this message, the responder MUST process the data from the message subject to the following constraints applied in the presented order:

1. All database operations MUST occur in a single transaction.

2. The constraints in section 3.2.5.14.5 MUST be satisfied.

3. If *LmPresent* is nonzero and *LmNewEncryptedWithLmOld* or *LmOldEncryptedWithLmNew* is zero, the responder MUST return an error.

4. If *NtPresent* is nonzero and *NtNewEncryptedWithNtOld* or *NtOldEncryptedWithNtNew* is zero, the responder MUST return an error.

5. If *NtCrossEncryptionPresent* is nonzero and *NtNewEncryptedWithLmNew* is zero, the responder MUST return an error.

6. If *LmCrossEncryptionPresent* is nonzero and *LmNewEncryptedWithNtNew* is zero, the responder MUST return an error.

7. If *LmPresent* and *NtPresent* are zero, the responder MUST return an error.

8. Let U be the user account referenced by *UserHandle*.

9. Let Stored-LM-Hash be the value of the **dbcsPwd** attribute from the database decrypted using the algorithm specified in section 2.2.11.1, using U's RelativeId (an unsigned integer) as the key. If the **dbcsPwd** attribute does not exist, let Stored-LM-Hash be "NULL".

10. Let Stored-NT-Hash be the value of the **unicodePwd** attribute from the database decrypted using the algorithm specified in section 2.2.11.1, using U's RelativeId (an unsigned integer) as the key. If the **unicodePwd** attribute does not exist, let Stored-NT-Hash be "NULL".

11. If *LmPresent* is nonzero and Stored-LM-Hash is not NULL, let Presented-New-LM-Hash be *LmNewEncryptedWithLmOld*, decrypted per the specification in section 2.2.11.1, using Stored-LM-Hash as the key; and let Presented-Old-LM-Hash be *LmOldEncryptedWithLmNew*, decrypted per the specification in section 2.2.11.1, using Presented-New-LM-Hash as the key. The values are not referenced below if *LmPresent* is zero.

12. If *NtPresent* is nonzero and Stored-NT-Hash is not NULL, let Presented-New-NT-Hash be *NtNewEncryptedWithNtOld*, decrypted per the specification in section 2.2.11.1 using Stored-NT-Hash as the key; and let Presented-Old-NT-Hash be *NtOldEncryptedWithNtNew*, decrypted per the specification in section 2.2.11.1 using Presented-New-NT-Hash as the key. The values are not referenced below if *NtPresent* is zero.

13. If all of the following conditions are true, the responder MUST abort processing and return the error status STATUS_LM_CROSS_ENCRYPTION_REQUIRED.

    1. *NtPresent* is nonzero.

    2. *LmPresent* is zero.

    3. *LmCrossEncryptionPresent* is zero.

    4. Stored-NT-Hash is non-NULL and equals Presented-Old-NT-Hash.

14. If all of the following conditions are true, the responder MUST abort processing and return the error status STATUS_NT_CROSS_ENCRYPTION_REQUIRED.

    1. *NtPresent* is zero.

    2. *LmPresent* is nonzero.

    3. *NtCrossEncryptionPresent* is zero.

    4. Stored-NT-Hash is NULL.

5. Stored-LM-Hash is non-NULL and equals Presented-Old-LM-Hash.

15. Exactly one of the three following conditions MUST be true; otherwise the responder MUST satisfy the constraints in section 3.2.5.14.6 and then return STATUS_WRONG_PASSWORD.

    1. *LmPresent* is nonzero, Stored-LM-Hash is non-NULL and equals Presented-Old-LM-Hash, *NtPresent* is nonzero, Stored-NT-Hash is non-NULL, and Stored-NT-Hash equals Presented-Old-NT-Hash.

    2. *LmPresent* is nonzero, Stored-LM-Hash is non-NULL and equals Presented-Old-LM-Hash, *NtPresent* is zero, Stored-NT-Hash is NULL.

    3. *NtPresent* is nonzero, Stored-NT-Hash is non-NULL and equals Presented-Old-NT-Hash, *LmPresent* is zero, Stored-LM-Hash is NULL.

16. If *LmPresent* is nonzero, the **dbcsPwd** attribute MUST be updated with Presented-New-LM-Hash.

17. If *LmPresent* is zero and *LmCrossEncryptionPresent* is nonzero, the **dbcsPwd** attribute MUST be updated with the value of *LmNewEncryptedWithNtNew*, decrypted using the algorithm specified in section 2.2.11.1, using Presented-New-NT-Hash as the decryption key.

18. If *NtPresent* is nonzero, the **unicodePwd** attribute MUST be updated with Presented-New-NT-Hash.

19. If *NtPresent* is zero and *NtCrossEncryptionPresent* is nonzero, the **unicodePwd** attribute MUST be updated with the value of *NtNewEncryptedWithLmNew*, decrypted using the algorithm specified in section 2.2.11.1, using Presented-New-LM-Hash as the decryption key.

20. On database error, the responder MUST return the data error; on general processing error, the responder MUST return STATUS_WRONG_PASSWORD; otherwise, return STATUS_SUCCESS.

3.2.5.10.2 SamrOemChangePasswordUser2 (Opnum 54)

The **SamrOemChangePasswordUser2** method changes a user's password.

```
long SamrOemChangePasswordUser2(
  [in] handle_t BindingHandle,
  [in, unique] PRPC STRING ServerName,
  [in] PRPC STRING UserName,
  [in, unique] PSAMPR ENCRYPTED USER PASSWORD NewPasswordEncryptedWithOldLm,
  [in, unique] PENCRYPTED_LM_OWF_PASSWORD OldLmOwfPasswordEncryptedWithNewLm
);
```

BindingHandle: An RPC binding handle parameter as specified in [C706-Ch2Intro].

ServerName: A counted string, encoded in the OEM character set, containing the NETBIOS name of the responder; this parameter MAY<55> be ignored by the responder.

UserName: A counted string, encoded in the OEM character set, containing the name of the user whose password is to be changed; see message processing below for details on how this value is used as a database key to locate the account that is the target of this password change operation.

NewPasswordEncryptedWithOldLm: A cleartext password encrypted per the specification of **SAMPR_ENCRYPTED_USER_PASSWORD (section 2.2.7.21)**, where the key is the LM hash of the existing password for the target user (as presented by the requester). The

cleartext password MUST be encoded in an OEM code page character set (as opposed to UTF-16).

OldLmOwfPasswordEncryptedWithNewLm: The LM hash of the target user's existing password (as presented by the requester) encrypted per the specification of **ENCRYPTED_LM_OWF_PASSWORD (section 2.2.3.3)**, where the key is the LM hash of the cleartext password obtained from decrypting *NewPasswordEncryptedWithOldLm* (see description above for decryption details).

Upon receiving this message, the responder MUST process the data from the message subject to the following constraints:

1. All database operations MUST occur in a single transaction.
2. The responder MUST encode the *UserName* parameter into UTF-16 using the OEM code page.
3. Let U be the user account with the **sAMAccountName** attribute value of *UserName*. The responder MUST return STATUS_WRONG_PASSWORD if no such account exists.
4. The constraints in section 3.2.5.14.5 MUST be satisfied.
5. Let Stored-LM-Hash be the value of the **dbcsPwd** attribute from the database decrypted using the algorithm specified in section 2.2.11.1, using U's RelativeId as the key. If this attribute does not exist, STATUS_WRONG_PASSWORD MUST be returned.
6. Let Presented-Clear-Text be the cleartext value sent by the requester. This value is obtained by decrypting *NewPasswordEncryptedWithOldLm* per the specification of **SAMPR_ENCRYPTED_USER_PASSWORD** using Stored-LM-Hash as the key, and then translating the result into a UTF-16 encoded string (using the OEM code page).
7. Let Presented-Old-LM-Hash be the value of *OldLmOwfPasswordEncryptedWithNewLm* that has been decrypted per the specification of **ENCRYPTED_LM_OWF_PASSWORD**, using the LM hash of Presented-Clear-Text as the key.
8. If Presented-Old-LM-Hash is not equal to Stored-LM-Hash, the responder MUST satisfy the constraints in section 3.2.5.14.6, abort processing, and return STATUS_WRONG_PASSWORD.
9. The responder MUST update the **clearTextPassword** attribute with Presented-Clear-Text.
10. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.

3.2.5.10.3 SamrUnicodeChangePasswordUser2 (Opnum 55)

The **SamrUnicodeChangePasswordUser2** method changes a user account's password.

```
long SamrUnicodeChangePasswordUser2(
  [in] handle_t BindingHandle,
  [in, unique] PRPC_UNICODE_STRING ServerName,
  [in] PRPC_UNICODE_STRING UserName,
  [in, unique] PSAMPR_ENCRYPTED_USER_PASSWORD NewPasswordEncryptedWithOldNt,
  [in, unique] PENCRYPTED_NT_OWF_PASSWORD OldNtOwfPasswordEncryptedWithNewNt,
  [in] unsigned char LmPresent,
  [in, unique] PSAMPR_ENCRYPTED_USER_PASSWORD NewPasswordEncryptedWithOldLm,
  [in, unique] PENCRYPTED_LM_OWF_PASSWORD OldLmOwfPasswordEncryptedWithNewNt
);
```

BindingHandle: An RPC binding handle parameter as specified in [C706-Ch2Intro].

ServerName: A null-terminated string containing the NETBIOS name of the responder; this parameter MAY<56> be ignored by the responder.

UserName: The name of the user. See the message processing below for details on how this value is used as a database key to locate the account that is the target of this password change operation.

NewPasswordEncryptedWithOldNt: A cleartext password encrypted per the specification of **SAMPR_ENCRYPTED_USER_PASSWORD (section 2.2.7.21)**, where the key is the NT hash of the existing password for the target user (as presented by the requester in the *OldNtOwfPasswordEncryptedWithNewNt* parameter).

OldNtOwfPasswordEncryptedWithNewNt: The NT hash of the target user's existing password (as presented by the requester) encrypted per the specification of **ENCRYPTED_LM_OWF_PASSWORD (section 2.2.3.3)**, where the key is the NT hash of the cleartext password obtained from decrypting *NewPasswordEncryptedWithOldNt*.

LmPresent: If this parameter is zero, *NewPasswordEncryptedWithOldLm* and *OldLmOwfPasswordEncryptedWithOldLm* MUST be ignored; otherwise these fields MUST be processed.

NewPasswordEncryptedWithOldLm: A cleartext password encrypted per the specification of **SAMPR_ENCRYPTED_USER_PASSWORD**, where the key is the LM hash of the existing password for the target user (as presented by the requester).

OldLmOwfPasswordEncryptedWithNewNt: The LM hash the target user's existing password (as presented by the requester) encrypted per the specification of **ENCRYPTED_LM_OWF_PASSWORD**, where the key is the NT hash of the cleartext password obtained from decrypting *NewPasswordEncryptedWithOldNt*.

Upon receiving this message, the responder MUST process the data from the message subject to the following constraints:

1. All database operations MUST occur in a single transaction.
2. Let U be the user account with the **sAMAccountName** attribute value of UserName. The responder MUST return STATUS_WRONG_PASSWORD, if no such account exists.
3. The constraints in section 3.2.5.14.5 MUST be satisfied.
4. Let Stored-NT-Hash be the value of the **unicodePwd** attribute from the database decrypted using the algorithm specified in section 2.2.11.1, using U's *RelativeId* as the key. If the attribute does not exist, let Stored-NT-Hash be "NULL".
5. Let Stored-LM-Hash be the value of the **dbcsPwd** attribute from the database decrypted using the algorithm specified in section 2.2.11.1, using U's *RelativeId* as the key. If the attribute does not exist, let Stored-LM-Hash be "NULL".
6. If Stored-NT-Hash is NULL, and LmPresent is zero or Stored-LM-Hash is NULL, the responder MUST abort processing and return STATUS_WRONG_PASSWORD.
7. If Stored-NT-Hash is not NULL, then:

1. Let Presented-Clear-Text be the cleartext value sent by the requester, obtained by decrypting *NewPasswordEncryptedWithOldNt* per the specification of **SAMPR_ENCRYPTED_USER_PASSWORD**, using Stored-NT-Hash as the key, AND

2. Let Presented-Old-NT-Hash be the value of *OldNtOwfPasswordEncryptedWithNewNt* decrypted per the specification of **ENCRYPTED_LM_OWF_PASSWORD**, using the NT hash of Presented-Clear-Text as the key.

8. If Stored-NT-Hash is NULL, then:

   1. Let Presented-Clear-Text be the cleartext value sent by the requester, obtained by decrypting *NewPasswordEncryptedWithOldLm* per the specification of **SAMPR_ENCRYPTED_USER_PASSWORD**, using Stored-LM-Hash as the key, AND

   2. Let Presented-Old-LM-Hash be the value of *OldLmOwfPasswordEncryptedWithNewNt* decrypted per the specification of **ENCRYPTED_LM_OWF_PASSWORD**, using the NT hash of Presented-Clear-Text as the key.

9. Exactly one of the two following conditions MUST be true; otherwise, the responder MUST satisfy the constraints in section 3.2.5.14.6 and return STATUS_WRONG_PASSWORD.

   1. Stored-NT-Hash is non-NULL and equals Presented-Old-NT-Hash.

   2. Stored-NT-Hash is NULL, Stored-LM-Hash is non-NULL and equals Presented-Old-LM-Hash.

10. The responder MUST update the **clearTextPassword** attribute with Presented-Clear-Text.

11. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.

3.2.5.10.4 SamrUnicodeChangePasswordUser3 (Opnum 63)

The **SamrUnicodeChangePasswordUser3** method changes a user account's password. This method is similar to **SamrUnicodeChangePasswordUser2**, except that it includes a password change failure message and an extra parameter to transmit policy information in the event of a password restriction.<57>

This method is reserved.<58>

```
long SamrUnicodeChangePasswordUser3(
  [in] handle t BindingHandle,
  [in, unique] PRPC UNICODE STRING ServerName,
  [in] PRPC_UNICODE_STRING UserName,
  [in, unique] PSAMPR_ENCRYPTED_USER_PASSWORD NewPasswordEncryptedWithOldNt,
  [in, unique] PENCRYPTED NT OWF PASSWORD OldNtOwfPasswordEncryptedWithNewNt,
  [in] unsigned char LmPresent,
  [in, unique] PSAMPR ENCRYPTED USER PASSWORD NewPasswordEncryptedWithOldLm,
  [in, unique] PENCRYPTED LM OWF PASSWORD OldLmOwfPassswordEncryptedWithNewLmOrNt,
  [in, unique] PSAMPR_ENCRYPTED_USER_PASSWORD AdditionalData,
  [out] PDOMAIN_PASSWORD_INFORMATION* EffectivePasswordPolicy,
  [out] PUSER PWD CHANGE FAILURE INFORMATION* PasswordChangeInfo
);
```

BindingHandle: An RPC binding handle parameter as specified in [C706-Ch2Intro].

ServerName: A null-terminated string containing the NETBIOS name of the responder; this parameter MAY<59> be ignored by the responder.

UserName: The name of the user.

NewPasswordEncryptedWithOldNt: A cleartext password encrypted per the specification of **SAMPR_ENCRYPTED_USER_PASSWORD (section 2.2.7.21)**, where the key is the NT hash of the existing password for the target user (as presented by the requester in the *OldNtOwfPasswordEncryptedWithNewNt* parameter).

OldNtOwfPasswordEncryptedWithNewNt: The NT hash of the target user's existing password (as presented by the requester) encrypted per the specification of **ENCRYPTED_NT_OWF_PASSWORD**, where the key is the NT hash of the cleartext password obtained from decrypting *NewPasswordEncryptedWithOldNt*.

LmPresent: If this parameter is zero, *NewPasswordEncryptWithOldLm* and *OldLmOwfPasswordEncryptedWithOldLm* MUST be ignored; otherwise these fields MUST be processed.

NewPasswordEncryptedWithOldLm: A cleartext password encrypted per the specification of **SAMPR_ENCRYPTED_USER_PASSWORD**, where the key is the LM hash of the existing password for the target user (as presented by the requester).

OldLmOwfPasswordEncryptedWithNewLmOrNt: The LM hash of the target user's existing password (as presented by the requester) encrypted per the specification of **ENCRYPTED_LM_OWF_PASSWORD**, where the key is the NT hash of the cleartext password obtained from decrypting *NewPasswordEncryptedWithOldNt*.

AdditionalData: Not used.**Note** MUST be set to NULL and ignored on receipt.

EffectivePasswordPolicy: Pointer to a pointer that points to a **USER_DOMAIN_PASSWORD_INFORMATION** structure that contains the password policy. This pointer is valid only if the password change fails due to a password restriction; otherwise, it is NULL.

PasswordChangeInfo: Pointer to a pointer that points to a **USER_PWD_CHANGE_FAILURE_INFORMATION** structure that contains information regarding the failure to change the password. This pointer is valid only if the password change fails due to a password restriction; otherwise, it is NULL.

3.2.5.11 Lookup Pattern

These methods enable a requester to translate from a security ID (either a SID or a RID) to a user-friendly name, and vice versa. This action is useful when an end user is setting access control via a security descriptor. However, the translation methods specified in [MS-LSAT] sections **3.1.4.5** and **3.1.4.9** are superior because they translate a wider range of SIDs.

A requester must first obtain a handle to the object of interest through an "open" method. See section 3.2.5.1.

See section 1.3 for a description of the "lookup" pattern of methods.

3.2.5.11.1 SamrLookupDomainInSamServer (Opnum 5)

The **SamrLookupDomainInSamServer** method obtains the SID of a domain object, given its name.

```
long SamrLookupDomainInSamServer(
  [in] SAMPR_HANDLE ServerHandle,
```

```
  [in] PRPC_UNICODE_STRING Name,
  [out] PRPC_SID* DomainId
);
```

ServerHandle: An RPC context handle, as specified in section 2.2.3.2, representing a server object.

Name: A UTF-16 encoded string.

DomainId: A SID value of a domain that matches the *Name* passed in. The match must be exact (no wildcards are permitted). See message processing below for more details.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

Upon receiving this message, the responder MUST process the data from the message subject to the following constraints.

1. *ServerHandle* MUST have the required access specified in section 3.2.2.1.
2. If the *Name* input parameter matches an attribute value per the following table, the associated value in the "Return attribute" column MUST be returned via the *DomainId* parameter.

Matching object	Matching attribute	Return object	Return attribute
domain object	name	domain object	objectSid
built-in object	name	built-in object	objectSid

3. If there is no match, an error MUST be returned.
4. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.

3.2.5.11.2 SamrLookupNamesInDomain (Opnum 17)

The **SamrLookupNamesInDomain** method translates a set of account names into a set of RIDs.

```
long SamrLookupNamesInDomain(
  [in] SAMPR_HANDLE DomainHandle,
  [in, range(0,1000)] unsigned long Count,
  [in, size_is(1000), length_is(Count)]
    RPC_UNICODE_STRING Names[*],
  [out] PSAMPR_ULONG_ARRAY RelativeIds,
  [out] PSAMPR_ULONG_ARRAY Use
);
```

DomainHandle: An RPC context handle, as specified in section 2.2.3.2, representing a domain object.

Count: The number of elements in *Names*. The maximum value of 1,000 is chosen to limit the amount of memory that the requester can force the responder to allocate.

Names: An array of strings that are to be mapped to RIDs.

RelativeIds: An array of RIDs of accounts that correspond to the elements in *Names*.

Use: An array of SID_NAME_USE enumeration values that describe the type of account for each entry in *RelativeIds*.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

On receiving this message, the responder MUST process the data from the message subject to the following constraints:

1. *DomainHandle* MUST have the required access specified in section 3.2.2.1.
2. Let U be the set of all database objects whose **objectSid**'s domain prefix matches the domain prefix of the domain referenced by *DomainHandle*.
3. For each element in *Names* that matches a database object's **sAMAccountName** attribute value in the set U, the requester MUST fill in *RelativeIds* and *Use* as follows:
    1. Let 'i' be the current element of *Names*.
    2. **RelativeIds.Element[i]** is the RID of the matched object's **objectSid** attribute value.
    3. **Use.Element[i]** is set as follows:

objectClass	GroupType	Use
User	n/a	SidTypeUser
Group	GROUP_TYPE_ACCOUNT_GROUP	SidTypeGroup
Group	GROUP_TYPE_UNIVERSAL_GROUP	SidTypeGroup
Group	Any value not matching above criteria for Group	SidTypeAlias

4. For each element in *Names* that does not match a database object's **sAMAccountName** attribute value in the set U, the requester MUST fill in *RelativeIds* and *Use* as follows:
    1. Let 'i' be the current element of *Names*.
    2. RelativeIds.Element[i] is 0.
    3. **Use.Element[i]** is SidTypeUnknown.
5. The responder MUST return a database processing error if any occurred; otherwise:
    1. **RelativeIds.Count** MUST be set to the input parameter *Count* on successful completion of the method.
    2. **Use.Count** MUST be set to the input parameter *Count* on successful completion of the method.
    3. If the number of matched accounts is equal to the input parameter *Count*, STATUS_SUCCESS MUST be returned.

4. If the number of matched accounts is less than the input parameter *Count*, but greater than 0, STATUS_SOME_NOT_MAPPED MUST be returned. Note that this is not an error condition.

5. If the number of matched accounts is 0, STATUS_NONE_MAPPED MUST be returned.

3.2.5.11.3 SamrLookupIdsInDomain (Opnum 18)

The **SamrLookupIdsInDomain** method translates a set of RIDs into account names.

```
long SamrLookupIdsInDomain(
  [in] SAMPR_HANDLE DomainHandle,
  [in, range(0,1000)] unsigned long Count,
  [in, size_is(1000), length_is(Count)]
    unsigned long* RelativeIds,
  [out] PSAMPR_RETURNED_USTRING_ARRAY Names,
  [out] PSAMPR_ULONG_ARRAY Use
);
```

DomainHandle: An RPC context handle, as specified in section 2.2.3.2, representing a domain object.

Count: The number of elements in *RelativeIds*. The maximum value of 1,000 is chosen to limit the amount of memory that the requester can force the responder to allocate.

RelativeIds: An array of RIDs that are to be mapped to account names.

Names: A structure containing an array of account names that correspond to the elements in *RelativeIds*.

Use: A structure containing an array of SID_NAME_USE enumeration values that describe the type of account for each entry in *RelativeIds*.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

On receiving this message, the responder MUST process the data from the message subject to the following constraints:

1. *DomainHandle* MUST have the required access specified in section 3.2.2.1.

2. Let U be the set of all database objects whose objectSid's domain prefix matches the domain prefix of the domain referenced by *DomainHandle*.

3. For each element in *RelativeIds* that matches the RID of a database object's **objectSid** attribute value in the set U, the requester MUST fill in *Names* and *Use* as follows:

    1. Let 'i' be the current element of *RelativeIds*.

    2. **Names.Element[i]** is the **sAMAccoutName** attribute value of the matched object.

    3. **Use.Element[i]** is set as follows:

objectClass	GroupType	Use
User	n/a	SidTypeUser

objectClass	GroupType	Use
Group	GROUP_TYPE_ACCOUNT_GROUP	SidTypeGroup
Group	GROUP_TYPE_UNIVERSAL_GROUP	SidTypeGroup
Group	Any value not matching above criteria for Group	SidTypeAlias

4. For each element in *RelativeIds* that does not match the RID of a database object's **objectSid** attribute value, the requester MUST fill in *Names* and *Use* as follows:

    1. Let 'i' be the current element of *RelativeIds*.

    2. All fields of **Names.Element[i]** MUST be set to 0.

    3. **Use.Element[i]** is SidTypeUnknown.

5. The responder MUST return a database processing error if any occurred; otherwise:

    1. **Names.Count** MUST be set to the input parameter *Count* on successful completion of the method.

    2. **Use.Count** MUST be set to the input parameter *Count* on successful completion of the method.

    3. If the number of matched accounts is equal to the input parameter *Count*, 0 MUST be returned.

    4. If the number of matched accounts is less than the input parameter *Count*, but greater than 0, STATUS_SOME_NOT_MAPPED MUST be returned. Note that this is not an error condition.

    5. If the number of matched accounts is 0, STATUS_NONE_MAPPED MUST be returned.

3.2.5.12 Security Pattern

These methods enable a requester to set the access control on a server, domain, group, alias, or user object.

These methods require a handle obtained from an "open" or "create" method. See sections 3.2.5.1 and 3.2.5.4.

A non-normative description of these methods is helpful to understand the intent of the message processing. The remainder of this section contains such a description.

Two points are significant:

1. The message processing requirements between DC and non-DC configurations are very different.

2. Windows clients use a very small subset of the functionality exposed in these methods.

The DC message processing requirements differ from the non-DC case because the database objects on which the responder operates are also exposed through the LDAP model for read and update, and have a different ACE format than what this protocol exposes. Specifically, in the DC case, the database objects have security descriptors with an object ACE format (specified in [MS-DTYP] section 2.4.4), whereas these methods expect and return security descriptors with a simple ACE format (specified in [MS-DTYP] section 2.4.4). Therefore, the message processing for these methods

must convert between these two models. In general, this is an intractable problem because new access masks and object ACE types can be added that are not expressible through this protocol.

Fortunately, Windows clients use a small subset of the functionality exposed through these methods. Specifically, Windows clients use **SamrQuerySecurityObject** and **SamrSetSecurityObject** only to control whether a user account can change its password (the relevant access mask is USER_CHANGE_PASSWORD, specified in section 2.2.1.7). Accordingly, the responder of these methods is required to support only this narrow request; other requests can be safely ignored.

In the DC case, general security-descriptor manipulation is best achieved through LDAP. [MS-ADTS] section 5 specifies the Active Directory security model in detail.

For the non-DC case, because the security descriptor on the database objects is not exposed through any other protocol, a responder implementation has much greater breadth in implementing the access control specified in the security descriptor presented in a method call to **SamrSetSecurityObject**. Furthermore, because no other protocol can modify the security descriptor on the database objects in a non-DC configuration, it is possible to translate an object ACE format security descriptor to a simple ACE format. Non-DC responders have the requirement to return, via **SamrQuerySecurityObject**, the same access control specification that was specified to a previous call to **SamrSetSecurityObject**, and to enforce all access control permissions specified through **SamrSetSecurityObject**.

See section 1.3 for a description of the "security" pattern of methods.

3.2.5.12.1 SamrSetSecurityObject (Opnum 2)

The **SamrSetSecurityObject** method sets the access control on a server, domain, user, group, or alias object.

```
long SamrSetSecurityObject(
  [in] SAMPR_HANDLE ObjectHandle,
  [in] SECURITY_INFORMATION SecurityInformation,
  [in] PSAMPR_SR_SECURITY_DESCRIPTOR SecurityDescriptor
);
```

ObjectHandle: An RPC context handle, as specified in section 2.2.3.2, representing a server, domain, user, group, or alias object.

SecurityInformation: A bitfield that indicates the fields of *SecurityDescriptor* that are requested to be set.

The SECURITY_INFORMATION type is defined in [MS-DTYP] section 2.4.7. The following bits are valid; all other bits MUST be zero on send and ignored on receipt.

Value	Meaning
OWNER_SECURITY_INFORMATION 0x00000001	Refers to the Owner member of the security descriptor.
GROUP_SECURITY_INFORMATION 0x00000002	Refers to the Group member of the security descriptor.
DACL_SECURITY_INFORMATION 0x00000004	Refers to the DACL of the security descriptor.

Value	Meaning
SACL_SECURITY_INFORMATION 0x00000008	Refers to the SACL of the security descriptor.

SecurityDescriptor: A security descriptor expressing access that is specific to the *ObjectHandle*.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

Message processing for this method is specified in the following two sections.

3.2.5.12.1.1 SamrSetSecurityObject (DC Configuration)

Upon receiving this message, the responder MUST process the data from the message subject to all of the following constraints:

1. The access control specified in *SecurityDescriptor* MUST be a valid security descriptor containing simple ACEs; otherwise the responder MUST return an error status. [MS-DTYP] section **2.4.6** contains the specification for a valid security descriptor. On error, the responder MUST abort processing and return an error.

2. *ObjectHandle* MUST have the required access specified in the following table, based on the set bits in the *SecurityInformation* parameter. The responder MUST ignore set bits in *SecurityInformation* that are not specified in the table. On error, the responder MUST abort processing and return STATUS_ACCESS_DENIED.

Security information bits	Required access
SACL_SECURITY_INFORMATION	ACCESS_SYSTEM_SECURITY
OWNER_SECURITY_INFORMATION	WRITE_OWNER
OWNER_SECURITY_INFORMATION	WRITE_OWNER
DACL_SECURITY_INFORMATION	WRITE_DAC

3. If the database object referenced by *ObjectHandle* is not a user object and the DACL_SECURITY_INFORMATION is not set in *SecurityInformation*, the responder MUST silently ignore the request by aborting processing and returning 0.

4. Otherwise, the responder MUST determine if the DACL of *SecurityDescriptor* of the input message matches one of the following DACLs. The ordering of the ACEs is not relevant. Let Self denote the SID of the user object referenced by *ObjectHandle*.

   - DACL a.

SID	Access mask
WorldSid	USER_EXECUTE | USER_READ
AdministratorsSid	USER_ALL_ACCESS
AccountOperatorsSid	USER_ALL_ACCESS

SID	Access mask
Self	USER_WRITE

- DACL b.

SID	Access mask
WorldSid	(USER_EXECUTE | USER_READ) & ~ USER_CHANGE_PASSWORD
AdministratorsSid	USER_ALL_ACCESS
AccountOperatorsSid	USER_ALL_ACCESS
Self	USER_WRITE

- DACL c.

SID	Access mask
WorldSid	(USER_EXECUTE | USER_READ) & ~ USER_CHANGE_PASSWORD
AdministratorsSid	USER_ALL_ACCESS
AccountOperatorsSid	USER_ALL_ACCESS

- DACL d.

SID	Access mask
WorldSid	USER_EXECUTE | USER_READ
AdministratorsSid	USER_ALL_ACCESS
Self	USER_WRITE

5. If there is no match from constraint 2, the responder MUST silently ignore the request by aborting processing and returning 0.

6. If the matching DACL grants USER_CHANGE_PASSWORD to World, the responder MUST update the **ntSecurityDescriptor** attribute for the target user such that the target user has the ability to change his or her password; otherwise, the responder MUST update the **ntSecurityDescriptor** attribute for the target user such that the target does not have the ability to change his or her password. See<60> for an example of how to do this.

7. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.

3.2.5.12.1.2 SamrSetSecurityObject (Non-DC Configuration)

Upon receiving this message, the responder MUST process the data from the message subject to all the following constraints:

1. The access control specified in *SecurityDescriptor* MUST be a valid security descriptor containing simple ACEs; otherwise the responder MUST return an error status. [MS-DTYP] section **2.4.6** contains the specification for a valid security descriptor.

2. *ObjectHandle* MUST have the required access specified in the following table, based on the set bits in the *SecurityInformation* parameter. The responder MUST ignore set bits in *SecurityInformation* that are not specified in the table. On error, the responder MUST abort processing and return STATUS_ACCESS_DENIED.

Security information bits	Required access
SACL_SECURITY_INFORMATION	ACCESS_SYSTEM_SECURITY
OWNER_SECURITY_INFORMATION	WRITE_OWNER
OWNER_SECURITY_INFORMATION	WRITE_OWNER
DACL_SECURITY_INFORMATION	WRITE_DAC

3. The responder MUST update the **ntSecurityDescriptor** attribute value on the object referenced by *ObjectHandle* such that all of the following constraints are satisfied:

    1. All accesses granted and denied in the input security descriptor (*SecurityDescriptor*) are granted during subsequent method calls across this interface (for all time), per the tables specified in section 3.2.2.1.

    2. If the target object is a domain object, all ACEs containing DOMAIN_CREATE_USER, DOMAIN_CREATE_ALIAS, or DOMAIN_CREATE_GROUP MUST grant or deny (depending on the type of ACE) the trustee of the ACE the ability to create a user, alias, or group as specified in **SamrCreateUser2InDomain (section 3.2.5.4.4)**, **SamrCreateAliasInDomain (section 3.2.5.4.3)**, or **SamrCreateGroupInDomain (section 3.2.5.4.2)**.

    3. If the target object is a user object, all ACEs containing the specified access mask in the following table MUST grant or deny (depending on the type of ACE) the trustee to update associated attributes.

Access mask	Attribute
USER_WRITE_ACCOUNT	sAMAccountName displayName primaryGroupId homeDirectory homeDrive scriptPath profilePath Description userWorkstations logonHours accountExpires userAccountControl userParameters
USER_WRITE_PREFERENCE	comment

Access mask	Attribute
	countryCode codePage
USER_FORCE_PASSWORD_CHANGE	clearTextPassword pwdLastSet dbcsPwd unicodePwd

4. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.

3.2.5.12.2 SamrQuerySecurityObject (Opnum 3)

The **SamrQuerySecurityObject** method queries the access control on a server, domain, user, group, or alias object.

```
long SamrQuerySecurityObject(
  [in] SAMPR_HANDLE ObjectHandle,
  [in] SECURITY_INFORMATION SecurityInformation,
  [out] PSAMPR_SR_SECURITY_DESCRIPTOR* SecurityDescriptor
);
```

ObjectHandle: An RPC context handle, as specified in section 2.2.3.2, representing a server, domain, user, group, or alias object.

SecurityInformation: A bitfield that indicates which fields of *SecurityDescriptor* the requester is requesting to be returned.

The SECURITY_INFORMATION type is defined in [MS-DTYP] section 2.4.7. The following bits are valid; all other bits MUST be zero on send and ignored on receipt.

Value	Meaning
OWNER_SECURITY_INFORMATION 0x00000001	Refers to the Owner member of the security descriptor.
GROUP_SECURITY_INFORMATION 0x00000002	Refers to the Group member of the security descriptor.
DACL_SECURITY_INFORMATION 0x00000004	Refers to the DACL of the security descriptor.
SACL_SECURITY_INFORMATION 0x00000008	Refers to the SACL of the security descriptor.

SecurityDescriptor: A security descriptor expressing accesses that are specific to the *ObjectHandle* and the owner and group of the object. [MS-DTYP] section **2.4.6** contains the specification for a valid security descriptor.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

Message processing for this method is specified in the following two sections.

3.2.5.12.2.1 SamrQuerySecurityObject (DC Configuration)

Let Self denote the **objectSid** attribute value, if any, of the object referenced by *ObjectHandle*.

Upon receiving this message, the responder MUST process the data from the message subject to all of the following constraints:

1. *ObjectHandle* MUST have the required access specified in the following table, based on the bits contained in the *SecurityInformation* parameter. On error, the responder MUST abort processing and return STATUS_ACCESS_DENIED.

Security information bits	Required access
SACL_SECURITY_INFORMATION	ACCESS_SYSTEM_SECURITY
OWNER_SECURITY_INFORMATION	READ_CONTROL
GROUP_SECURITY_INFORMATION	READ_CONTROL
DACL_SECURITY_INFORMATION	READ_CONTROL

2. The responder MUST return, via the *SecurityDescriptor* parameter, a security descriptor that contains fields based on the bits contained in the *SecurityInformation* parameter, and that satisfies all of the following constraints:

    1. The **Owner** and **Group** fields of the security descriptor MUST be the administrator's SID (S-1-5-32-544).

    2. The DACL MUST contain the following specified ACEs:

        If *ObjectHandle* refers to the server object, the DACL MUST contain the following ACEs:

SID	Access mask
WorldSid	SAM_SERVER_EXECUTE | SAM_SERVER_READ
AdministratorSid	SAM_SERVER_ALL_ACCESS

        Else if *ObjectHandle* refers to a domain object, the DACL MUST contain the following ACEs:

SID	Access mask
WorldSid	DOMAIN_EXECUTE | DOMAIN_READ
AdministratorSid	DOMAIN_ALL_ACCESS
AccountOperatorsSid	DOMAIN_EXECUTE | DOMAIN_READ | DOMAIN_CREATE_USER | DOMAIN_CREATE_GROUP | DOMAIN_CREATE_ALIAS

Else if *ObjectHandle* refers a group or alias object that is the Domain Admins group or Administrators alias, or a member of Domain Admins or Administrators, the DACL MUST contain the following ACEs:

SID	Access mask
WorldSid	GROUP_EXECUTE | GROUP_READ
AdministratorSid	GROUP_ALL_ACCESS

Else if *ObjectHandle* refers to any group or alias object that does not satisfy the previous condition, the DACL MUST contain the following ACEs:

SID	Access mask
WorldSid	GROUP_EXECUTE | GROUP_READ
AdministratorSid	GROUP_ALL_ACCESS
AccountOperatorsSid	GROUP_ALL_ACCESS

Else if *ObjectHandle* refers to a user object that is a member of Domain Admins or Administrators, the DACL MUST contain the following ACEs:

SID	Access mask
WorldSid	USER_EXECUTE | USER_READ
AdministratorSid	USER_ALL_ACCESS
The SID of the user referenced by *ObjectHandle*	USER_WRITE

Else if *ObjectHandle* refers to user object whose **ntSecurityDescriptor** does not grant Self or World the ab721a53-1e2f-11d0-9819-00aa0040529b Control Access Right, the DACL MUST contain the following ACEs:

SID	Access mask
WorldSid	USER_EXECUTE | USER_READ | ~USER_CHANGE_PASSWORD
AdministratorSid	USER_ALL_ACCESS
AccountOperatorsSid	USER_ALL_ACCESS
The SID of the user referenced by *ObjectHandle*	USER_WRITE

Otherwise, the DACL MUST contain the following ACEs:

SID	Access mask
WorldSid	USER_EXECUTE | USER_READ
AdministratorSid	USER_ALL_ACCESS
AccountOperatorsSid	USER_ALL_ACCESS
The SID of the user referenced by *ObjectHandle*	USER_WRITE

3. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.

3.2.5.12.2.2 SamrQuerySecurityObject (Non-DC Configuration)

Upon receiving this message, the responder MUST process the data from the message subject to the following constraints:

1. *ObjectHandle* MUST have the required access specified in the following table, based on the bits contained in the *SecurityInformation* parameter. On error, the responder MUST abort processing and return STATUS_ACCESS_DENIED.

Security information bits	Required access
SACL_SECURITY_INFORMATION	ACCESS_SYSTEM_SECURITY
OWNER_SECURITY_INFORMATION	READ_CONTROL
GROUP_SECURITY_INFORMATION	READ_CONTROL
DACL_SECURITY_INFORMATION	READ_CONTROL

2. The responder MUST return the *SecurityDescriptor* parameter with a security descriptor that contains fields based on the bits contained in the *SecurityInformation* parameter. The security descriptor expresses the owner and group of the referenced object and an access control (SACL and DACL) that has been specified either by default settings, or by previous calls to **SamrSetSecurityObject**. The security descriptor MUST be in terms of simple ACEs and ACCESS_MASK values as specified in the following table, based on the object type that *ObjectHandle* references.

Object type	ACCESS_MASK section
Server	2.2.1.1
Domain	2.2.1.4
Group	2.2.1.5
Alias	2.2.1.6
User	2.2.1.7

3. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.

3.2.5.13 Miscellaneous

See section 1.3 for a description of these methods.

3.2.5.13.1 SamrCloseHandle (Opnum 1)

The **SamrCloseHandle** method closes (that is, releases responder-side resources used by) any context handle obtained from this RPC interface.

```
long SamrCloseHandle(
  [in, out] SAMPR_HANDLE* SamHandle
);
```

SamHandle: An RPC context handle, as specified in section 2.2.3.2, representing any context handle returned from this interface.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

Upon receiving this message, the responder MUST process the data from the message subject to the following constraints:

1. If *SamHandle* is 0, the responder MUST return an error.
2. Otherwise, the responder MUST release any internal resources associated with the handle, and then MUST return 0 for the value of SamHandle and a return code of STATUS_SUCCESS.

3.2.5.13.2 SamrShutdownSamServer (Opnum 4)

The **SamrShutdownSamServer** method shuts down a SAM server's RPC interface. This method requires SAM_SERVER_SHUTDOWN access to the server.

This method is reserved and is not intended for network traffic.<61>

```
long SamrShutdownSamServer(
  [in] SAMPR_HANDLE ServerHandle
);
```

ServerHandle: SAM server handle, which is obtained from a call to **SamrConnect5**.

<62>

3.2.5.13.3 SamrSetMemberAttributesOfGroup (Opnum 26)

The **SamrSetMemberAttributesOfGroup** method sets the attributes of a member relationship.

```
long SamrSetMemberAttributesOfGroup(
  [in] SAMPR_HANDLE GroupHandle,
  [in] unsigned long MemberId,
  [in] unsigned long Attributes
);
```

GroupHandle: An RPC context handle, as specified in section 2.2.3.2, representing a group object.

MemberId: A RID that represents a member of a group (which is a user or machine account).

Attributes: The characteristics of the membership relationship. For legal values, see section 2.2.1.10.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

On receiving this message, the responder MUST process the data from the message subject to the following constraints:

1. *GroupHandle* MUST have the required access specified in section 3.2.2.1.
2. In a non-DC configuration, the *MemberId* parameter MUST be a member of the group referenced by *GroupHandle*; otherwise, processing MUST be aborted and an error returned.
3. For a message processing specification of the *Attributes* field, see section 3.2.5.14.7.
4. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.

3.2.5.13.4 SamrGetUserDomainPasswordInformation (Opnum 44)

The **SamrGetUserDomainPasswordInformation** method obtains select password policy information (without requiring a domain handle).

```
long SamrGetUserDomainPasswordInformation(
  [in] SAMPR_HANDLE UserHandle,
  [out] PUSER_DOMAIN_PASSWORD_INFORMATION PasswordInformation
);
```

UserHandle: An RPC context handle, as specified in section 2.2.3.2, representing a user object.

PasswordInformation: Password policy information from the user's domain.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

On receiving this message, the responder MUST process the data from the message subject to the following constraints:

1. The security identity of the requester MUST have DOMAIN_READ_PASSWORD_PARAMETERS access to the account domain object; if not, the responder MUST abort processing and return STATUS_ACCESS_DENIED.
2. If the RelativeId of the **objectSid** attribute of the user object referenced by *UserHandle* is DOMAIN_RID_USER_KRBTGT, or if the **userAccountControl** attribute does not contain UF_INTERDOMAIN_TRUST_ACCOUNT, UF_WORKSTATION_TRUST_ACCOUNT, or UF_SERVER_TRUST_ACCOUNT, *PasswordInformation* MUST be set to all zeros, and the responder MUST end processing and return STATUS_SUCCESS.

3. The output parameter **PasswordInformation.MinPasswordLength** MUST be set to the Effective-MinimumPasswordLength attribute value (see section 3.2.1.5).

4. The output parameter **PasswordInformation.PasswordProperties** MUST be set to the **pwdProperties** attribute value on the account domain object. In addition:

    1. If the Effective-PasswordComplexityEnabled value (see section 3.2.1.5) is set, **PasswordInformation.PasswordProperties** MUST contain DOMAIN_PASSWORD_COMPLEX.

    2. If the Effective-PasswordReversibleEncryptionEnabled value (see section 3.2.1.5) is set, **PasswordInformation.PasswordProperties** MUST contain DOMAIN_PASSWORD_STORE_CLEARTEXT.

5. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.

3.2.5.13.5 SamrGetDomainPasswordInformation (Opnum 56)

The **SamrGetDomainPasswordInformation** method obtains select password policy information (without authenticating to the responder).

```
long SamrGetDomainPasswordInformation(
  [in] handle_t BindingHandle,
  [in, unique] PRPC_UNICODE_STRING Unused,
  [out] PUSER_DOMAIN_PASSWORD_INFORMATION PasswordInformation
);
```

BindingHandle: An RPC binding handle parameter, as specified in [C706-Ch2Intro].

Unused: A string value that is unused by the protocol. It is ignored by the responder. The requester MAY<63> set any value.

PasswordInformation: Password policy information from the account domain.

There is no security enforced for this method.

Upon receiving this message, the responder MUST process the data from the message subject to the following constraints:

1. The output parameter **PasswordInformation.MinPasswordLength** MUST be set to the **minPwdLength** attribute value on the account domain object.

2. The output parameter **PasswordInformation.PasswordProperties** MUST be set to the **pwdProperties** attribute value on the account domain object.

3. The method MUST return 0.

3.2.5.13.6 SamrRidToSid (Opnum 65)

The **SamrRidToSid** method obtains the SID of an account, given a RID.

```
long SamrRidToSid(
  [in] SAMPR_HANDLE ObjectHandle,
  [in] unsigned long Rid,
```

```
  [out] PRPC_SID* Sid
);
```

ObjectHandle: An RPC context handle, as specified in section 2.2.3.2. The message processing shown below contains details on which types of *ObjectHandle* are accepted by the responder.

Rid: A RID of an account.

Sid: The SID of the account referenced by *Rid*.

This protocol asks the RPC runtime, via the **strict_context_handle** attribute, to reject the use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

Upon receiving this message, the responder MUST process the data from the message subject to the following constraints:

1. The *ObjectHandle* MUST reference a domain, user, group, or alias object.
2. The output parameter *Sid* MUST be set to the **objectSid** attribute value of the object referenced by the *Rid* parameter.
3. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.

3.2.5.13.7 SamrSetDSRMPassword (Opnum 66)

The **SamrSetDSRMPassword** method sets a local recovery password.

```
long SamrSetDSRMPassword(
  [in] handle_t BindingHandle,
  [in, unique] PRPC_UNICODE_STRING Unused,
  [in] unsigned long UserId,
  [in, unique] PENCRYPTED_NT_OWF_PASSWORD EncryptedNtOwfPassword
);
```

BindingHandle: An RPC binding handle parameter, as specified in [C706-Ch2Intro].

Unused: A string value. This value is not used in the protocol, and is ignored by the responder.

UserId: A RID of a user account. See the message processing below for details on restrictions on this value.

EncryptedNtOwfPassword: The NT hash of the new password (as presented by the requester) encrypted per the specification of **ENCRYPTED_NT_OWF_PASSWORD**, where the key is the *UserId*.

Upon receiving this message, the responder MUST process the data from the message subject to the following constraints:

1. The requester MUST be a member of the **Administrators** alias.
2. On a non-DC configuration, the responder MUST return an error code.
3. The responder MAY<64> enforce parameter checks on the *UserId* parameter.

4. The responder MAY<65> decrypt *EncryptedNtOwfPassword* using *UserId* as a key and use the result to store the password of a local recovery account.

5. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.

3.2.5.13.8 SamrValidatePassword (Opnum 67)

The **SamrValidatePassword** method validates an application password against the locally stored policy.

```
long SamrValidatePassword(
  [in] handle_t Handle,
  [in] PASSWORD_POLICY_VALIDATION_TYPE ValidationType,
  [in, switch_is(ValidationType)]
    PSAM_VALIDATE_INPUT_ARG InputArg,
  [out, switch_is(ValidationType)]
    PSAM_VALIDATE_OUTPUT_ARG* OutputArg
);
```

Handle: An RPC binding handle parameter, as specified in [C706-Ch2Intro].

ValidationType: The password policy validation requested.

InputArg: The password-related material to validate.

OutputArg: The result of the validation.

On receiving this message, the responder MUST process the data from the message subject to the following constraints:

1. The requester MUST have SAM_SERVER_LOOKUP_DOMAIN access on the server object (the same server object that is referenced by **SamrConnect5 (section 3.2.5.1.1)** and DOMAIN_READ_PASSWORD_PARAMETERS on the account domain object.

2. Let the following symbolic names correspond to the values specified in the table:

Symbolic name	Attribute value on the account domain object
DomainPasswordHistoryLength	pwdHistoryLength
DomainLockoutDuration	lockoutDuration
DomainLockoutObservationWindow	lockoutObservationWindow
DomainLockoutThreshold	lockoutThreshold
DomainMinimumPasswordLength	minPasswordLength
DomainMaximumPasswordAge	minPwdAge
DomainMinimumPasswordAge	maxPwdAge

3. Any field of OutputArg that is modified MUST cause the associated bit in PresentFields (in the **SAM_VALIDATE_PERSISTED_FIELDS** structure) to be set per the following table:

Bit	Corresponding field
SAM_VALIDATE_PASSWORD_LAST_SET	PasswordLastSet
SAM_VALIDATE_BAD_PASSWORD_TIME	BadPasswordTime
SAM_VALIDATE_LOCKOUT_TIME	LockoutTime
SAM_VALIDATE_BAD_PASSWORD_COUNT	BadPasswordCount
SAM_VALIDATE_PASSWORD_HISTORY	PasswordHistoryLength

4. Additional constraints in the following sections MUST be satisfied based on the *ValidationType* input parameter, per the following table. If the *ValidationType* input parameter does not match a row in the table, an error MUST be returned.

ValidationType	Section
SamValidateAuthentication	3.2.5.13.8.1
SamValidatePasswordChange	3.2.5.13.8.2
SamValidatePasswordReset	3.2.5.13.8.3

3.2.5.13.8.1 SamValidateAuthentication

The following table lists the constraints that MUST be satisfied (in the order presented) in order to return the associated output parameters to the requester. All fields of ValidateAuthenticationOutput MUST be set to 0 before any constraints are met.

Condition (fields based on ValidateAuthenticationInput)	ValidateAuthenticationOutput changes
LockoutTime plus DomainLockoutDuration is greater than the current time.	1. ValidationStatus MUST be set to SamValidateAccountLockedOut. 2. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.
LockoutTime plus DomainLockoutDuration is less than or equal to the current time.	LockoutTime MUST be set to 0 (and continue processing).
PasswordLastSet is zero.	1. ValidationStatus MUST be set to SamValidateAccountLockedOut. 2. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.
PasswordLastSet plus DomainMaximumPasswordAge is greater than the current time.	1. ValidationStatus MUST be set to

Condition (fields based on ValidateAuthenticationInput)	ValidateAuthenticationOutput changes
	SamValidatePasswordExpired. 2. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.
PasswordMatch is zero, and BadPasswordTime plus DomainLockoutObservationWindow is greater than or equal to the current time.	1. ValidationStatus MUST be set to SamValidatePasswordIncorrect. 2. BadPasswordCount MUST be set to 1. 3. BadPasswordTime MUST be set to the current time. 4. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.
PasswordMatch is zero, and BadPasswordTime plus DomainLockoutObservationWindow is less than the current time.	1. ValidationStatus MUST be set to SamValidatePasswordIncorrect. 2. BadPasswordCount MUST be set to ValidateAuthenticationInput.BadPasswordCount plus 1. 3. BadPasswordTime MUST be set to the current time. 4. If DomainLockoutThreshold is greater than 0 and BadPasswordCount is greater than or equal to DomainLockoutThreshold, LockoutTime MUST be set to the current time. 5. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.
PasswordMatched is nonzero.	1. ValidationStatus MUST be set to SamValidateSuccess. 2. If BadPasswordCount is nonzero, BadPasswordCount MUST be set to 0. 3. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.

3.2.5.13.8.2 SamValidatePasswordChange

The following table lists the constraints that MUST be satisfied (in the order presented) in order to return the associated output parameters to the requester. All fields of ValidatePasswordChangeOutput MUST be set to 0 before any constraints are met.

Condition (fields based on ValidatePasswordChangeInput)	ValidatePasswordChangeOutput changes
LockoutTime plus DomainLockoutDuration is greater than the current time.	1. ValidationStatus MUST be set to SamValidateAccountLockedOut. 2. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.
LockoutTime plus DomainLockoutDuration is less than or equal to the current time.	LockoutTime MUST be set to 0.
PasswordLastSet plus DomainMinimumPasswordAge is greater than the current time.	1. ValidationStatus MUST be set to SamValidatePasswordTooRecent. 2. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.
PasswordMatch is zero, and BadPasswordTime plus DomainLockoutObservationWindow is greater than or equal to the current time.	1. ValidationStatus MUST be set to SamValidatePasswordIncorrect. 2. BadPasswordCount MUST be set to 1. 3. BadPasswordTime MUST be set to the current time. 4. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.
PasswordMatch is zero, and BadPasswordTime plus DomainLockoutObservationWindow is less than the current time.	1. ValidationStatus MUST be set to SamValidatePasswordIncorrect. 2. BadPasswordCount MUST be set to ValidateAuthenticationInput.BadPasswordCount plus 1. 3. BadPasswordTime MUST be set to the current time. 4. If DomainLockoutThreshold is greater than 0 and BadPasswordCount is greater than or equal to DomainLockoutThreshold, LockoutTime MUST be set to the current time. 5. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.
PasswordMatch is nonzero, and HashedPassword is equal to at least one of the first DomainPasswordHistoryLength elements of PasswordHistory (without exceeding the number of elements in PasswordHistory)	1. ValidateStatus MUST be set to SamValidatePasswordIsInHistory. 2. The responder MUST return a database processing error if any exist; otherwise, return STATUS_SUCCESS.

Condition (fields based on ValidatePasswordChangeInput)	ValidatePasswordChangeOutput changes
where the Length field of HashedPassword is equal to the Length field of the PasswordHistory element.	
PasswordMatch is nonzero.	1. The constraints in section 3.2.1.8.5 MUST be satisfied where sAMAccountName is ValidatePasswordChangeInput.UserAccountName, userAccountControl is UF_NORMAL_ACCOUNT, and objectSid and displayName are empty; on error ValidationStatus MUST be set as follows: 1. If the minimum password length constraint fails, ValidationStatus MUST be SamValidatePasswordTooShort. 2. If maximum password length constraint fails, ValidationStatus MUST be SamValidatePasswordTooLong. 3. If there is an implementation-specific password filter and that filter fails, ValidationStatus MUST be SamValidatePasswordFilterError. 4. If any other constraint fails, ValidationStatus MUST be SamValidatePasswordNotComplexEnough. 2. If any constraint from item 1 failed, the responder MUST return STATUS_SUCCESS. 3. Otherwise (no constraint from item 1 failed), PasswordHistory MUST be updated such that ValidatePasswordChangeInput.HashedPassword is the first element in PasswordHistory, and ValidatePasswordChangeInput.InputPersistedFields.PasswordHistory elements are used, starting from the left, to fill the remaining elements of PasswordHistory such that PasswordHistory contains as many elements as possible up to DomainPasswordHistoryLength elements. 4. PasswordHistoryLength MUST be updated to be DomainPasswordHistoryLength. 5. PasswordLastSet MUST be set to the current time. 6. BadPasswordCount is set to 0. 7. ValidationStatus MUST be set to SamValidateSuccess. 8. The responder MUST return any processing errors; otherwise, it MUST return 0.

3.2.5.13.8.3 SamValidatePasswordReset

The following table lists the constraints that MUST be satisfied (in the order presented) in order to return the associated output parameters to the requester. All fields of ValidatePasswordResetOutput MUST be set to 0 before any constraints are met.

Condition (fields based on ValidatePasswordResetInput)	ValidatePasswordResetOutput changes
Always	1. The constraints in section 3.2.1.8.5 MUST be satisfied where sAMAccountName is ValidatePasswordChangeInput.UserAccountName, userAccountControl is UF_NORMAL_ACCOUNT, and objectSid and displayName are empty; on error ValidationStatus MUST be set as follows: 1. If the minimum password length constraint fails, ValidationStatus MUST be SamValidatePasswordTooShort. 2. If the maximum password length constraint fails, ValidationStatus MUST be SamValidatePasswordTooLong. 3. If there is an implementation-specific password filter and that filter fails, ValidationStatus MUST be SamValidatePasswordFilterError. 4. If any other constraint fails, ValidationStatus MUST be SamValidatePasswordNotComplexEnough. 2. If any constraint from item 1 failed, the responder MUST return STATUS_SUCCESS.
PasswordMustChangeAtNextLogon is nonzero.	PasswordLastSet MUST be set to zero.
PasswordMustChangeAtNextLogon is zero.	PasswordLastSet MUST be set to the current time.
ClearLockout is nonzero.	1. LockoutTime MUST be set to 0. 2. If ValidatePasswordResetInput.InputPersistedFields.BadPasswordCount is non-zero, BadPasswordCount MUST be set to 0.
Always	1. PasswordHistory MUST be updated such that ValidatePasswordResetInput.HashedPassword is the first element in PasswordHistory and ValidatePasswordResetInput.InputPersistedFields.PasswordHistory elements are used, starting from the left, to fill the remaining elements of PasswordHistory such that PasswordHistory contains as many elements as possible up to DomainPasswordHistoryLength elements.

Condition (fields based on ValidatePasswordResetInput)	ValidatePasswordResetOutput changes
	2. PasswordHistoryLength MUST be updated to be DomainPasswordHistoryLength. 3. PasswordLastSet MUST be set to the current time. 4. BadPasswordCount is set to 0. 5. ValidationStatus MUST be set to SamValidateSuccess. 6. The responder MUST return any processing errors; otherwise, it MUST return 0.

3.2.5.14 Supplemental Message Processing

3.2.5.14.1 distinguishedName Generation

This section contains constraints pertaining to the generation of a distinguishedName attribute value for objects created through this protocol. This section is referenced by the "create" pattern of methods, section 3.2.5.4. The constraints refer to an AccountType parameter from the referring section; if the object being created has the objectClass of a group, there is no AccountType parameter in the message. In this case, use an Account Type value of USER_NORMAL_ACCOUNT.

1. If the wellKnownObjects attribute on the account domain object exists and contains a value that matches the GUID associated with Account Type, where Account Type is the AccountType parameter from the message referencing this section, the distinguishedName MUST be suffixed with the associated value from the wellKnownObject attribute. Information about the syntax of the wellKnownObject attribute is specified in [MS-ADTS] section 7.1.1.4.

AccountType	wellKnownObject GUID
USER_NORMAL_ACCOUNT	a9d1ca15768811d1aded00c04fd8d5cd
USER_WORKSTATION_TRUST_ACCOUNT	aa312825768811d1aded00c04fd8d5cd
USER_SERVER_TRUST_ACCOUNT	a361b2ffffd211d1aa4b00c04fd7d83a

2. If the wellKnownObjects attribute does not exist or there is no match per constraint 1, the distinguishedName MUST be suffixed with the associated value per the table below.

AccountType	distinguishedName suffix
USER_NORMAL_ACCOUNT	CN=Users,<DN of account domain object>
USER_WORKSTATION_TRUST_ACCOUNT	CN=Computers,<DN of account domain object>
USER_SERVER_TRUST_ACCOUNT	CN=Domain Controllers,<DN of account domain object>

3. The responder MUST prefix the RDN directly in front of the suffix determined from steps 1 and 2. Implementations SHOULD<66> use the sAMAccountName as the value for the RDN, with the component type of "CN", if this choice matches the constraints of the distinguishedName attribute.

3.2.5.14.2 userAccountControl Mapping Table

Protocol UserAccountControl	Database userAccountControl
USER_ACCOUNT_DISABLED	UF_ACCOUNTDISABLE
USER_HOME_DIRECTORY_REQUIRED	UF_HOMEDIR_REQUIRED
USER_PASSWORD_NOT_REQUIRED	UF_PASSWD_NOTREQD
USER_ENCRYPTED_TEXT_PASSWORD_ALLOWED	UF_ENCRYPTED_TEXT_PASSWORD_ALLOWED
USER_NORMAL_ACCOUNT	UF_NORMAL_ACCOUNT
USER_INTERDOMAIN_TRUST_ACCOUNT	UF_INTERDOMAIN_TRUST_ACCOUNT
USER_WORKSTATION_TRUST_ACCOUNT	UF_WORKSTATION_TRUST_ACCOUNT
USER_SERVER_TRUST_ACCOUNT	UF_SERVER_TRUST_ACCOUNT
USER_DONT_EXPIRE_PASSWORD	UF_DONT_EXPIRE_PASSWD
USER_MNS_LOGON_ACCOUNT	UF_MNS_LOGON_ACCOUNT
USER_SMARTCARD_REQUIRED	UF_SMARTCARD_REQUIRED
USER_TRUSTED_FOR_DELEGATION	UF_TRUSTED_FOR_DELEGATION
USER_NOT_DELEGATED	UF_NOT_DELEGATED
USER_USE_DES_KEY_ONLY	UF_USE_DES_KEY_ONLY
USER_DONT_REQUIRE_PREAUTH	UF_DONT_REQUIRE_PREAUTH
USER_TRUSTED_TO_AUTHENTICATE_FOR_DELEGATION	UF_TRUSTED_TO_AUTHENTICATE_FOR_DELEGATION
USER_NO_AUTH_DATA_REQUIRED	UF_NO_AUTH_DATA_REQUIRED
USER_ACCOUNT_AUTO_LOCKED	UF_LOCKOUT
USER_PASSWORD_EXPIRED	UF_PASSWORD_EXPIRED

3.2.5.14.3 PasswordCanChange Generation

The PasswordCanChange value is computed as follows:

1. If either the dbcsPwd attribute or the unicodePwd attribute does not have a value, or if either of these is equal to the respective hash of a zero-length string, PasswordCanChange MUST be 0.

2. Otherwise, the PasswordCanChange value MUST be the pwdLastSet attribute value on the user object plus the Effective-MinimumPasswordAge attribute value (see section 3.2.1.5).

3.2.5.14.4 PasswordMustChange Generation

The PasswordMustChange value is computed as follows:

1. If the userAccountControl attribute value on the target user object contains any of the following bits: UF_DONT_EXPIRE_PASSWD, UF_SMARTCARD_REQUIRED,

UF_INTERDOMAIN_TRUST_ACCOUNT, UF_WORKSTATION_TRUST_ACCOUNT, or UF_SERVER_TRUST_ACCOUNT, the PasswordMustChange value MUST be 0x7FFFFFFF FFFFFFFF.

2. Else, if the pwdLastSet attribute value on the user object is 0, the PasswordMustChange value MUST be 0.

3. Else, if the Effective-MaximumPasswordAge attribute value (see section 3.2.1.5) is 0, the PasswordMustChange value MUST be 0x7FFFFFFF FFFFFFFF.

4. Otherwise, the PasswordMustChange value MUST be the pwdLastSet attribute value on the user object plus the Effective-MaximumPasswordAge attribute value (see section 3.2.1.5).

3.2.5.14.5 Account Lockout Enforcement and Reset

1. Let U be the user account that is the subject of a change password request.

2. If U's lockoutTime attribute value plus the attribute value of Effective-LockoutDuration (see section 3.2.1.5) is less than the current time, the responder MUST abort the request and return STATUS_ACCOUNT_LOCKED_OUT.

3. Otherwise, U's lockoutTime MUST be updated to the value 0.

3.2.5.14.6 Account Lockout State Maintenance

1. Let U be the user account that is the subject of a change password request.

2. If the Effective-LockoutThreshold attribute value (see section 3.2.1.5) is greater than zero and U's lockoutTime attribute value is zero or non-existent, all of the following constraints apply:

    1. If the time period between U's badPwdTime attribute value and the current time is greater than the attribute value of the Effective-LockoutObservationWindow (see section 3.2.1.5), the responder MUST set U's badPwdCount attribute value to one. Otherwise, the responder MUST increment U's badPwdCount attribute value by one.

    2. The responder MUST update U's badPwdTime attribute value to the current time (with **FILETIME** syntax).

    3. If the Effective-LockoutThreshold attribute value (see section 3.2.1.5) is greater than zero, and BadPasswordCount is greater than or equal to lockoutThreshold, the responder MUST update U's lockoutTime attribute to the current time (with **FILETIME** syntax).

3.2.5.14.7 Attributes Field Handling

This protocol associates a field called "Attributes" with a group object and a user membership for a group. This field is a bitfield that uses values from the space specified in section 2.2.1.10.

For a group object, this field can be set via **SamrSetInformationGroup** and queried via **SamrQueryInformationGroup** and the **SamrQueryDisplayInformation** family of methods.

For a user membership, this field can be set via **SamrAddMemberToGroup** and **SamrSetMemberAttributesOfGroup** and queried via **SamrGetGroupsForUser** and **SamrGetMembersInGroup**.

This section specifies the message processing for this field for the aforementioned methods.

On a DC configuration:

- On query, the returned value MUST be a logical union of the following bits: SE_GROUP_MANDATORY, SE_GROUP_ENABLED_BY_DEFAULT, and SE_GROUP_ENABLED.
- On set, this field is ignored. The requester SHOULD<67> set the value to the logical union of the following bits: SE_GROUP_MANDATORY, SE_GROUP_ENABLED_BY_DEFAULT, and SE_GROUP_ENABLED.

On a non-DC configuration:

- Any value set via **SamrSetInformationGroup** MUST be returned via a subsequent call to **SamrQueryInformationGroup** or the **SamrQueryDisplayInformation** family of methods at any time in the future (not just within the current session). If no such **SamrSetInformationGroup** call has been made, a default value of zero MUST be returned.
- Any value set via **SamrAddMemberToGroup** or **SamrSetMemberAttributesOfGroup** MUST be returned via a subsequent call to **SamrGetGroupsForUser** or **SamrGetMembersInGroup** at any time in the future (not just within the current session). If no such call to **SamrAddMemberToGroup** or **SamrSetMemberAttributesOfGroup** has been made, a default value of zero MUST be returned.

3.2.5.14.8 Domain Field to Attribute Name Mapping

This table specifies the field-to-database-attribute mapping, where the field is a field in a domain-related structure such as **SAMPR_DOMAIN_GENERAL_INFORMATION (section 2.2.4.10)**, and the database attribute is an attribute defined on a domain object. These attributes are from the data model specified in section 3.2.1.

Field name	Database attribute
CreationTime	creationTime
DomainModifiedCount	modifiedCount
DomainName	Name
ForceLogoff	forceLogoff
LockoutDuration	lockoutDuration
LockoutObservationWindow	lockoutObservationWindow
LockoutThreshold	lockoutThreshold
ModifiedCountAtLastPromotion	modifiedCountAtLastProm
MaxPasswordAge	maxPwdAge
MinPasswordAge	minPwdAge
MinPasswordLength	minPwdLength
PasswordHistoryLength	pwdHistoryLength
PasswordProperties	pwdProperties
OemInformation	oEMInformation
ReplicaSourceNodeName	domainReplica

3.2.5.14.9 Group Field to Attribute Name Mapping

This table specifies the field-to-database-attribute mapping, where the field is a field in a group-related structure such as **SAMPR_GROUP_GENERAL_INFORMATION (section 2.2.5.3)**, and the database attribute is an attribute defined on a group object. These attributes are from the data model specified in section 3.2.1.

Field name	Database attribute or value
AdminComment	Description
Attributes	See section 3.2.5.14.7 for a message processing specification.
Member	The number of values in the member attribute.
Name	sAMAccountName

3.2.5.14.10 Alias Field to Attribute Name Mapping

This table specifies the field-to-database-attribute mapping, where the field is a field in a group-related structure such as **SAMPR_ALIAS_GENERAL_INFORMATION (section 2.2.6.2)**, and the database attribute is an attribute defined on an alias object. These attributes are from the data model specified in section 3.2.1.

Field name	Database attribute or value
AdminComment	Description
Member	The number of values in the member attribute.
Name	sAMAccountName

3.2.5.14.11 User Field to Attribute Name Mapping

This table specifies the field-to-database-attribute mapping, where the field is a field in a user-related structure such as **SAMPR_USER_ALL_INFORMATION (section 2.2.7.6)**, and the database attribute is an attribute defined on a user object. These attributes are from the data model specified in section 3.2.1.

Field name	Database attribute
LastLogon	lastLogon
LastLogoff	lastLogoff
PasswordLastSet	pwdLastSet
AccountExpires	accountExpires
PasswordCanChange	See section 3.2.5.14.3 for message processing.
PasswordMustChange	See section 3.2.5.14.4 for message processing.
UserName	sAMAccountName
FullName	displayName

Field name	Database attribute
HomeDirectory	homeDirectory
HomeDirectoryDrive	homeDrive
ScriptPath	scriptPath
ProfilePath	profilePath
AdminComment	description
WorkStations	userWorkstations
UserComment	comment
Parameters	userParameters
UserId	RID of objectSid
PrimaryGroupId	primaryGroupId
UserAccountControl	userAccountControl*
LogonHours	logonHours
BadPasswordCount	badPwdCount
LogonCount	logonCount
CountryCode	countryCode
CodePage	codePage

* On read of UserAccountControl, the database attribute value MUST be:

1. Augmented with the UF_LOCKOUT bit if the lockoutTime attribute value on the target object is nonzero, and its value plus the Effective-LockoutDuration attribute value (section 3.2.1.5) is less than the current time.
2. Augmented with the UF_PASSWORD_EXPIRED if PasswordMustChange is less than the current time.
3. Translated per the table in section 3.2.5.14.2.

3.2.6 Timer Events

This protocol introduces no timer events.

3.2.7 Other Local Events

This protocol introduces no other local events.

4 Protocol Examples

4.1 Creating a User Account

The following sequence of methods and parameters creates a user account given a network address of "msdc-1", a domain name of "ms", and a user name of "testuser".

1. Send **SamrConnect**

Parameter field	Parameter value
ServerName	msdc-1
DesiredAccess	0x31

2. Receive **SamrConnect**

Parameter field	Parameter value
Status	0
ServerHandle	[implementation-specific value] serverHandle

3. Send **SamrLookupDomainInSamServer**

Parameter field	Parameter value
ServerHandle	serverHandle
Name.Length	4
Name.MaximumLength	4
Name.Buffer	ms

4. Receive **SamrLookupDomainInSamServer**

Parameter field	Parameter value
Status	0
DomainId	[implementation-specific SID]. For example: S-1-5-21-3448151421-356457007-600757626

5. Send **SamrOpenDomain**

Parameter field	Parameter value
ServerHandle	serverHandle
DesiredAccess	0x00000010
DomainId	S-1-5-21-3448151421-356457007-600757626

6. Receive **SamrOpenDomain**

Parameter field	Parameter value
Status	0
DomainHandle	[implementation-specific value] domainHandle

7. Send **SamrCreateUser2InDomain**

Parameter field	Parameter value
DomainHandle	domainHandle
Name.Length	16
Name.MaximumLength	16
Name.Buffer	testuser
AccountType	0x00000080
DesiredAccess	0x02000000

8. Receive **SamrCreateUser2InDomain**

Parameter field	Parameter value
Status	0
UserHandle	[implementation-specific value] userHandle
GrantedAccess	0xf07ff
RelativeId	2810

9. Send **SamrCloseHandle**

Parameter field	Parameter value
Handle	userHandle

10. Receive **SamrCloseHandle**

Parameter field	Parameter value
Status	0
Handle	0

11. Send **SamrCloseHandle**

Parameter field	Parameter value
Handle	domainHandle

12. Receive **SamrCloseHandle**

Parameter field	Parameter value
Status	0
Handle	0

13. Send **SamrCloseHandle**

Parameter field	Parameter value
Handle	serverHandle

14. Receive **SamrCloseHandle**

Parameter field	Parameter value
Status	0
Handle	0

4.2 Enabling a User Account

The following sequence of methods and parameters enables the user account created in the previous example. This is performed on the machine with the network address of "msdc-1", a domain name of "ms", and a user name of "testuser" with Relative ID = 2810.

1. Send **SamrConnect**

Parameter field	Parameter value
ServerName	msdc-1
DesiredAccess	0x31

2. Receive **SamrConnect**

Parameter field	Parameter value
Status	0
ServerHandle	[implementation-specific value] serverHandle

3. Send **SamrLookupDomainInSamServer**

Parameter field	Parameter value
ServerHandle	serverHandle
Name.Length	4
Name.MaximumLength	4

Parameter field	Parameter value
Name.Buffer	ms

4. Receive **SamrLookupDomainInSamServer**

Parameter field	Parameter value
Status	0
DomainId	[implementation-specific SID]. For example: S-1-5-21-3448151421-356457007-600757626

5. Send **SamrOpenDomain**

Parameter field	Parameter value
ServerHandle	serverHandle
DesiredAccess	0x00000010
DomainId	S-1-5-21-3448151421-356457007-600757626

6. Receive **SamrOpenDomain**

Parameter field	Parameter value
Status	0
DomainHandle	[implementation-specific value] domainHandle

7. Send **SamrOpenUser**

Parameter field	Parameter value
DomainHandle	domainHandle
DesiredAccess	0x02000000
UserId	2810

8. Receive **SamrOpenUser**

Parameter field	Parameter value
Status	0
UserHandle	[implementation-specific value] userHandle

9. Send **SamrSetInformationUser2**

Parameter field	Parameter value
DomainHandle	domainHandle
UserInformationClass	16
Buffer	Control = { 0x00000010 }

10. Receive **SamrSetInformationUser2**

Parameter field	Parameter value
Status	0

11. Send **SamrCloseHandle**

Parameter field	Parameter value
Handle	userHandle

12. Receive **SamrCloseHandle**

Parameter field	Parameter value
Status	0
Handle	0

13. Send **SamrCloseHandle**

Parameter field	Parameter value
Handle	domainHandle

14. Receive **SamrCloseHandle**

Parameter field	Parameter value
Status	0
Handle	0

15. Send **SamrCloseHandle**

Parameter field	Parameter value
Handle	serverHandle

16. Receive **SamrCloseHandle**

Parameter field	Parameter value
Status	0

Parameter field	Parameter value
Handle	0

4.3 Encrypting an NT or LM Hash

The following example shows actual values for the cleartext passwords and password hashes as well as the key derivations necessary to apply [FIPS81].

- Old password is "OLDPASSWORD"

  LM hash of "OLDPASSWORD":

  ```
  c9 b8 1d 93 9d 6f d8 0c d4 08 e6 b1 05 74 18 64
  ```

  NT hash of "OLDPASSWORD":

  ```
  66 77 b2 c3 94 31 13 55 b5 4f 25 ee c5 bf ac f5
  ```

- New password is "NEWPASSWORD"

  LM hash of "NEWPASSWORD":

  ```
  09 ee ab 5a a4 15 d6 e4 d4 08 e6 b1 05 74 18 64
  ```

  NT hash of "NEWPASSWORD":

  ```
  25 67 81 a6 20 31 28 9d 3c 2c 98 c1 4f 1e fc 8c
  ```

To demonstrate sample data values for the 7-byte InputKey and 8-byte OutputKey used in section 2.2.11.1.2, the following values are used for the encryption of the old NT hash with the new NT hash shown above.

1. Split the NT hash of the old password into two blocks (2.2.11.1.1):

   ```
   Block 1: 66 77 b2 c3 94 31 13 55
   Block 2: b5 4f 25 ee c5 bf ac f5
   ```

2. Split the NT hash of the new password into two blocks (2.2.11.1.1):

   ```
   Block 1: 25 67 81 a6 20 31 28 9d
   Block 2: 3c 2c 98 c1 4f 1e fc 8c
   ```

3. The 7-byte keys are derived as stated in section 2.2.11.1.4 using the 16-byte hash value. Apply the algorithm in section 2.2.11.1.2 to transform the 7-byte key into an 8-byte key.

   ```
   7 byte InputKey for block 1:  25 67 81 a6 20 31 28
   ```

```
8 byte OutputKey for block 1: 25 b3 e0 34 62 01 c4 51

7 byte InputKey for block 2:  9d 3c 2c 98 c1 4f 1e
8 byte OutputKey for block 2: 9d 9e 0b 92 8c 0b 3d 3d
```

4. Apply [FIPS81] to encrypt both blocks using these keys.

   OldNtOwfEncryptedWithNewNt:

```
da 39 84 64 27 f5 e6 c9 48 2c 8f e9 b3 3a 16 07
```

Likewise, the following values are used for encryption of the old LM hash with the new NT hash.

1. Split the LM hash of the old password into two blocks (2.2.11.1.1):

```
Block 1: c9 b8 1d 93 9d 6f d8 0c
Block 2: d4 08 e6 b1 05 74 18 64
```

2. As before, split the NT hash of the new password into two blocks (2.2.11.1.1):

```
Block 1: 25 67 81 a6 20 31 28 9d
Block 2: 3c 2c 98 c1 4f 1e fc 8c

7 byte InputKey for block 1:  25 67 81 a6 20 31 28
8 byte OutputKey for block 1: 25 b3 e0 34 62 01 c4 51

7 byte InputKey for block 2:  9d 3c 2c 98 c1 4f 1e
8 byte OutputKey for block 2: 9d 9e 0b 92 8c 0b 3d 3d
```

3. Apply [FIPS81] to encrypt both blocks using these keys.

   OldLmOwfEncryptedWithNewNt:

```
80 45 7a 72 72 5a 37 9c ed 8b 07 d2 fd 6f 46 ff
```

5 Security

5.1 Security Considerations for Implementers

Sensitive information, such as the cleartext password for accounts, is communicated through this protocol; therefore, implementers must pay special attention to the secrecy of this data. While this protocol does not use transport-level encryption (with the exception of **SamrValidatePassword**), it does rely on the key strength of the SMB transport for encrypting cleartext data.

Using **SamrSetInformationUser2** with UserInternal4InformationNew and UserInternal5InformationNew is the best choice a requester can make for setting a cleartext password through this protocol because the cryptography used is the strongest in this protocol.

Creating a user object is a multi-step process in this protocol. These steps are outlined in the example in section 4.1. After completing these steps correctly, the responder creates a user object in its abstract database. However, the user object is not usable for authentication in this state. The user object needs to be enabled for authentication. The steps for enabling a user object are outlined in the example in section 4.2. Optionally, a password can be set on the user object. As specified in the previous paragraph, **SamrSetInformationUser2** with UserInternal4InformationNew and UserInternal5InformationNew is the best choice for setting a cleartext password in this protocol.

5.2 Index of Security Parameters

Security Parameter	Section
Service Principal Name for responder	2.1
Encryption algorithm for hashes	2.2.11.1
End-user password (to set)	3.2.5.6.4
End-user password (to change)	3.2.5.10.1
End-user password (to change)	3.2.5.10.2
End-user password (to change)	3.2.5.10.3
Recovery password (to set)	3.2.5.13.7
End-user application password (set, change, and authenticate)	3.2.5.13.8
Encryption key for storing an encrypted LM hash	3.2.1.8.6
Encryption key for storing an encrypted NT hash	3.2.1.8.7

6 Appendix A: Full IDL

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

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

[
    uuid(12345778-1234-ABCD-EF00-0123456789AC),
    version(1.0),
    ms_union,
    pointer_default(unique)
]

interface samr{

typedef struct _RPC_STRING {
    unsigned short Length;
    unsigned short MaximumLength;
    [size_is(MaximumLength), length_is(Length)] char * Buffer;
} RPC_STRING, *PRPC_STRING;


typedef struct _OLD_LARGE_INTEGER {
    unsigned long LowPart;
    long HighPart;
} OLD_LARGE_INTEGER, *POLD_LARGE_INTEGER;

typedef [handle] wchar_t * PSAMPR_SERVER_NAME;

typedef [context_handle] void * SAMPR_HANDLE;

typedef struct _ENCRYPTED_LM_OWF_PASSWORD {
    char data[16];
} ENCRYPTED_LM_OWF_PASSWORD,
  *PENCRYPTED_LM_OWF_PASSWORD,
  ENCRYPTED_NT_OWF_PASSWORD,
  *PENCRYPTED_NT_OWF_PASSWORD;

typedef struct _SAMPR_ULONG_ARRAY {
    unsigned long Count;
    [size_is(Count)] unsigned long * Element;
} SAMPR_ULONG_ARRAY, *PSAMPR_ULONG_ARRAY;

typedef struct _SAMPR_SID_INFORMATION {
    PRPC_SID    SidPointer;
} SAMPR_SID_INFORMATION, *PSAMPR_SID_INFORMATION;

typedef struct _SAMPR_PSID_ARRAY {
    [range(0, 1024)] unsigned long Count;
    [size_is(Count)] PSAMPR_SID_INFORMATION Sids;
} SAMPR_PSID_ARRAY, *PSAMPR_PSID_ARRAY;

typedef struct _SAMPR_RETURNED_USTRING_ARRAY {
    unsigned long Count;
```

```
    [size_is(Count)] PRPC_UNICODE_STRING  Element;
} SAMPR_RETURNED_USTRING_ARRAY, *PSAMPR_RETURNED_USTRING_ARRAY;


typedef enum _SID_NAME_USE {
    SidTypeUser = 1,
    SidTypeGroup,
    SidTypeDomain,
    SidTypeAlias,
    SidTypeWellKnownGroup,
    SidTypeDeletedAccount,
    SidTypeInvalid,
    SidTypeUnknown,
} SID_NAME_USE, *PSID_NAME_USE;

typedef struct _SAMPR_RID_ENUMERATION {
    unsigned long RelativeId;
    RPC_UNICODE_STRING Name;
} SAMPR_RID_ENUMERATION, *PSAMPR_RID_ENUMERATION;

typedef struct _SAMPR_ENUMERATION_BUFFER {
    unsigned long EntriesRead;
    [size_is(EntriesRead)] PSAMPR_RID_ENUMERATION Buffer;
} SAMPR_ENUMERATION_BUFFER, *PSAMPR_ENUMERATION_BUFFER;

typedef struct _SAMPR_SR_SECURITY_DESCRIPTOR {
    [range(0, 256 * 1024)] unsigned long Length;
    [size_is(Length)] unsigned char* SecurityDescriptor;
} SAMPR_SR_SECURITY_DESCRIPTOR, *PSAMPR_SR_SECURITY_DESCRIPTOR;

typedef struct _GROUP_MEMBERSHIP {
    unsigned long RelativeId;
    unsigned long Attributes;
} GROUP_MEMBERSHIP, *PGROUP_MEMBERSHIP;

typedef struct _SAMPR_GET_GROUPS_BUFFER {
    unsigned long MembershipCount;
    [size_is(MembershipCount)] PGROUP_MEMBERSHIP Groups;
} SAMPR_GET_GROUPS_BUFFER, *PSAMPR_GET_GROUPS_BUFFER;

typedef struct _SAMPR_GET_MEMBERS_BUFFER {
    unsigned long MemberCount;
    [size_is(MemberCount)] unsigned long* Members;
    [size_is(MemberCount)] unsigned long* Attributes;
} SAMPR_GET_MEMBERS_BUFFER, *PSAMPR_GET_MEMBERS_BUFFER;

typedef struct _SAMPR_REVISION_INFO_V1 {
    unsigned long Revision;
    unsigned long SupportedFeatures;
} SAMPR_REVISION_INFO_V1, *PSAMPR_REVISION_INFO_V1;

typedef [switch_type(unsigned long)] union {
    [case(1)] SAMPR_REVISION_INFO_V1 V1;
}SAMPR_REVISION_INFO, *PSAMPR_REVISION_INFO;

typedef struct _USER_DOMAIN_PASSWORD_INFORMATION {
    unsigned short MinPasswordLength;
    unsigned long PasswordProperties;
```

```
} USER_DOMAIN_PASSWORD_INFORMATION,
  *PUSER_DOMAIN_PASSWORD_INFORMATION;

typedef enum _DOMAIN_SERVER_ENABLE_STATE {
    DomainServerEnabled = 1,
    DomainServerDisabled
} DOMAIN_SERVER_ENABLE_STATE, *PDOMAIN_SERVER_ENABLE_STATE;

typedef struct _DOMAIN_STATE_INFORMATION {
    DOMAIN_SERVER_ENABLE_STATE DomainServerState;
} DOMAIN_STATE_INFORMATION, *PDOMAIN_STATE_INFORMATION;

typedef enum _DOMAIN_SERVER_ROLE {
    DomainServerRoleBackup = 2,
    DomainServerRolePrimary = 3
} DOMAIN_SERVER_ROLE, *PDOMAIN_SERVER_ROLE;

typedef struct _DOMAIN_PASSWORD_INFORMATION {
    unsigned short MinPasswordLength;
    unsigned short PasswordHistoryLength;
    unsigned long PasswordProperties;
    OLD_LARGE_INTEGER MaxPasswordAge;
    OLD_LARGE_INTEGER MinPasswordAge;
} DOMAIN_PASSWORD_INFORMATION, *PDOMAIN_PASSWORD_INFORMATION;

typedef struct _DOMAIN_LOGOFF_INFORMATION {
    OLD_LARGE_INTEGER ForceLogoff;
} DOMAIN_LOGOFF_INFORMATION, *PDOMAIN_LOGOFF_INFORMATION;

typedef struct _DOMAIN_SERVER_ROLE_INFORMATION {
    DOMAIN_SERVER_ROLE DomainServerRole;
} DOMAIN_SERVER_ROLE_INFORMATION, *PDOMAIN_SERVER_ROLE_INFORMATION;

typedef struct _DOMAIN_MODIFIED_INFORMATION {
    OLD_LARGE_INTEGER DomainModifiedCount;
    OLD_LARGE_INTEGER CreationTime;
} DOMAIN_MODIFIED_INFORMATION, *PDOMAIN_MODIFIED_INFORMATION;

typedef struct _DOMAIN_MODIFIED_INFORMATION2 {
    OLD_LARGE_INTEGER DomainModifiedCount;
    OLD_LARGE_INTEGER CreationTime;
    OLD_LARGE_INTEGER ModifiedCountAtLastPromotion;
} DOMAIN_MODIFIED_INFORMATION2, *PDOMAIN_MODIFIED_INFORMATION2;

typedef struct _SAMPR_DOMAIN_GENERAL_INFORMATION {
    OLD_LARGE_INTEGER          ForceLogoff;
    RPC_UNICODE_STRING         OemInformation;
    RPC_UNICODE_STRING         DomainName;
    RPC_UNICODE_STRING         ReplicaSourceNodeName;
    OLD_LARGE_INTEGER          DomainModifiedCount;
    DOMAIN_SERVER_ENABLE_STATE DomainServerState;
    DOMAIN_SERVER_ROLE         DomainServerRole;
    unsigned char              UasCompatibilityRequired;
    unsigned long              UserCount;
    unsigned long              GroupCount;
    unsigned long              AliasCount;
} SAMPR_DOMAIN_GENERAL_INFORMATION,
  *PSAMPR_DOMAIN_GENERAL_INFORMATION;
```

```
typedef struct _SAMPR_DOMAIN_GENERAL_INFORMATION2 {
    SAMPR_DOMAIN_GENERAL_INFORMATION  I1;
    OLD_LARGE_INTEGER                 LockoutDuration;
    OLD_LARGE_INTEGER                 LockoutObservationWindow;
    unsigned short                    LockoutThreshold;
} SAMPR_DOMAIN_GENERAL_INFORMATION2,
  *PSAMPR_DOMAIN_GENERAL_INFORMATION2;

typedef struct _SAMPR_DOMAIN_OEM_INFORMATION {
    RPC_UNICODE_STRING OemInformation;
} SAMPR_DOMAIN_OEM_INFORMATION, *PSAMPR_DOMAIN_OEM_INFORMATION;

typedef struct _SAMPR_DOMAIN_NAME_INFORMATION {
    RPC_UNICODE_STRING DomainName;
} SAMPR_DOMAIN_NAME_INFORMATION, *PSAMPR_DOMAIN_NAME_INFORMATION;

typedef struct SAMPR_DOMAIN_REPLICATION_INFORMATION {
    RPC_UNICODE_STRING ReplicaSourceNodeName;
} SAMPR_DOMAIN_REPLICATION_INFORMATION,
  *PSAMPR_DOMAIN_REPLICATION_INFORMATION;

typedef struct _SAMPR_DOMAIN_LOCKOUT_INFORMATION {
    OLD_LARGE_INTEGER  LockoutDuration;
    OLD_LARGE_INTEGER  LockoutObservationWindow;
    unsigned short     LockoutThreshold;
} SAMPR_DOMAIN_LOCKOUT_INFORMATION,
  *PSAMPR_DOMAIN_LOCKOUT_INFORMATION;

typedef enum _DOMAIN_INFORMATION_CLASS {
    DomainPasswordInformation = 1,
    DomainGeneralInformation = 2,
    DomainLogoffInformation = 3,
    DomainOemInformation = 4,
    DomainNameInformation = 5,
    DomainReplicationInformation = 6,
    DomainServerRoleInformation = 7,
    DomainModifiedInformation = 8,
    DomainStateInformation = 9,
    DomainUasInformation = 10,
    DomainGeneralInformation2 = 11,
    DomainLockoutInformation = 12,
    DomainModifiedInformation2 = 13
} DOMAIN_INFORMATION_CLASS;

typedef [switch_type(DOMAIN_INFORMATION_CLASS)] union
_SAMPR_DOMAIN_INFO_BUFFER {
    [case(DomainPasswordInformation)]
        DOMAIN_PASSWORD_INFORMATION          Password;
    [case(DomainGeneralInformation)]
        SAMPR_DOMAIN_GENERAL_INFORMATION     General;
    [case(DomainLogoffInformation)]
        DOMAIN_LOGOFF_INFORMATION            Logoff;
    [case(DomainOemInformation)]
        SAMPR_DOMAIN_OEM_INFORMATION         Oem;
    [case(DomainNameInformation)]
        SAMPR_DOMAIN_NAME_INFORMATION        Name;
    [case(DomainServerRoleInformation)]
```

```
        DOMAIN_SERVER_ROLE_INFORMATION       Role;
    [case(DomainReplicationInformation)]
        SAMPR_DOMAIN_REPLICATION_INFORMATION Replication;
    [case(DomainModifiedInformation)]
        DOMAIN_MODIFIED_INFORMATION          Modified;
    [case(DomainStateInformation)]
        DOMAIN_STATE_INFORMATION             State;
    [case(DomainGeneralInformation2)]
        SAMPR_DOMAIN_GENERAL_INFORMATION2    General2;
    [case(DomainLockoutInformation)]
        SAMPR_DOMAIN_LOCKOUT_INFORMATION     Lockout;
    [case(DomainModifiedInformation2)]
        DOMAIN_MODIFIED_INFORMATION2         Modified2;
} SAMPR_DOMAIN_INFO_BUFFER, *PSAMPR_DOMAIN_INFO_BUFFER;

typedef enum _DOMAIN_DISPLAY_INFORMATION {
    DomainDisplayUser = 1,
    DomainDisplayMachine,
    DomainDisplayGroup,
    DomainDisplayOemUser,
    DomainDisplayOemGroup
} DOMAIN_DISPLAY_INFORMATION, *PDOMAIN_DISPLAY_INFORMATION;

typedef struct _SAMPR_DOMAIN_DISPLAY_USER {
    unsigned long      Index;
    unsigned long      Rid;
    unsigned long      AccountControl;
    RPC_UNICODE_STRING AccountName;
    RPC_UNICODE_STRING AdminComment;
    RPC_UNICODE_STRING FullName;
} SAMPR_DOMAIN_DISPLAY_USER, *PSAMPR_DOMAIN_DISPLAY_USER;

typedef struct _SAMPR_DOMAIN_DISPLAY_MACHINE {
    unsigned long      Index;
    unsigned long      Rid;
    unsigned long      AccountControl;
    RPC_UNICODE_STRING AccountName;
    RPC_UNICODE_STRING AdminComment;
} SAMPR_DOMAIN_DISPLAY_MACHINE, *PSAMPR_DOMAIN_DISPLAY_MACHINE;

typedef struct _SAMPR_DOMAIN_DISPLAY_GROUP {
    unsigned long      Index;
    unsigned long      Rid;
    unsigned long      Attributes;
    RPC_UNICODE_STRING AccountName;
    RPC_UNICODE_STRING AdminComment;
} SAMPR_DOMAIN_DISPLAY_GROUP, *PSAMPR_DOMAIN_DISPLAY_GROUP;

typedef struct _SAMPR_DOMAIN_DISPLAY_OEM_USER {
    unsigned long           Index;
    RPC_STRING              OemAccountName;
} SAMPR_DOMAIN_DISPLAY_OEM_USER, *PSAMPR_DOMAIN_DISPLAY_OEM_USER;

typedef struct _SAMPR_DOMAIN_DISPLAY_OEM_GROUP {
    unsigned long           Index;
    RPC_STRING              OemAccountName;
} SAMPR_DOMAIN_DISPLAY_OEM_GROUP, *PSAMPR_DOMAIN_DISPLAY_OEM_GROUP;
```

```
typedef struct _SAMPR_DOMAIN_DISPLAY_USER_BUFFER {
    unsigned long EntriesRead;
    [size_is(EntriesRead)] PSAMPR_DOMAIN_DISPLAY_USER Buffer;
} SAMPR_DOMAIN_DISPLAY_USER_BUFFER,
  *PSAMPR_DOMAIN_DISPLAY_USER_BUFFER;

typedef struct _SAMPR_DOMAIN_DISPLAY_MACHINE_BUFFER {
    unsigned long EntriesRead;
    [size_is(EntriesRead)] PSAMPR_DOMAIN_DISPLAY_MACHINE Buffer;
} SAMPR_DOMAIN_DISPLAY_MACHINE_BUFFER,
  *PSAMPR_DOMAIN_DISPLAY_MACHINE_BUFFER;

typedef struct _SAMPR_DOMAIN_DISPLAY_GROUP_BUFFER {
    unsigned long EntriesRead;
    [size_is(EntriesRead)] PSAMPR_DOMAIN_DISPLAY_GROUP Buffer;
} SAMPR_DOMAIN_DISPLAY_GROUP_BUFFER,
  *PSAMPR_DOMAIN_DISPLAY_GROUP_BUFFER;

typedef struct _SAMPR_DOMAIN_DISPLAY_OEM_USER_BUFFER {
    unsigned long EntriesRead;
    [size_is(EntriesRead)] PSAMPR_DOMAIN_DISPLAY_OEM_USER Buffer;
} SAMPR_DOMAIN_DISPLAY_OEM_USER_BUFFER,
  *PSAMPR_DOMAIN_DISPLAY_OEM_USER_BUFFER;

typedef struct _SAMPR_DOMAIN_DISPLAY_OEM_GROUP_BUFFER {
    unsigned long EntriesRead;
    [size_is(EntriesRead)] PSAMPR_DOMAIN_DISPLAY_OEM_GROUP Buffer;
} SAMPR_DOMAIN_DISPLAY_OEM_GROUP_BUFFER,
  *PSAMPR_DOMAIN_DISPLAY_OEM_GROUP_BUFFER;

typedef [switch_type(DOMAIN_DISPLAY_INFORMATION)] union
_SAMPR_DISPLAY_INFO_BUFFER {
    [case(DomainDisplayUser)]
        SAMPR_DOMAIN_DISPLAY_USER_BUFFER      UserInformation;
    [case(DomainDisplayMachine)]
        SAMPR_DOMAIN_DISPLAY_MACHINE_BUFFER   MachineInformation;
    [case(DomainDisplayGroup)]
        SAMPR_DOMAIN_DISPLAY_GROUP_BUFFER     GroupInformation;
    [case(DomainDisplayOemUser)]
        SAMPR_DOMAIN_DISPLAY_OEM_USER_BUFFER  OemUserInformation;
    [case(DomainDisplayOemGroup)]
        SAMPR_DOMAIN_DISPLAY_OEM_GROUP_BUFFER OemGroupInformation;
} SAMPR_DISPLAY_INFO_BUFFER, *PSAMPR_DISPLAY_INFO_BUFFER;

typedef struct _GROUP_ATTRIBUTE_INFORMATION {
    unsigned long Attributes;
} GROUP_ATTRIBUTE_INFORMATION, *PGROUP_ATTRIBUTE_INFORMATION;

typedef struct _SAMPR_GROUP_GENERAL_INFORMATION {
    RPC_UNICODE_STRING Name;
    unsigned long Attributes;
    unsigned long MemberCount;
    RPC_UNICODE_STRING AdminComment;
} SAMPR_GROUP_GENERAL_INFORMATION,
  *PSAMPR_GROUP_GENERAL_INFORMATION;

typedef struct _SAMPR_GROUP_NAME_INFORMATION {
    RPC_UNICODE_STRING Name;
```

```
} SAMPR_GROUP_NAME_INFORMATION, *PSAMPR_GROUP_NAME_INFORMATION;

typedef struct _SAMPR_GROUP_ADM_COMMENT_INFORMATION {
    RPC_UNICODE_STRING AdminComment;
} SAMPR_GROUP_ADM_COMMENT_INFORMATION,
  *PSAMPR_GROUP_ADM_COMMENT_INFORMATION;

typedef enum _GROUP_INFORMATION_CLASS {
    GroupGeneralInformation = 1,
    GroupNameInformation,
    GroupAttributeInformation,
    GroupAdminCommentInformation,
    GroupReplicationInformation
} GROUP_INFORMATION_CLASS;

typedef [switch_type(GROUP_INFORMATION_CLASS)] union
_SAMPR_GROUP_INFO_BUFFER {
    [case(GroupGeneralInformation)]
        SAMPR_GROUP_GENERAL_INFORMATION     General;
    [case(GroupNameInformation)]
        SAMPR_GROUP_NAME_INFORMATION        Name;
    [case(GroupAttributeInformation)]
        GROUP_ATTRIBUTE_INFORMATION         Attribute;
    [case(GroupAdminCommentInformation)]
        SAMPR_GROUP_ADM_COMMENT_INFORMATION AdminComment;
    [case(GroupReplicationInformation)]
        SAMPR_GROUP_GENERAL_INFORMATION     DoNotUse;
} SAMPR_GROUP_INFO_BUFFER, *PSAMPR_GROUP_INFO_BUFFER;

typedef struct _SAMPR_ALIAS_GENERAL_INFORMATION {
    RPC_UNICODE_STRING Name;
    unsigned long MemberCount;
    RPC_UNICODE_STRING AdminComment;
} SAMPR_ALIAS_GENERAL_INFORMATION,
  *PSAMPR_ALIAS_GENERAL_INFORMATION;

typedef struct _SAMPR_ALIAS_NAME_INFORMATION {
    RPC_UNICODE_STRING Name;
} SAMPR_ALIAS_NAME_INFORMATION, *PSAMPR_ALIAS_NAME_INFORMATION;

typedef struct _SAMPR_ALIAS_ADM_COMMENT_INFORMATION {
    RPC_UNICODE_STRING AdminComment;
} SAMPR_ALIAS_ADM_COMMENT_INFORMATION,
  *PSAMPR_ALIAS_ADM_COMMENT_INFORMATION;

typedef enum _ALIAS_INFORMATION_CLASS {
    AliasGeneralInformation = 1,
    AliasNameInformation,
    AliasAdminCommentInformation,
} ALIAS_INFORMATION_CLASS;

typedef [switch_type(ALIAS_INFORMATION_CLASS)] union
_SAMPR_ALIAS_INFO_BUFFER {
    [case(AliasGeneralInformation)]
        SAMPR_ALIAS_GENERAL_INFORMATION     General;
    [case(AliasNameInformation)]
        SAMPR_ALIAS_NAME_INFORMATION        Name;
    [case(AliasAdminCommentInformation)]
```

```
        SAMPR_ALIAS_ADM_COMMENT_INFORMATION AdminComment;
} SAMPR_ALIAS_INFO_BUFFER, *PSAMPR_ALIAS_INFO_BUFFER;

typedef struct _SAMPR_ENCRYPTED_USER_PASSWORD {
    unsigned char Buffer[ (256 * 2) + 4 ];
} SAMPR_ENCRYPTED_USER_PASSWORD, *PSAMPR_ENCRYPTED_USER_PASSWORD;

typedef struct _SAMPR_ENCRYPTED_USER_PASSWORD_NEW {
    unsigned char Buffer[ (256 * 2) + 4 + 16];
} SAMPR_ENCRYPTED_USER_PASSWORD_NEW,
  *PSAMPR_ENCRYPTED_USER_PASSWORD_NEW;

typedef struct _USER_PRIMARY_GROUP_INFORMATION {
    unsigned long PrimaryGroupId;
} USER_PRIMARY_GROUP_INFORMATION, *PUSER_PRIMARY_GROUP_INFORMATION;

typedef struct _USER_CONTROL_INFORMATION {
    unsigned long UserAccountControl;
} USER_CONTROL_INFORMATION, *PUSER_CONTROL_INFORMATION;

typedef struct _USER_EXPIRES_INFORMATION {
    OLD_LARGE_INTEGER AccountExpires;
} USER_EXPIRES_INFORMATION, *PUSER_EXPIRES_INFORMATION;

typedef struct _SAMPR_LOGON_HOURS {
    unsigned short UnitsPerWeek;
    [size_is(1260), length_is((UnitsPerWeek+7)/8)]
        unsigned char*  LogonHours;
} SAMPR_LOGON_HOURS, *PSAMPR_LOGON_HOURS;

typedef struct _SAMPR_USER_ALL_INFORMATION {
    OLD_LARGE_INTEGER LastLogon;
    OLD_LARGE_INTEGER LastLogoff;
    OLD_LARGE_INTEGER PasswordLastSet;
    OLD_LARGE_INTEGER AccountExpires;
    OLD_LARGE_INTEGER PasswordCanChange;
    OLD_LARGE_INTEGER PasswordMustChange;
    RPC_UNICODE_STRING UserName;
    RPC_UNICODE_STRING FullName;
    RPC_UNICODE_STRING HomeDirectory;
    RPC_UNICODE_STRING HomeDirectoryDrive;
    RPC_UNICODE_STRING ScriptPath;
    RPC_UNICODE_STRING ProfilePath;
    RPC_UNICODE_STRING AdminComment;
    RPC_UNICODE_STRING WorkStations;
    RPC_UNICODE_STRING UserComment;
    RPC_UNICODE_STRING Parameters;
    RPC_UNICODE_STRING LmOwfPassword;
    RPC_UNICODE_STRING NtOwfPassword;
    RPC_UNICODE_STRING PrivateData;
    SAMPR_SR_SECURITY_DESCRIPTOR SecurityDescriptor;
    unsigned long UserId;
    unsigned long PrimaryGroupId;
    unsigned long UserAccountControl;
    unsigned long WhichFields;
    SAMPR_LOGON_HOURS LogonHours;
    unsigned short BadPasswordCount;
    unsigned short LogonCount;
```

```
    unsigned short CountryCode;
    unsigned short CodePage;
    unsigned char LmPasswordPresent;
    unsigned char NtPasswordPresent;
    unsigned char PasswordExpired;
    unsigned char PrivateDataSensitive;
} SAMPR_USER_ALL_INFORMATION, *PSAMPR_USER_ALL_INFORMATION;

typedef struct _SAMPR_USER_GENERAL_INFORMATION {
    RPC_UNICODE_STRING UserName;
    RPC_UNICODE_STRING FullName;
    unsigned long PrimaryGroupId;
    RPC_UNICODE_STRING AdminComment;
    RPC_UNICODE_STRING UserComment;
} SAMPR_USER_GENERAL_INFORMATION, *PSAMPR_USER_GENERAL_INFORMATION;

typedef struct _SAMPR_USER_PREFERENCES_INFORMATION {
    RPC_UNICODE_STRING UserComment;
    RPC_UNICODE_STRING Reserved1;
    unsigned short CountryCode;
    unsigned short CodePage;
} SAMPR_USER_PREFERENCES_INFORMATION,
  *PSAMPR_USER_PREFERENCES_INFORMATION;

typedef struct _SAMPR_USER_PARAMETERS_INFORMATION {
    RPC_UNICODE_STRING Parameters;
} SAMPR_USER_PARAMETERS_INFORMATION,
  *PSAMPR_USER_PARAMETERS_INFORMATION;

typedef struct _SAMPR_USER_LOGON_INFORMATION {
    RPC_UNICODE_STRING UserName;
    RPC_UNICODE_STRING FullName;
    unsigned long UserId;
    unsigned long PrimaryGroupId;
    RPC_UNICODE_STRING HomeDirectory;
    RPC_UNICODE_STRING HomeDirectoryDrive;
    RPC_UNICODE_STRING ScriptPath;
    RPC_UNICODE_STRING ProfilePath;
    RPC_UNICODE_STRING WorkStations;
    OLD_LARGE_INTEGER LastLogon;
    OLD_LARGE_INTEGER LastLogoff;
    OLD_LARGE_INTEGER PasswordLastSet;
    OLD_LARGE_INTEGER PasswordCanChange;
    OLD_LARGE_INTEGER PasswordMustChange;
    SAMPR_LOGON_HOURS LogonHours;
    unsigned short BadPasswordCount;
    unsigned short LogonCount;
    unsigned long UserAccountControl;
} SAMPR_USER_LOGON_INFORMATION, *PSAMPR_USER_LOGON_INFORMATION;

typedef struct _SAMPR_USER_ACCOUNT_INFORMATION {
    RPC_UNICODE_STRING UserName;
    RPC_UNICODE_STRING FullName;
    unsigned long UserId;
    unsigned long PrimaryGroupId;
    RPC_UNICODE_STRING HomeDirectory;
    RPC_UNICODE_STRING HomeDirectoryDrive;
    RPC_UNICODE_STRING ScriptPath;
```

```
    RPC_UNICODE_STRING ProfilePath;
    RPC_UNICODE_STRING AdminComment;
    RPC_UNICODE_STRING WorkStations;
    OLD_LARGE_INTEGER LastLogon;
    OLD_LARGE_INTEGER LastLogoff;
    SAMPR_LOGON_HOURS LogonHours;
    unsigned short BadPasswordCount;
    unsigned short LogonCount;
    OLD_LARGE_INTEGER PasswordLastSet;
    OLD_LARGE_INTEGER AccountExpires;
    unsigned long UserAccountControl;
} SAMPR_USER_ACCOUNT_INFORMATION, *PSAMPR_USER_ACCOUNT_INFORMATION;

typedef struct _SAMPR_USER_A_NAME_INFORMATION {
    RPC_UNICODE_STRING UserName;
} SAMPR_USER_A_NAME_INFORMATION, *PSAMPR_USER_A_NAME_INFORMATION;

typedef struct _SAMPR_USER_F_NAME_INFORMATION {
    RPC_UNICODE_STRING FullName;
} SAMPR_USER_F_NAME_INFORMATION, *PSAMPR_USER_F_NAME_INFORMATION;

typedef struct _SAMPR_USER_NAME_INFORMATION {
    RPC_UNICODE_STRING UserName;
    RPC_UNICODE_STRING FullName;
} SAMPR_USER_NAME_INFORMATION, *PSAMPR_USER_NAME_INFORMATION;

typedef struct _SAMPR_USER_HOME_INFORMATION {
    RPC_UNICODE_STRING HomeDirectory;
    RPC_UNICODE_STRING HomeDirectoryDrive;
} SAMPR_USER_HOME_INFORMATION, *PSAMPR_USER_HOME_INFORMATION;

typedef struct _SAMPR_USER_SCRIPT_INFORMATION {
    RPC_UNICODE_STRING ScriptPath;
} SAMPR_USER_SCRIPT_INFORMATION, *PSAMPR_USER_SCRIPT_INFORMATION;

typedef struct _SAMPR_USER_PROFILE_INFORMATION {
    RPC_UNICODE_STRING ProfilePath;
} SAMPR_USER_PROFILE_INFORMATION, *PSAMPR_USER_PROFILE_INFORMATION;

typedef struct _SAMPR_USER_ADMIN_COMMENT_INFORMATION {
    RPC_UNICODE_STRING AdminComment;
} SAMPR_USER_ADMIN_COMMENT_INFORMATION,
  *PSAMPR_USER_ADMIN_COMMENT_INFORMATION;

typedef struct _SAMPR_USER_WORKSTATIONS_INFORMATION {
    RPC_UNICODE_STRING WorkStations;
} SAMPR_USER_WORKSTATIONS_INFORMATION,
  *PSAMPR_USER_WORKSTATIONS_INFORMATION;

typedef struct _SAMPR_USER_LOGON_HOURS_INFORMATION {
    SAMPR_LOGON_HOURS LogonHours;
} SAMPR_USER_LOGON_HOURS_INFORMATION,
  *PSAMPR_USER_LOGON_HOURS_INFORMATION;

typedef struct _SAMPR_USER_INTERNAL1_INFORMATION {
    ENCRYPTED_NT_OWF_PASSWORD  EncryptedNtOwfPassword;
    ENCRYPTED_LM_OWF_PASSWORD  EncryptedLmOwfPassword;
    unsigned char              NtPasswordPresent;
```

```
    unsigned char              LmPasswordPresent;
    unsigned char              PasswordExpired;
} SAMPR_USER_INTERNAL1_INFORMATION,
  *PSAMPR_USER_INTERNAL1_INFORMATION;

typedef struct _SAMPR_USER_INTERNAL4_INFORMATION {
    SAMPR_USER_ALL_INFORMATION I1;
    SAMPR_ENCRYPTED_USER_PASSWORD UserPassword;
} SAMPR_USER_INTERNAL4_INFORMATION,
  *PSAMPR_USER_INTERNAL4_INFORMATION;

typedef struct _SAMPR_USER_INTERNAL4_INFORMATION_NEW {
    SAMPR_USER_ALL_INFORMATION I1;
    SAMPR_ENCRYPTED_USER_PASSWORD_NEW UserPassword;
} SAMPR_USER_INTERNAL4_INFORMATION_NEW,
  *PSAMPR_USER_INTERNAL4_INFORMATION_NEW;

typedef struct _SAMPR_USER_INTERNAL5_INFORMATION {
    SAMPR_ENCRYPTED_USER_PASSWORD UserPassword;
    unsigned char                 PasswordExpired;
} SAMPR_USER_INTERNAL5_INFORMATION,
  *PSAMPR_USER_INTERNAL5_INFORMATION;

typedef struct _SAMPR_USER_INTERNAL5_INFORMATION_NEW {
    SAMPR_ENCRYPTED_USER_PASSWORD_NEW UserPassword;
    unsigned char                     PasswordExpired;
} SAMPR_USER_INTERNAL5_INFORMATION_NEW,
  *PSAMPR_USER_INTERNAL5_INFORMATION_NEW;

typedef enum _USER_INFORMATION_CLASS {
    UserGeneralInformation = 1,
    UserPreferencesInformation = 2,
    UserLogonInformation = 3,
    UserLogonHoursInformation = 4,
    UserAccountInformation = 5,
    UserNameInformation = 6,
    UserAccountNameInformation = 7,
    UserFullNameInformation = 8,
    UserPrimaryGroupInformation = 9,
    UserHomeInformation = 10,
    UserScriptInformation = 11,
    UserProfileInformation = 12,
    UserAdminCommentInformation = 13,
    UserWorkStationsInformation = 14,
    UserControlInformation = 16,
    UserExpiresInformation = 17,
    UserInternal1Information = 18,
    UserParametersInformation = 20,
    UserAllInformation = 21,
    UserInternal4Information = 23,
    UserInternal5Information = 24,
    UserInternal4InformationNew = 25,
    UserInternal5InformationNew = 26,
} USER_INFORMATION_CLASS, *PUSER_INFORMATION_CLASS;

typedef [switch_type(USER_INFORMATION_CLASS)] union
_SAMPR_USER_INFO_BUFFER {
    [case(UserGeneralInformation)]
```

```
        SAMPR_USER_GENERAL_INFORMATION        General;
    [case(UserPreferencesInformation)]
        SAMPR_USER_PREFERENCES_INFORMATION    Preferences;
    [case(UserLogonInformation)]
        SAMPR_USER_LOGON_INFORMATION          Logon;
    [case(UserLogonHoursInformation)]
        SAMPR_USER_LOGON_HOURS_INFORMATION    LogonHours;
    [case(UserAccountInformation)]
        SAMPR_USER_ACCOUNT_INFORMATION        Account;
    [case(UserNameInformation)]
        SAMPR_USER_NAME_INFORMATION           Name;
    [case(UserAccountNameInformation)]
        SAMPR_USER_A_NAME_INFORMATION         AccountName;
    [case(UserFullNameInformation)]
        SAMPR_USER_F_NAME_INFORMATION         FullName;
    [case(UserPrimaryGroupInformation)]
        USER_PRIMARY_GROUP_INFORMATION        PrimaryGroup;
    [case(UserHomeInformation)]
        SAMPR_USER_HOME_INFORMATION           Home;
    [case(UserScriptInformation)]
        SAMPR_USER_SCRIPT_INFORMATION         Script;
    [case(UserProfileInformation)]
        SAMPR_USER_PROFILE_INFORMATION        Profile;
    [case(UserAdminCommentInformation)]
        SAMPR_USER_ADMIN_COMMENT_INFORMATION  AdminComment;
    [case(UserWorkStationsInformation)]
        SAMPR_USER_WORKSTATIONS_INFORMATION   WorkStations;
    [case(UserControlInformation)]
        USER_CONTROL_INFORMATION              Control;
    [case(UserExpiresInformation)]
        USER_EXPIRES_INFORMATION              Expires;
    [case(UserInternal1Information)]
        SAMPR_USER_INTERNAL1_INFORMATION      Internal1;
    [case(UserParametersInformation)]
        SAMPR_USER_PARAMETERS_INFORMATION     Parameters;
    [case(UserAllInformation)]
        SAMPR_USER_ALL_INFORMATION            All;
    [case(UserInternal4Information)]
        SAMPR_USER_INTERNAL4_INFORMATION      Internal4;
    [case(UserInternal5Information)]
        SAMPR_USER_INTERNAL5_INFORMATION      Internal5;
    [case(UserInternal4InformationNew)]
        SAMPR_USER_INTERNAL4_INFORMATION_NEW  Internal4New;
    [case(UserInternal5InformationNew)]
        SAMPR_USER_INTERNAL5_INFORMATION_NEW  Internal5New;
} SAMPR_USER_INFO_BUFFER, *PSAMPR_USER_INFO_BUFFER;

typedef enum _PASSWORD_POLICY_VALIDATION_TYPE{
    SamValidateAuthentication = 1,
    SamValidatePasswordChange,
    SamValidatePasswordReset
} PASSWORD_POLICY_VALIDATION_TYPE;

typedef struct _SAM_VALIDATE_PASSWORD_HASH{
    unsigned long Length;
    [unique,size_is(Length)]
    unsigned char* Hash;
} SAM_VALIDATE_PASSWORD_HASH, *PSAM_VALIDATE_PASSWORD_HASH;
```

```
typedef struct _SAM_VALIDATE_PERSISTED_FIELDS{
    unsigned long PresentFields;
    LARGE_INTEGER PasswordLastSet;
    LARGE_INTEGER BadPasswordTime;
    LARGE_INTEGER LockoutTime;
    unsigned long BadPasswordCount;
    unsigned long PasswordHistoryLength;
    [unique,size_is(PasswordHistoryLength)]
    PSAM_VALIDATE_PASSWORD_HASH PasswordHistory;
} SAM_VALIDATE_PERSISTED_FIELDS, *PSAM_VALIDATE_PERSISTED_FIELDS;

typedef enum _SAM_VALIDATE_VALIDATION_STATUS{
    SamValidateSuccess = 0,
    SamValidatePasswordMustChange,
    SamValidateAccountLockedOut,
    SamValidatePasswordExpired,
    SamValidatePasswordIncorrect,
    SamValidatePasswordIsInHistory,
    SamValidatePasswordTooShort,
    SamValidatePasswordTooLong,
    SamValidatePasswordNotComplexEnough,
    SamValidatePasswordTooRecent,
    SamValidatePasswordFilterError
} SAM_VALIDATE_VALIDATION_STATUS, *PSAM_VALIDATE_VALIDATION_STATUS;

typedef struct _SAM_VALIDATE_STANDARD_OUTPUT_ARG{
    SAM_VALIDATE_PERSISTED_FIELDS ChangedPersistedFields;
    SAM_VALIDATE_VALIDATION_STATUS ValidationStatus;
} SAM_VALIDATE_STANDARD_OUTPUT_ARG,
  *PSAM_VALIDATE_STANDARD_OUTPUT_ARG;

typedef struct _SAM_VALIDATE_AUTHENTICATION_INPUT_ARG{
    SAM_VALIDATE_PERSISTED_FIELDS InputPersistedFields;
    unsigned char PasswordMatched;
} SAM_VALIDATE_AUTHENTICATION_INPUT_ARG,
  *PSAM_VALIDATE_AUTHENTICATION_INPUT_ARG;

typedef struct  _SAM_VALIDATE_PASSWORD_CHANGE_INPUT_ARG{
    SAM_VALIDATE_PERSISTED_FIELDS InputPersistedFields;
    RPC_UNICODE_STRING ClearPassword;
    RPC_UNICODE_STRING UserAccountName;
    SAM_VALIDATE_PASSWORD_HASH HashedPassword;
    unsigned char PasswordMatch;
} SAM_VALIDATE_PASSWORD_CHANGE_INPUT_ARG,
  *PSAM_VALIDATE_PASSWORD_CHANGE_INPUT_ARG;

typedef struct _SAM_VALIDATE_PASSWORD_RESET_INPUT_ARG{
    SAM_VALIDATE_PERSISTED_FIELDS InputPersistedFields;
    RPC_UNICODE_STRING ClearPassword;
    RPC_UNICODE_STRING UserAccountName;
    SAM_VALIDATE_PASSWORD_HASH HashedPassword;
    unsigned char  PasswordMustChangeAtNextLogon;
    unsigned char  ClearLockout;
} SAM_VALIDATE_PASSWORD_RESET_INPUT_ARG,
  *PSAM_VALIDATE_PASSWORD_RESET_INPUT_ARG;

typedef
```

```
[switch_type(PASSWORD_POLICY_VALIDATION_TYPE)]
union _SAM_VALIDATE_INPUT_ARG{
    [case(SamValidateAuthentication)]
        SAM_VALIDATE_AUTHENTICATION_INPUT_ARG
        ValidateAuthenticationInput;
    [case(SamValidatePasswordChange)]
        SAM_VALIDATE_PASSWORD_CHANGE_INPUT_ARG
        ValidatePasswordChangeInput;
    [case(SamValidatePasswordReset)]
        SAM_VALIDATE_PASSWORD_RESET_INPUT_ARG
        ValidatePasswordResetInput;
} SAM_VALIDATE_INPUT_ARG, *PSAM_VALIDATE_INPUT_ARG;

typedef
[switch_type(PASSWORD_POLICY_VALIDATION_TYPE)]
union _SAM_VALIDATE_OUTPUT_ARG{
    [case(SamValidateAuthentication)]
        SAM_VALIDATE_STANDARD_OUTPUT_ARG
        ValidateAuthenticationOutput;
    [case(SamValidatePasswordChange)]
        SAM_VALIDATE_STANDARD_OUTPUT_ARG
        ValidatePasswordChangeOutput;
    [case(SamValidatePasswordReset)]
        SAM_VALIDATE_STANDARD_OUTPUT_ARG
        ValidatePasswordResetOutput;
} SAM_VALIDATE_OUTPUT_ARG, *PSAM_VALIDATE_OUTPUT_ARG;


// opnum 0
long SamrConnect(
    [in, unique] PSAMPR_SERVER_NAME ServerName,
    [out] SAMPR_HANDLE * ServerHandle,
    [in] unsigned long DesiredAccess
    );

// opnum 1
long
SamrCloseHandle(
    [in,out] SAMPR_HANDLE * SamHandle
    );

// opnum 2
long
SamrSetSecurityObject(
    [in] SAMPR_HANDLE ObjectHandle,
    [in] SECURITY_INFORMATION SecurityInformation,
    [in] PSAMPR_SR_SECURITY_DESCRIPTOR SecurityDescriptor
    );

// opnum 3
long
SamrQuerySecurityObject(
    [in] SAMPR_HANDLE ObjectHandle,
    [in] SECURITY_INFORMATION SecurityInformation,
    [out] PSAMPR_SR_SECURITY_DESCRIPTOR * SecurityDescriptor
    );

// opnum 4
```

```
void Opnum4NotUsedOnWire(void);

// opnum 5
long
SamrLookupDomainInSamServer(
    [in] SAMPR_HANDLE ServerHandle,
    [in] PRPC_UNICODE_STRING  Name,
    [out] PRPC_SID * DomainId
    );

// opnum 6
long
SamrEnumerateDomainsInSamServer(
    [in] SAMPR_HANDLE ServerHandle,
    [in,out] unsigned long * EnumerationContext,
    [out] PSAMPR_ENUMERATION_BUFFER * Buffer,
    [in] unsigned long PreferedMaximumLength,
    [out] unsigned long * CountReturned
    );

// opnum 7
long
SamrOpenDomain(
    [in] SAMPR_HANDLE ServerHandle,
    [in] unsigned long   DesiredAccess,
    [in] PRPC_SID DomainId,
    [out] SAMPR_HANDLE * DomainHandle
    );

// opnum 8
long
SamrQueryInformationDomain(
    [in] SAMPR_HANDLE DomainHandle,
    [in] DOMAIN_INFORMATION_CLASS DomainInformationClass,
    [out, switch_is(DomainInformationClass)]
        PSAMPR_DOMAIN_INFO_BUFFER * Buffer
    );

// opnum 9
long
SamrSetInformationDomain(
    [in] SAMPR_HANDLE DomainHandle,
    [in] DOMAIN_INFORMATION_CLASS DomainInformationClass,
    [in, switch_is(DomainInformationClass)]
        PSAMPR_DOMAIN_INFO_BUFFER DomainInformation
    );

// opnum 10
long
SamrCreateGroupInDomain(
    [in] SAMPR_HANDLE DomainHandle,
    [in] PRPC_UNICODE_STRING Name,
    [in] unsigned long   DesiredAccess,
    [out] SAMPR_HANDLE  * GroupHandle,
    [out] unsigned long * RelativeId
    );

// opnum 11
```

```
long
SamrEnumerateGroupsInDomain(
    [in] SAMPR_HANDLE DomainHandle,
    [in,out] unsigned long  * EnumerationContext,
    [out] PSAMPR_ENUMERATION_BUFFER * Buffer,
    [in] unsigned long    PreferedMaximumLength,
    [out] unsigned long  * CountReturned
    );

// opnum 12
long
SamrCreateUserInDomain(
    [in] SAMPR_HANDLE DomainHandle,
    [in] PRPC_UNICODE_STRING Name,
    [in] unsigned long   DesiredAccess,
    [out] SAMPR_HANDLE  * UserHandle,
    [out] unsigned long * RelativeId
    );

// opnum 13
long
SamrEnumerateUsersInDomain(
    [in] SAMPR_HANDLE DomainHandle,
    [in,out] unsigned long  * EnumerationContext,
    [in] unsigned long    UserAccountControl,
    [out] PSAMPR_ENUMERATION_BUFFER * Buffer,
    [in] unsigned long    PreferedMaximumLength,
    [out] unsigned long  * CountReturned
    );

// opnum 14
long
SamrCreateAliasInDomain(
    [in] SAMPR_HANDLE DomainHandle,
    [in] PRPC_UNICODE_STRING AccountName,
    [in] unsigned long   DesiredAccess,
    [out] SAMPR_HANDLE  * AliasHandle,
    [out] unsigned long * RelativeId
    );

// opnum 15
long
SamrEnumerateAliasesInDomain(
    [in] SAMPR_HANDLE DomainHandle,
    [in,out] unsigned long  * EnumerationContext,
    [out] PSAMPR_ENUMERATION_BUFFER * Buffer,
    [in] unsigned long    PreferedMaximumLength,
    [out] unsigned long  * CountReturned
    );

// opnum 16
long
SamrGetAliasMembership(
    [in] SAMPR_HANDLE DomainHandle,
    [in] PSAMPR_PSID_ARRAY SidArray,
    [out] PSAMPR_ULONG_ARRAY Membership
    );
```

```
// opnum 17
long
SamrLookupNamesInDomain(
    [in] SAMPR_HANDLE DomainHandle,
    [in, range(0, 1000)] unsigned long Count,
    [in, size_is(1000), length_is(Count)] RPC_UNICODE_STRING Names[*],
    [out] PSAMPR_ULONG_ARRAY RelativeIds,
    [out] PSAMPR_ULONG_ARRAY Use
    );

// opnum 18
long
SamrLookupIdsInDomain(
    [in] SAMPR_HANDLE DomainHandle,
    [in, range(0, 1000)] unsigned long    Count,
    [in, size_is(1000), length_is(Count)] unsigned long *RelativeIds,
    [out] PSAMPR_RETURNED_USTRING_ARRAY   Names,
    [out] PSAMPR_ULONG_ARRAY Use
    );

// opnum 19
long
SamrOpenGroup(
    [in] SAMPR_HANDLE DomainHandle,
    [in] unsigned long   DesiredAccess,
    [in] unsigned long   GroupId,
    [out] SAMPR_HANDLE  * GroupHandle
    );

// opnum 20
long
SamrQueryInformationGroup(
    [in] SAMPR_HANDLE GroupHandle,
    [in] GROUP_INFORMATION_CLASS GroupInformationClass,
    [out, switch_is(GroupInformationClass)]
        PSAMPR_GROUP_INFO_BUFFER * Buffer
    );

// opnum 21
long
SamrSetInformationGroup(
    [in] SAMPR_HANDLE GroupHandle,
    [in] GROUP_INFORMATION_CLASS GroupInformationClass,
    [in, switch_is(GroupInformationClass)]
        PSAMPR_GROUP_INFO_BUFFER Buffer
    );

// opnum 22
long
SamrAddMemberToGroup(
    [in] SAMPR_HANDLE GroupHandle,
    [in] unsigned long   MemberId,
    [in] unsigned long   Attributes
    );

// opnum 23
long
SamrDeleteGroup(
```

```
    [in,out] SAMPR_HANDLE  * GroupHandle
    );

// opnum 24
long
SamrRemoveMemberFromGroup(
    [in] SAMPR_HANDLE GroupHandle,
    [in] unsigned long   MemberId
    );

// opnum 25
long
SamrGetMembersInGroup(
    [in] SAMPR_HANDLE GroupHandle,
    [out] PSAMPR_GET_MEMBERS_BUFFER * Members
    );

// opnum 26
long
SamrSetMemberAttributesOfGroup(
    [in] SAMPR_HANDLE GroupHandle,
    [in] unsigned long   MemberId,
    [in] unsigned long   Attributes
    );

// opnum 27
long
SamrOpenAlias(
    [in] SAMPR_HANDLE DomainHandle,
    [in] unsigned long   DesiredAccess,
    [in] unsigned long   AliasId,
    [out] SAMPR_HANDLE  * AliasHandle
    );

// opnum 28
long
SamrQueryInformationAlias(
    [in] SAMPR_HANDLE AliasHandle,
    [in] ALIAS_INFORMATION_CLASS AliasInformationClass,
    [out, switch_is(AliasInformationClass)]
        PSAMPR_ALIAS_INFO_BUFFER * Buffer
    );

// opnum 29
long
SamrSetInformationAlias(
    [in] SAMPR_HANDLE AliasHandle,
    [in] ALIAS_INFORMATION_CLASS AliasInformationClass,
    [in, switch_is(AliasInformationClass)]
        PSAMPR_ALIAS_INFO_BUFFER Buffer
    );

// opnum 30
long
SamrDeleteAlias(
    [in, out] SAMPR_HANDLE  * AliasHandle
    );
```

```
// opnum 31
long
SamrAddMemberToAlias(
    [in] SAMPR_HANDLE AliasHandle,
    [in] PRPC_SID MemberId
    );

// opnum 32
long
SamrRemoveMemberFromAlias(
    [in] SAMPR_HANDLE AliasHandle,
    [in] PRPC_SID MemberId
    );

// opnum 33
long
SamrGetMembersInAlias(
    [in] SAMPR_HANDLE AliasHandle,
    [out] PSAMPR_PSID_ARRAY Members
    );

// opnum 34
long
SamrOpenUser(
    [in] SAMPR_HANDLE DomainHandle,
    [in] unsigned long   DesiredAccess,
    [in] unsigned long   UserId,
    [out] SAMPR_HANDLE  * UserHandle
    );

// opnum 35
long
SamrDeleteUser(
    [in,out] SAMPR_HANDLE  * UserHandle
    );

// opnum 36
long
SamrQueryInformationUser(
    [in] SAMPR_HANDLE UserHandle,
    [in] USER_INFORMATION_CLASS  UserInformationClass,
    [out, switch_is(UserInformationClass)]
        PSAMPR_USER_INFO_BUFFER * Buffer
    );

// opnum 37
long
SamrSetInformationUser(
    [in] SAMPR_HANDLE UserHandle,
    [in] USER_INFORMATION_CLASS  UserInformationClass,
    [in, switch_is(UserInformationClass)]
        PSAMPR_USER_INFO_BUFFER Buffer
    );

// opnum 38
long
SamrChangePasswordUser(
    [in] SAMPR_HANDLE UserHandle,
```

```
    [in] unsigned char    LmPresent,
    [in, unique] PENCRYPTED_LM_OWF_PASSWORD LmOldEncryptedWithLmNew,
    [in, unique] PENCRYPTED_LM_OWF_PASSWORD LmNewEncryptedWithLmOld,
    [in] unsigned char    NtPresent,
    [in, unique] PENCRYPTED_NT_OWF_PASSWORD NtOldEncryptedWithNtNew,
    [in, unique] PENCRYPTED_NT_OWF_PASSWORD NtNewEncryptedWithNtOld,
    [in] unsigned char    NtCrossEncryptionPresent,
    [in, unique] PENCRYPTED_NT_OWF_PASSWORD NtNewEncryptedWithLmNew,
    [in] unsigned char    LmCrossEncryptionPresent,
    [in, unique] PENCRYPTED_LM_OWF_PASSWORD LmNewEncryptedWithNtNew
    );

// opnum 39
long
SamrGetGroupsForUser(
    [in] SAMPR_HANDLE UserHandle,
    [out] PSAMPR_GET_GROUPS_BUFFER * Groups
    );

// opnum 40
long
SamrQueryDisplayInformation (
    [in] SAMPR_HANDLE DomainHandle,
    [in] DOMAIN_DISPLAY_INFORMATION DisplayInformationClass,
    [in] unsigned long    Index,
    [in] unsigned long    EntryCount,
    [in] unsigned long    PreferredMaximumLength,
    [out] unsigned long  * TotalAvailable,
    [out] unsigned long  * TotalReturned,
    [out, switch_is(DisplayInformationClass)]
        PSAMPR_DISPLAY_INFO_BUFFER Buffer
    );

// opnum 41
long
SamrGetDisplayEnumerationIndex (
    [in] SAMPR_HANDLE DomainHandle,
    [in] DOMAIN_DISPLAY_INFORMATION DisplayInformationClass,
    [in] PRPC_UNICODE_STRING  Prefix,
    [out] unsigned long  * Index
    );

// opnum 42
void Opnum42NotUsedOnWire(void);

// opnum 43
void Opnum43NotUsedOnWire(void);

// opnum 44
long
SamrGetUserDomainPasswordInformation (
    [in] SAMPR_HANDLE UserHandle,
    [out] PUSER_DOMAIN_PASSWORD_INFORMATION PasswordInformation
    );

// opnum 45
long
SamrRemoveMemberFromForeignDomain (
```

```
    [in] SAMPR_HANDLE DomainHandle,
    [in] PRPC_SID MemberSid
    );

// opnum 46
long
SamrQueryInformationDomain2(
    [in] SAMPR_HANDLE DomainHandle,
    [in] DOMAIN_INFORMATION_CLASS DomainInformationClass,
    [out, switch_is(DomainInformationClass)]
        PSAMPR_DOMAIN_INFO_BUFFER * Buffer
    );

// opnum 47
long
SamrQueryInformationUser2(
    [in] SAMPR_HANDLE UserHandle,
    [in] USER_INFORMATION_CLASS  UserInformationClass,
    [out, switch_is(UserInformationClass)]
        PSAMPR_USER_INFO_BUFFER * Buffer
    );

// opnum 48
long
SamrQueryDisplayInformation2 (
    [in] SAMPR_HANDLE DomainHandle,
    [in] DOMAIN_DISPLAY_INFORMATION DisplayInformationClass,
    [in] unsigned long  Index,
    [in] unsigned long  EntryCount,
    [in] unsigned long  PreferredMaximumLength,
    [out] unsigned long  *TotalAvailable,
    [out] unsigned long  *TotalReturned,
    [out, switch_is(DisplayInformationClass)]
        PSAMPR_DISPLAY_INFO_BUFFER Buffer
    );

// opnum 49
long
SamrGetDisplayEnumerationIndex2 (
    [in] SAMPR_HANDLE DomainHandle,
    [in] DOMAIN_DISPLAY_INFORMATION DisplayInformationClass,
    [in] PRPC_UNICODE_STRING Prefix,
    [out] unsigned long  * Index
    );

// opnum 50
long
SamrCreateUser2InDomain(
    [in] SAMPR_HANDLE DomainHandle,
    [in] PRPC_UNICODE_STRING Name,
    [in] unsigned long   AccountType,
    [in] unsigned long   DesiredAccess,
    [out] SAMPR_HANDLE  * UserHandle,
    [out] unsigned long * GrantedAccess,
    [out] unsigned long * RelativeId
    );

// opnum 51
```

```
long
SamrQueryDisplayInformation3 (
    [in] SAMPR_HANDLE DomainHandle,
    [in] DOMAIN_DISPLAY_INFORMATION DisplayInformationClass,
    [in] unsigned long    Index,
    [in] unsigned long    EntryCount,
    [in] unsigned long    PreferredMaximumLength,
    [out] unsigned long  * TotalAvailable,
    [out] unsigned long  * TotalReturned,
    [out, switch_is(DisplayInformationClass)]
        PSAMPR_DISPLAY_INFO_BUFFER Buffer
    );

// opnum 52
long
SamrAddMultipleMembersToAlias(
    [in] SAMPR_HANDLE AliasHandle,
    [in] PSAMPR_PSID_ARRAY MembersBuffer
    );

// opnum 53
long
SamrRemoveMultipleMembersFromAlias(
    [in] SAMPR_HANDLE AliasHandle,
    [in] PSAMPR_PSID_ARRAY MembersBuffer
    );

// opnum 54
long
SamrOemChangePasswordUser2(
    [in] handle_t BindingHandle,
    [in,unique] PRPC_STRING ServerName,
    [in] PRPC_STRING UserName,
    [in,unique]
        PSAMPR_ENCRYPTED_USER_PASSWORD NewPasswordEncryptedWithOldLm,
    [in,unique]
        PENCRYPTED_LM_OWF_PASSWORD OldLmOwfPasswordEncryptedWithNewLm
    );

// opnum 55
long
SamrUnicodeChangePasswordUser2(
    [in] handle_t BindingHandle,
    [in,unique] PRPC_UNICODE_STRING ServerName,
    [in] PRPC_UNICODE_STRING UserName,
    [in,unique]
        PSAMPR_ENCRYPTED_USER_PASSWORD NewPasswordEncryptedWithOldNt,
    [in,unique]
        PENCRYPTED_NT_OWF_PASSWORD OldNtOwfPasswordEncryptedWithNewNt,
    [in] unsigned char LmPresent,
    [in,unique]
        PSAMPR_ENCRYPTED_USER_PASSWORD NewPasswordEncryptedWithOldLm,
    [in,unique]
        PENCRYPTED_LM_OWF_PASSWORD OldLmOwfPasswordEncryptedWithNewNt
    );

// opnum 56
long
```

```
SamrGetDomainPasswordInformation (
    [in] handle_t  BindingHandle,
    [in,unique] PRPC_UNICODE_STRING Unused,
    [out] PUSER_DOMAIN_PASSWORD_INFORMATION PasswordInformation
    );

// opnum 57
long
SamrConnect2(
    [in,unique,string]  PSAMPR_SERVER_NAME ServerName,
    [out] SAMPR_HANDLE *ServerHandle,
    [in] unsigned long  DesiredAccess
    );

// opnum 58
long
SamrSetInformationUser2(
    [in] SAMPR_HANDLE UserHandle,
    [in] USER_INFORMATION_CLASS UserInformationClass,
    [in, switch_is(UserInformationClass)]
        PSAMPR_USER_INFO_BUFFER Buffer
    );

// opnum 59
void Opnum59NotUsedOnWire(void);

// opnum 60
void Opnum60NotUsedOnWire(void);

// opnum 61
void Opnum61NotUsedOnWire(void);

// opnum 62
long
SamrConnect4(
    [in,unique,string] PSAMPR_SERVER_NAME ServerName,
    [out] SAMPR_HANDLE *ServerHandle,
    [in] unsigned long ClientRevision,
    [in] unsigned long DesiredAccess
    );

// opnum 63
void Opnum63NotUsedOnWire(void);

// opnum 64
long
SamrConnect5(
    [in,unique,string] PSAMPR_SERVER_NAME ServerName,
    [in] unsigned long DesiredAccess,
    [in] unsigned long InVersion,
    [in] [switch_is(InVersion)] SAMPR_REVISION_INFO *InRevisionInfo,
    [out] unsigned long *OutVersion,
    [out] [switch_is(*OutVersion)]
        SAMPR_REVISION_INFO *OutRevisionInfo,
    [out] SAMPR_HANDLE *ServerHandle
    );

// opnum 65
```

```
long
SamrRidToSid(
    [in] SAMPR_HANDLE ObjectHandle,
    [in] unsigned long Rid,
    [out] PRPC_SID * Sid
    );

// opnum 66
long
SamrSetDSRMPassword(
    [in] handle_t BindingHandle,
    [in,unique] PRPC_UNICODE_STRING Unused,
    [in] unsigned long    UserId,
    [in,unique] PENCRYPTED_NT_OWF_PASSWORD EncryptedNtOwfPassword
    );

// opnum 67
long
SamrValidatePassword(
    [in]  handle_t Handle,
    [in]  PASSWORD_POLICY_VALIDATION_TYPE ValidationType,
    [in, switch_is(ValidationType)] PSAM_VALIDATE_INPUT_ARG InputArg,
    [out, switch_is(ValidationType)]
        PSAM_VALIDATE_OUTPUT_ARG * OutputArg
    );

// Opnum 68
void Opnum68NotUsedOnWire(void);

// Opnum 69
void Opnum69NotUsedOnWire(void);

}
```

7 Appendix B: Windows Behavior

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

- Windows NT 3.1
- Windows NT 3.5
- Windows NT 3.51
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows Vista
- Windows Server 2008

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

<1> Section 1.3.2: There is no supported configuration in which this method is called from Windows clients. See section 2.2.3.14 for details on the conditions under which this method is called from a requester.

<2> Section 1.4: The data manipulated by the responder of this protocol is used as a security principal database for authentication protocols such as NTLM [MS-NLMP] and Kerberos [MS-KILE].

<3> Section 1.6: The DC implementation of this protocol is largely for backwards compatibility with Windows NT 4.0–style applications. The LDAP protocol can be used to access a super-set of the information exposed in this protocol. The notable exception to this rule is the **SamrUnicodeChangePasswordUser2** method, which is the primary protocol that Windows clients use to change passwords and join computers to a domain (described in [MS-ADTS] section 7.4).

<4> Section 1.6: Windows clients depend on this protocol in order to perform an end-user password change and join computers to a domain (as specified in [MS-ADTS] section 7.4).

<5> Section 1.7.2: Windows clients call deprecated methods under the following conditions. There is no benefit in doing so.

Deprecated method	Condition
SamrQueryInformationDomain	Windows clients call this method for information levels less than or equal to DomainUasInformation (see section 2.2.4.16 for a description of the information levels).
SamrQueryDisplayInformation	Windows clients call this method for information levels less than or equal to DomainDisplayMachine (see section 2.2.8.12 for a description of the information levels).
SamrQueryDisplayInformation2	Windows clients call this method for information levels less than or equal to DomainDisplayGroup (see section 2.2.8.12 for a description of the information levels).

Deprecated method	Condition
SamrGetDisplayEnumerationIndex	Windows clients call this method for information levels less than or equal to DomainDisplayMachine (see section 2.2.8.12 for a description of the information levels).
SamrQueryInformationUser	Windows clients call this method in all conditions; that is, **SamrQueryInformationUser2** is not called from any Windows clients.
SamrSetInformationUser	Windows clients call this method for information levels other than UserInternal4InformationNew and UserInternal5InformationNew) (see section 2.2.7.29 for a description of the information levels).

<6> Section 1.7.3: All information levels are supported in Windows NT 4.0 and later releases, with the exception of GroupReplicationInformation for **SamrQueryInformationGroup**. This information level is supported in Windows Server 2003 and later releases.

<7> Section 1.7.4: Windows clients (that is, requesters) do not currently use the information from the responder that is contained in the Revision and SupportedFeatures fields of the **SAMPR_REVISION_INFO_V1** structure. On the responder, only the Revision field from the requester is used. It is used to determine the types of error codes that the requester understands, so that more descriptive error codes can be returned to more recent clients.

<8> Section 2.1: Windows NT, Windows 2000, and Windows Server 2003 implementations of the responder for this protocol can be configured to use the SPX (NCACN_SPX) protocol, as specified in [MS-RPCE] section 2.1.1.3; the AppleTalk (NCACN_AT_DSP) protocol, as specified in [MS-RPCE] section 2.1.1.7; and the Banyan VINES protocol. This configuration can be enabled by adding the following registry values of type REG_DWORD and by modifying the value to be nonzero:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\LSA

- For SPX: NetWareClientSupport
- For Appletalk: AppletalkClientSupport
- For Banyan VINES: VinesClientSupport

In addition, none of the Windows implementations of the requestor for this protocol can be configured to use protocols that are not listed in section 2.1.

<9> Section 2.1: By default, the endpoint "\PIPE\samr" allows anonymous access on Windows NT 3.1, Windows NT 3.5, Windows NT 3.51, Windows 2000, Windows XP, Windows Server 2003, Windows Server 2003 R2, and Windows Vista. Anonymous access to this pipe on non–domain controller machines is removed by default on Windows Vista SP1 and Windows Server 2008. The pipe access check happens before any other access check, and therefore, overrides any other access.

<10> Section 2.1: Windows uses the identity from the RPC protocol to perform access checks, as specified in the message processing section of each method.

<11> Section 2.1: A service-specific service principal name is not registered for this protocol. Windows-based clients use the host-based service principal name to identify the server for mutual authentication for the SMB and TCP RPC transports.

<12> Section 2.1: The Windows-based client uses transport security to encrypt the message for **SamrValidatePassword**.

<13> Section 2.2.3.14: There is no supported configuration in which Windows responders of this protocol (for example, a DC) return nonzero values for the **SupportedFeatures** field. However, Windows clients, starting with Windows XP and for all Windows releases thereafter, are implemented to behave as specified above. For example, after calling **SamrCreateUser2InDomain (section 3.2.5.4.4)**, Windows NT 4.0–style requester applications assume that the RID returned by **SamrCreateUser2InDomain** can be concatenated with the domain SID in which the user was created to obtain the SID of the newly-created user. This assumption limits the responder's ability to create SIDs that differ in format from this assumption, and thus limits the number of accounts ever created to 2^32 (the maximum size of an unsigned integer, which is the datatype of a RID). For more information about the extensible structure of SIDs, see [MS-SECO] section 2.3.

To allow responders (in future implementations) to generate SIDs such that the RID is not an unsigned integer (for example, a 64-bit value), the SupportedFeatures value of 1 specifies to the requester that the **SamrRidToSid** method must be called to obtain the SID of a RID value returned from this protocol. In this scenario, the RID returned from the protocol is modeled as a "handle" to the account that **SamrRidToSid** uses to return the SID value.

The two reserved values (0x00000002 and 0x00000004) have no effect on the protocol; however, when set, the Windows NET API ([MSDN-NMF]) on the requester behaves as shown in the following table. These values are mutually exclusive with each other, though they may be combined using a logical OR with other bits.

Value	Description
0x00000002	All fields that return a RID value return the value 0 instead of the RID value returned from the SAM Remote Protocol (Client-to-Server).
0x00000004	All method calls that accept information levels that return a RID fail with a Windows error code of ERROR_NOT_SUPPORTED (defined in [MS-ERREF] section 2.2).

<14> Section 2.2.4.1: Windows always returns DomainServerEnabled and ignores any input to this field.

<15> Section 2.2.4.1: The Windows server will persist (that is, store in its database) values that are not defined, and return all bits to future queries. Windows clients ignore undefined bits.

<16> Section 2.2.7.1: Windows interactive-logon applications expect this value to be either a UNC path (for example, \\machine-name\share-name\directory-name) or a Windows file path without the drive letter (for example, \program files\billg) or a zero-length string.

<17> Section 2.2.7.1: Windows interactive-logon applications expect this value to be either a zero-length string or a string with two characters: an alphabetic character, 'a' through 'z', in lower- or uppercase, followed by a colon (':').

<18> Section 2.2.7.1: This value is not accurate in multiple-DC configurations, as this value is not replicated among DCs. Therefore, this field MUST NOT be used by requesters. Windows clients do not use this field.

<19> Section 2.2.7.1: This value is not accurate in multiple-DC configurations, as this value is not replicated among DCs. Windows clients do not use this field.

<20> Section 2.2.7.1: This value is not accurate in multiple-DC configurations, as this value is not replicated among DCs. Therefore, this field MUST NOT be used by requesters. Windows clients do not use this field.

<21> Section 2.2.7.21: Windows clients fill the unused portions of **Buffer** with random bytes.

<22> Section 2.2.7.22: Windows clients set the unused portions of **Buffer** to cryptographically random bytes.

<23> Section 2.2.7.23: This structure is used only for the **SamrUnicodeChangePasswordUser3** method. There are no Windows clients that call this method.

<24> Section 2.2.10.1: Windows sets this buffer to the repeating pattern 0x20 0x00 on update.

<25> Section 2.2.10.2: Windows sets this value to 1 or 2, but does not use the value.

<26> Section 2.2.10.3: Windows sets this value to 0x31 and ignores it on read.

<27> Section 2.2.10.4: Windows ignores this value on read; see section 3.2.1.8.11.4 for behavior on update.

<28> Section 2.2.10.4: Windows ignores this value on read; see section 3.2.1.8.11.4 for behavior on update.

<29> Section 2.2.10.4: Windows ignores this value on read; see section 3.2.1.8.11.4 for behavior on update.

<30> Section 3.2.1.2: On a non-DC configuration, Windows transforms the characters to uppercase, per [UNICODE3.1], and then byte-compares the values.

<31> Section 3.2.1.8.3: On a DC configuration, Windows initiates urgent replication (described in [MS-ADTS] section 3.1.1.1.14, under event-driven replication) when this attribute value changes.

<32> Section 3.2.1.8.8: On a DC configuration, Windows initiates urgent replication (described in [MS-ADTS] section 3.1.1.1.14, under event-driven replication) when this attribute value is set to 0 or this attribute value changes due to a password change request (as opposed to set), and **userAccountControl** contains the UF_NORMAL_ACCOUNT flag.

<33> Section 3.2.1.8.10: On a DC configuration, if the UF_SERVER_TRUST_ACCOUNT bit or the UF_WORKSTATION_TRUST_ACCOUNT bit changes on commit, an urgent replication is initiated. (Information about urgent replication is specified in [MS-ADTS] section 3.1.1.1.14.)

<34> Section 3.2.1.8.11.4: Windows uses the account's userPrincipalName as the DefaultSalt value. However, it does not use this value in any calculation.

<35> Section 3.2.1.9.2.1: If the constraints in step 1 cannot be satisfied, the responder returns an error code to the requester and initiates an asynchronous request to the domain for a **RIDAllocationPool**, if such an asynchronous call is not already active.

<36> Section 3.2.2: In Windows 2000 SP4, Windows Server 2003 SP1, and Windows XP SP2, the Windows implementation of RPC does not satisfy this requirement. Consequently, a security check is enforced by the responder of this protocol to ensure this constraint. Specifically, the responder ensures that the SID of the requester matches the SID of the requester that opened the handle. If this condition is not met, a processing error is returned to the requester.

<37> Section 3.2.4.2: The initial membership of this group depends on the version of Windows running on the first DC of the domain and on the administrator's choice between "pre-Windows 2000–compatible permissions mode" and "Windows 2000–only permissions mode". In Windows 2000 Server, in the pre-Windows 2000–compatible permissions mode, "Everyone" (S-1-1-0) is a member, and in the Windows 2000–only permissions mode, the membership is empty. In Windows Server 2003, in the pre-Windows 2000–compatible permissions mode, "Everyone" (S-1-1-0) and "Anonymous" (S-1-5-7) are members, and in the Windows 2000–only permissions mode, "Authenticated Users" (S-1-5-11) are members.

<38> Section 3.2.5: Opnums reserved for local use apply to Windows as follows.

Opnum	Description
4	Not used by Windows
42	Just returns STATUS_NOT_IMPLEMENTED. It is never used.
43	Just returns STATUS_NOT_IMPLEMENTED. It is never used.
59	Used only locally by Windows, never remotely.
60	Used only locally by Windows, never remotely.
61	Not used by Windows.
63	Not used by Windows.
68	Used only locally by Windows, never remotely.
69	Used only locally by Windows, never remotely.

<39> Section 3.2.5.1.1: *ServerName* is ignored on receipt.

<40> Section 3.2.5.1.1: Windows sets *OutVersion* to 1, OutRevisionInfo.Revision to 3, and the remaining fields to zero.

<41> Section 3.2.5.1.2: *ServerName* is ignored on receipt.

<42> Section 3.2.5.1.3: *ServerName* is ignored on receipt.

<43> Section 3.2.5.1.4: *ServerName* is ignored on receipt.

<44> Section 3.2.5.2.1: Windows uses the *EnumerationContext* to restart the enumeration on subsequent calls.

<45> Section 3.2.5.2.1: Windows does NOT validate the input, though the result of malformed information merely results in inconsistent output to the requester.

<46> Section 3.2.5.2.1: Windows estimates the number of entries to return by dividing *PreferedMaximumLength* by the number of bytes of a maximum-sized entry.

<47> Section 3.2.5.2.2: Windows uses the *EnumerationContext* to restart the enumeration on subsequent calls.

<48> Section 3.2.5.2.2: Windows does not validate the input, though the result of malformed information merely results in inconsistent output to the requester.

<49> Section 3.2.5.2.2: Windows estimates the number of entries to return by dividing *PreferedMaximumLength* by the number of bytes of a maximum-sized entry.

<50> Section 3.2.5.3.1: This value is estimated and is not accurate. Windows clients do not rely on the accuracy of this value.

<51> Section 3.2.5.3.1: On a non-DC configuration, *Index* is a per-element monotonically-increasing number. If *Index* (the message parameter) is 0, the start value is 0; otherwise the start value is one greater than *Index* (the message parameter).

On a DC, this value is an implementation-specific value that satisfies the requirement shown above.

<52> Section 3.2.5.5.1.1: On non-DC configurations, the exact value is returned. On DC configurations, Windows estimates this count.

<53> Section 3.2.5.5.1.1: On non-DC configurations, the exact value is returned. On DC configurations, Windows estimates this count.

<54> Section 3.2.5.5.1.1: On non-DC configurations, the exact value is returned. On DC configurations, Windows estimates this count.

<55> Section 3.2.5.10.2: Windows servers ignore the *ServerName* parameter.

<56> Section 3.2.5.10.3: Windows servers ignore the *ServerName* parameter.

<57> Section 3.2.5.10.4: Clients connecting to servers that are running Windows 2000 Server or Windows NT Server must use **SamrUnicodeChangePasswordUser2**. Clients connecting to servers that are running Windows Server 2003 and later can also use **SamrUnicodeChangePasswordUser3**.

<58> Section 3.2.5.10.4: There are no Windows clients that call this method.

<59> Section 3.2.5.10.4: Windows servers ignore the *ServerName* parameter.

<60> Section 3.2.5.12.1.1: If USER_CHANGE_PASSWORD is not granted to World on receipt, Windows adds the following (deny) ACEs to the **ntSecurityDescriptor** value:

Field name	Value
Ace Type	ACCESS_DENIED_OBJECT_ACE_TYPE
SID	PRINCIPAL_SELF_SID
Access Mask	ACTRL_DS_CONTROL_ACCESS
ObjectGuid	ab721a53-1e2f-11d0-9819-00aa0040529b

Field name	Value
Ace Type	ACCESS_DENIED_OBJECT_ACE_TYPE
SID	World
Access Mask	ACTRL_DS_CONTROL_ACCESS
ObjectGuid	ab721a53-1e2f-11d0-9819-00aa0040529b

If USER_CHANGE_PASSWORD is granted to Self or World on receipt, Windows removes the above two ACEs (if present) and adds the following two ACEs, if not already present:

Field name	Value
Ace Type	ACCESS_ALLOWED_OBJECT_ACE_TYPE
SID	Self

Field name	Value
Access Mask	ACTRL_DS_CONTROL_ACCESS
ObjectGuid	ab721a53-1e2f-11d0-9819-00aa0040529b

Field name	Value
Ace Type	ACCESS_ALLOWED_OBJECT_ACE_TYPE
SID	World
Access Mask	ACTRL_DS_CONTROL_ACCESS
ObjectGuid	ab721a53-1e2f-11d0-9819-00aa0040529b

<61> Section 3.2.5.13.2: Starting with Windows Vista and Windows Server 2008, **SamrShutdownSamServer** has been deprecated and always returns STATUS_NOT_SUPPORTED.

<62> Section 3.2.5.13.2: Windows Vista returns STATUS_NOT_SUPPORTED.

Note STATUS_NOT_SUPPORTED has a value of 0xC00000BB.

<63> Section 3.2.5.13.5: Windows clients set this value to be the NULL-terminated NETBIOS name of the responder.

<64> Section 3.2.5.13.7: Windows Server enforces that the *UserId* parameter is 0x1F4.

<65> Section 3.2.5.13.7: Windows does not decrypt the value, but stores the encrypted value directly in an implementation-specific store.

<66> Section 3.2.5.14.1: Windows uses the sAMAccountName attribute unless the sAMAccountName attribute contains characters that are not allowed for an **RDN** (RDN syntax is specified in [MS-ADTS] section 3.1.1.1.4), in which case the objectSid is used (in string form). If the sAMAccountName is not a unique RDN for the given container, the responder returns STATUS_USER_EXISTS to the requester.

<67> Section 3.2.5.14.7: Windows clients do not set this field.

8 Index

Q

R

S

T

U

V

W