07 Fakultät für Informations-, Medien- und Elektrotechnik,
Studiengang: Information Engineering

Institut für Nachrichtentechnik
Informatics laboratory

Bachelor thesis

Thema: Active Directory Replication

Untertitel: Documentation and evaluation of the DRSUAPI replication protocol, as well as a partial implementation of the replication from Windows 2003 to Samba 4.0.

Student: Stefan Metzmacher

Matrikelnummer: 11027678

E-Mail: metze@samba.org

Referent: Prof. Dr. phil. Gregor Büchel

Korreferent: Prof. Dr.-Ing. Andreas Grebe

Abgabedatum: 18.03.2007

This is still a draft of the translated version!

Table of Contents

List of Figures

Chapter 1. Introduction

In the current IT-world, directory services play a central part. They provide a central storage location for user accounts including group memberships as well as software configurations of any type. Furthermore, directory services provide user authentication as well as configurable guidelines that allow access to objects, which are saved within the directory and their attributes.

From the point of view of the person accountable for the IT department within a company, it is desirable to be able to choose freely between alternative products in the market of directory services. The important aspect here is that all the information that was stored in the directory can be reexported in order to be able to switch to a different product. Along the lines of: “My data belongs to me and not to the software, that manages it!”

From the product manufacturers point of view, it is desirable to bind the client to their respective own product. With all products, almost all the data can be accessed through the *Lightweight Directory Access Protocol* (LDAP) [RFC4510], a standardised networking protocol for the access to directory services. But an access to user passwords is not possible with most proprietary products, even through encrypted data connections with administrator privileges. But since in most environments, out of redundancy considerations, several copies of directory data are being maintained on different servers, non-standardised networking protocols are usually used for the alignment of data stocks including user passwords. This is why a customer is bound to a specific product, since it is in the most cases practically impossible to issue a new password to every user.

The market leader Microsoft uses in its directory service “Active Directory” also a proprietary replication protocol called *Directory Replication Service Update API* (DRSUAPI), which has been evaluated within this bachelor thesis.

1.1. Presentation of the problem

1.1.1. What is Active-Directory?

Microsofts directory service is called “Active Directory” and is part of the products “Windows 2000 Server” and “ Windows 2003 Server ”. Active-Directory plays the part of the so called *Domain Controller* (DC) within *Windows Domains*, which is responsible for user management and user authentication. Sometimes, Windows 2000 and Windows 2003 are abbreviated W2K and W2K3 respectively

Windows NT4 Domains use exclusively proprietary network protocols for user management, as for instance SAMR and LSA. NTLMSSP as well as NETLOGON/SCHANNEL have been used as authentication protocol. Furthermore, there was no directory service, but a *Security Account Manager* (SAM), which stored users and groups in the “Windows Registry”. Under Windows NT4 it had been still differentiated between *Primary Domain Controller* (PDC) and *Backup Domain Controller* (BDC), although write access was only given with PDC.

Active Directory integrates a set of modern networking and authentication protocols, such as LDAP, DNS, Kerberos 5[1] and DIGEST-MD5. But additionally the “old” protocols of NT4 are being used. The *NT Directory Service* (NTDS) now constitutes the data basis for all supported protocols and constitutes the main component of the Active Directory.

Typically, authentication protocols are used by Active Directory for the authentication of file- and printer access by means of the SMB protocol[2], but also with e-Mail or web services.

[1] Also known as Kerberos-V or Krb5

1.1.2. What is Samba?

Samba is a software package that provides file- and printer services compatible with Microsoft products under Unix/Linux operating systems. Samba is “free software” and available under the “GNU General Public License” (see [SaTe04], [FSF01], [FSF02] and [WikiPe02]).

Samba is being developed since 1992 by Andrew Tridgell and since 1996 by the Samba-Team. Back then, Tridgell wanted to access the hard disk of his “Sun Sparcstation” from his PC via “Pathworks for DOS” and for this he wrote a server program, without knowing that the protocol he used was “Server Message Block” (SMB). At this time, Microsoft still had a very little share of the market, but this changed after the introduction of Windows NT3.5 and later on Windows NT4. Microsoft build its proprietary authentication- and user management protcols on the basis of protocols SMB and DCERPC. In the process, SMB and DCERPC, among other things, were provided with undocumented extensions. The Samba-Team pursues the goal to be as compatible to Microsoft products as possible. For this, several methods for network analysis were developed (see [Tridge01] and [Tridge02]). Today, Samba is in version 3.0 largely compatible with Microsoft products as file-, print- and registration server.

Since 2004, some members of the Samba-Team, including Andrew Tridgell, Andrew Bartlett, Simo Sorce, Jelmer Vernooij and the author of this bachelor thesis, work on a complete overhaul of the ageing Samba source code. Goal of this restructuring is an Active Directory compatible implementation of a domain controller that supports all the relevant protocols. The new version is called Samba 4.0, although currently only “Technology Previews”, which are merely developer versions, have been released. An “alpha status” has not yet been reached.

1.1.3. Why is the subject of importance?

Samba is most frequently used in combination with the “Free Operating System” Linux (see [FSF01], [FSF02] and [WikiPe02]). More and more persons responsible for the IT department value the fact that everyone is able to view and extend the source code of Linux and Samba. In proprietary software, backdoors and spyware are often assumed, which makes “Free Software”, like Linux and Samba especially interesting for governments, among others, the Deutsche Bundestag uses Samba in combination with Linux. In many cases, the missing licensing costs are also a big factor in the decision for Samba. Furthermore, Linux and Samba can be compared very well to Windows servers in areas of stability, performance and security.

The Samba-Team receives every now and then inquiries by its users, if or from which point on, Samba could replace a domain controller for Active-Directory. Many would love to switch from Windows servers to Samba servers, but strongly need some special characteristics of Active Directory.

1.1.4. Samba should be a domain controller for Active Directory.

In order to convert Samba 4.0 into a domain controller compatible with Active Directory, among other things a LDAP-server and a Kerberos-server[3] have been built into the server program `smbd`.

Currently, Samba 4.0 can be used as Active Directory domain controller opposite to Windows domain members [4] Specifically, this means that a computer that is Windows member

[2]Also called CIF protocol
[3]A Kerberos *Key Distribution Center* (KDC)
[4]This is how workstations and servers are called, which usually don't possess local user accounts and thus have to use authentication services provided by domain controllers.

readily joins a domain controlled by Samba 4.0, believing that it is actually a domain controlled by Windows 2003 servers. Subsequently, one can log in with an user account of the domain to the member computer with Kerberos authentication.

In order to be managing a domain equivalently among Windows servers, Samba 4.0 is still missing some features. Currently, access protection for directory objects is managed very trivially, that means an administrator may read and write all data, and everybody else may read everything but the passwords. Furthermore, written data is not examined for syntactic correctness, that means there is no so-called directory schema, which defines the architecture of objects and their hierarchy as well as the syntax of object attributes. However, the most important missing function is the synchronisation of directory data with “Windows colleagues”, as to maintain an always current replica of the directory. The process of synchronisation is usually called “replication” and is the main subject of this bachelor thesis.

1.1.5. What are the difficulties

By this time, good documentation on the technologies and concepts used within Active Directory do exist. Those are given by Microsoft themselves through the internet portals “Microsoft TechNet” and “Microsoft MSDN” (see [MSTN01], [MSTN02], [MSTN03], [MSTN04], [MSTN05], [MSTN06] and [MSDN01]). However, those are directed exclusively at administrators and thus do not contain documentation over the used networking protocols. Accordingly, the main difficulty of the implementation of the DRSUAPI replication protocol lies in the fact that there is no published specification, as is for example the case with LDAP or Kerberos 5.

Because of the missing specification, the use of methods, described by Tridgell, for the development of a networking protocol specification is required (see [Tridge01] and [Tridge02]).

1.1.6. Why partial implementation

A complete implementation of DRSUAPI server and DRSUAPI client as well as all administration tools would easily exceed the scope of this bachelor thesis. Thus, in this bachelor thesis only the points given in the chapter “objectives” will be dealt with.

1.2. Preliminary work

1.2.1. Student research paper “Active Directory Replication”

Diese Bachelorarbeit baut auf der Studienarbeit “Active Directory Replikation” im Fach Datennetze, von Michael Kohlgraf sowie dem Autor dieser Bachelorarbeit, auf (siehe [KoMe2005]). In the student research paper, the layers 5 (session layer) and 6 (depiction layer) of the OSI-Model[5] Here, particularly the NDR-Coding[6] and the interface definition with help from IDL[7] were considered. However, the function `DsGetNCChanges()`, that is used for replication, had not been explored within the paper.

1.2.2. Development within the scope of the Samba-Project

In the scope of the development of Samba 4.0, I explored further the DRSUAPI-Protocol. Here, also the IDL descriptions for the functions `DsGetNCChanges()`, `DsAddEntry()` and `DsReplicaUpdateRefs()`, necessary for replication, have been elaborated. Samba 4.0 has a test program `smbtorture`, which can be used to test networking protocols and internal interfaces. Here, a new test, called “RPC-DSSYNC”, has been added, which demonstrated the functionality of `DsGetNCChanges()`. Because of this, it could be reconstructed, how directory objects along with their attributes are transmitted. However, some details were not clear, like for example the connection specific encryption of several password attributes.

Samba 4.0 provides all the technologies that are necessary for replication, such as Kerberos-5, LDAP and DCERPC. Because of this, the basis for this bachelor thesis is Samba 4.0 whereby the developments are returned into the Samba sourcecode.

Within this bachelor thesis, only the developer version Samba 4.0 is taken into account. This is why, in the following chapters the explicit version declaration 4.0 is neglected and with “Samba”, Samba 4.0 is referred to implicitly.

[5] Also called 7-Layer-Model (see [WikiPe01])
[6] “Network Data Representation” (see [OpGr02])
[7] “Interface Definition Language” (see [OpGr03])

Chapter 2. Objectives

In this chapter, the objectives of this bachelor thesis are defined.

2.1. Implementation of the function `libnet_BecomeDC()`

Within the scope of this bachelor thesis, the function `libnet_BecomeDC()` shall be implemented, which switches the Server-Role[1] of a Samba-Server in an Active-Directory-Domain in Mixed-Mode controlled by a Windows-2003-Server[2] from Domain-Member to Domain-Controller changes.

For this, Sambas Management-Api *LIBNET* is extended by the function `libnet_BecomeDC()`. In order to achieve completeness, the function `libnet_UnbecomeDC()` will be also implemented, which switches the Server-Role[1] back to Domain-Member. However, because of size reasons, only the function `libnet_BecomeDC()` will be explained in detail.

2.2. Implementation of a Schema-API

Within Sambas DSDB-Component[3] a Schema-API including Schema-Cache shall be developed, in order to give access to the Directory-Schema[4] The Schema-Cache will be realised through the C-Structures `struct dsdb_schema`, `struct dsdb_class` and `struct dsdb_attribute`. Whereby `struct dsdb_schema` is a container for `struct dsdb_class`, or rather `struct dsdb_attribute` instances. The Schema-API consequently gives C-Functions for access to the Schema-Cache. For a successful replication, a Schema-Cache with Schema-API is strongly required, since otherwise some parts of the replication messages cannot be interpreted. Within the scope of this bachelor thesis, the Schema-API will be implemented only for this reason. Use of the Schema-API for the validation of write access to directory objects will not be taken into account.

Furthermore, a LDB-Modul[5] with the name “schema_fsmo” will be developed. This module shall at a later point - among other things - validate write accesses to the directory schema[4], since the write accesses within Active-Directory are only possible to a selected server, the so-called “Schema-Master”[6]. However, within the scope of this bachelor thesis, only the loading

[1] The following server roles are defined: “Single”, “Domain-Member” as well as “Domain-Controller”

[2] Active-Directory-Domains can be operated in two modes. In “Mixed-Mode”, Windows-NT4 *Backup-Domain-Controller* are supported, which is why some functions of Active-Directory, such as for example nested group memberships, are deactivated. In “Native-Mode”, only Active-Directory-Domain-Controller are supported. The mode is set upon installation of Active-Directory and can later only be changed from Mixed-Mode into Native-Mode, but not the other way around.

[3] DSDB stands for “Directory Service Database”

[4] The Directory-Schema contains attribute- and class definitions, which regulate the structure and hierarchy of objects saved within the directory.

[5] LDB (Lightweight Database) is Sambas LDAP-type interface for the storage of LDAP-type objects. LDB is strongly modularized and can be extended by LDB-Modules, in order to extend the functions regarding the public LDB-API.

[6] In active-Directory, the following 5 “Flexible Service Operation Master”-Roles are defined: “Schema-Master”, “Domain-Naming-Master”, “RID-master”, “PDC-Emulator-Master” and “Infrastructure-Master” (See also [MSTN03])

of the Schema-Cache for use within the Schema-API will be developed.

2.3. Implementation of a LDB-Module for the storage of replication meta data

In order to store the meta data required for replication by Samba, a LDB-Module[5] with the name “repl_meta_data”will be developed. For direct write accesses to directory objects, this module shall create the for the replication required meta data and save attributes of the respective object in “replPropertyMetaData”. This task will be implemented within the scope of this bachelor thesis only for the `ldb_add()` function in an exemplary manner.

Within the DSDB-Component[3] also, the function `dsdb_extended_replicated_objects_commit()` will be implemented, which applies the changes to the local directory data that are contained in a replication message. Here, the changes are primarily conditioned using the Schema-API and then, using a so-called *LDB-Extended-Operation*[7] named `DSDB_EXTENDED_REPLICATED_OBJECTS_OID`, given to the LDB-Layer through the function `ldb_extended()`. The LDB-Module “repl_meta_data” accepts this request and applies the changes.

2.4. Requirements and limits

Within the course of the processing time, Samba is daily advanced and new understandings are made on a weekly basis. That is why some discoveries can only partially be explained in this bachelor thesis, it would take too long otherwise

Since the subject of this bachelor thesis is very specific, basic knowledge in the areas of informatics, databases, data networks, operating systems, distribution systems as well as knowledge in the area of LDAP and directory services have to be expected.

Furthermore, the present Samba C-Functions can not be explained, for this, the complete Samba 4.0 source codes are available under [SaTe01] (see also [SaTe02]).

Within the scope of this bachelor thesis, the treatment of so-called *Linked-Attributes*[8] for the implementation is neglected entirely.

[7] With aid of *LDB-Extended-Operations* one can add new functions to the LDAP-Protocol, or the LDB-API.

[8] *Linked-Attributes* are attributes, that save links to other directory objects. These attributes always present themselves in pairs, whereby there exist a *Forward-Link* and a *Backward-Link*, that each link to the other object. Only the *Forward-Link* can be changed, the *Backward-Link* is always automatically matched. In the replication, only the *Forward-Link* is transmitted, whereby the *Backward-Link* has to be created automatically. In an Active-Directory-Domain in Native-Mode, *Linked-Attributes* will be replicated separately of the objects.

Chapter 3. Description of the replication system

In the following chapter, the fundamental replication model is introduced. Furthermore, the applied technologies will be briefly explained.

The descriptions refer to the Microsoft Technet Document “How the Active Directory Model Works” [MSTN01]. Additionally, test programs created by the Samba-Team deliver proofed facts that also enter into the descriptions, since there are no specifications to be found in [MSTN01].

3.1. What is a directory service

3.1.1. Overview

A directory service stores directory-objects in a tree-structure, which is called *Directory Information Tree* (DIT). Figure 3.1, “A typical Directory-Tree with directory objects.” displays a DIT, within a LDAP-Client with graphical interface. Each object has a *Distinguished Name* (DN), for instance “OU=newtop, DC=sub1, DC=w2k3, DC=vmnet1, DC=vm, DC=base”, whereby the individual components of the Dn are called *Relative Distinguished Name* (RDN). In the example, “OU=newtop” is the RDN of the object and “DC=sub1,DC=w2k3,DC=vmnet1,DC=vm,DC=base” the parent-object. “OU=newtop, DC=sub1, DC=w2k3, DC=vmnet1, DC=vm, DC=base” is thus a child-object of “DC=sub1, DC=w2k3, DC=vmnet1, DC=vm, DC=base”. Generally the namespace of the directory is global, related to the internet. Each *Directory Service Agent* (DSA)[1], however, only stores a part of the entire namespace. But each DSA provides the root-object, which contains an empty DN and is called *Root DSA-specific Entry* (RootDSE).

[1]The server instance of the directory service is called “Directory Service Agent” in english.

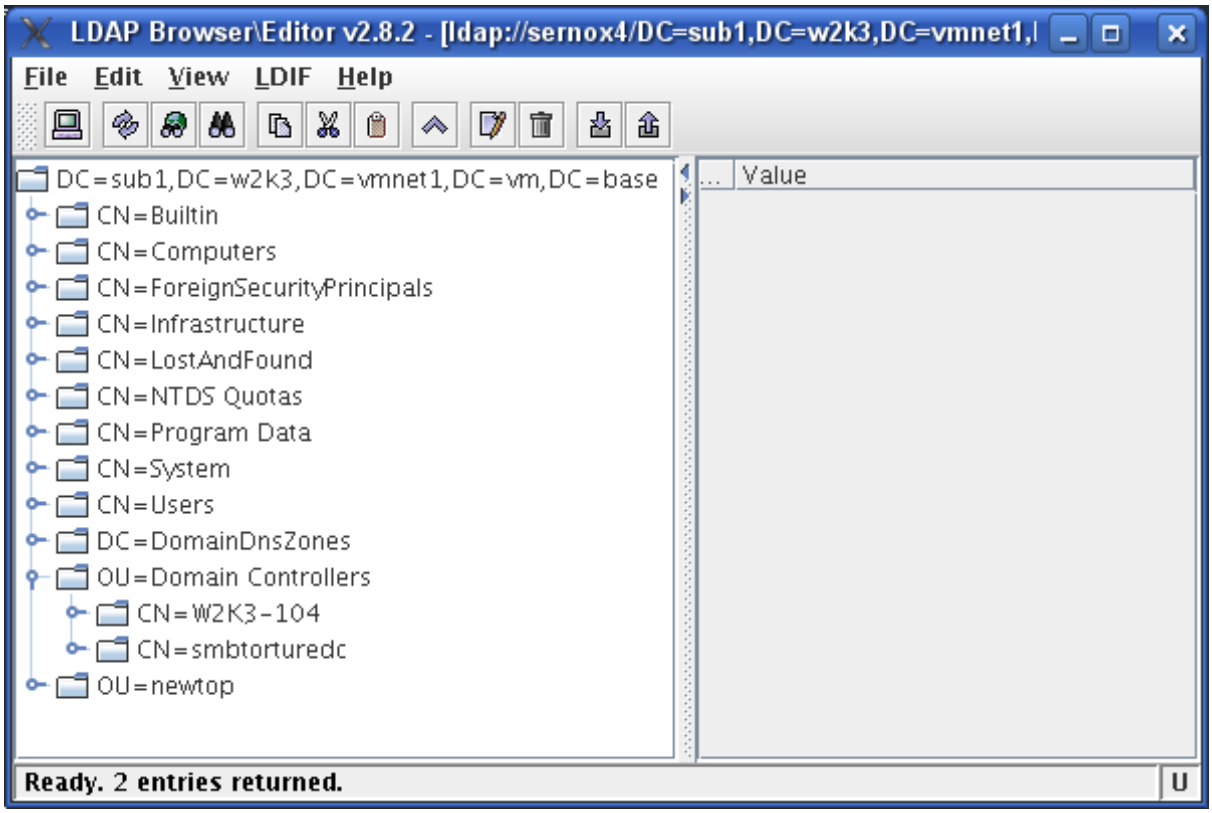

Figure 3.1. A typical Directory-Tree with directory objects.

3.1.2. RootDSE

Figure 3.2, “A typical RootDSE object (a pair of attribute values has been removed in order to give a better overview).” shows the RootDSE-object of a Windows-2003-server in *LDAP Data Interchange Format* (LDIF) [RFC2849][2]. With the attribute “namingContexts” the DSA indicates, which parts of the global namespace it provides. In addition, Windows-2003-Server provide the attributes “defaultNamingContext”, “configurationNamingContext”, “schemaNamingContext” and “rootDomainNamingContext” (compare Figure 3.2, “A typical RootDSE object (a pair of attribute values has been removed in order to give a better overview).”). Each domain-controller stores at least three writable *Directory Partitions*[3] The *domain partition*, the *configuration partition* and the *schema partition* Figure 3.3, “A typical domain forest within Windows 2003.” shows a so-called domain forest, in which several domain trees can be logically consolidated. Here, only one configuration partition exists for a forest and only one schema partition. The attribute “rootDomainNamingContext” always contains the DN of the root domain within the domain tree, whereas the attribute “defaultNamingContext” always contains the DN of the local domain, for that the DSA is responsible. That means that each DSA can only manage one domain. The DNs of the domains are based on their DNS names. In the example “sub1.w2k3.vmnet1.vm.base” is the DNS do-

[2]The first row always contains the DN of the object with a prefixed “dn: ”. Objects always consist of their DN and a set of object-attributes. All following rows until the next blank line are pairs of attribute-name and attribute-value. If a row starts with a blank space, the following characters still belong to the attribute-value of the previous row. For attribute-values, that can be displayed as ASCII-string, attribute name and attribute-value are separated by “: ”. If this is not possible, the attribute-value is coded using *Base64* [RFC4648] and “:: ” is used for the separation of attribute-name and attribute-value.

[3]“Directory Partition” is a synonym for “naming context” (NC)). The term directory partition is used within this bachelor thesis.

main name. Furthermore exist *application partitions* and non writable partition copies. Those two special cases are not further explained in this bachelor thesis.

```
dn:
currentTime: 20070304212854.0Z
subschemaSubentry: CN=Aggregate,CN=Schema,CN=Configuration,DC=w2k3,DC=vmnet1,D
 C=vm,DC=base
dsServiceName: CN=NTDS Settings,CN=W2K3-104,CN=Servers,CN=Standardname-des-ers
 ten-Standorts,CN=Sites,CN=Configuration,DC=w2k3,DC=vmnet1,DC=vm,DC=base
namingContexts: CN=Configuration,DC=w2k3,DC=vmnet1,DC=vm,DC=base
namingContexts: CN=Schema,CN=Configuration,DC=w2k3,DC=vmnet1,DC=vm,DC=base
namingContexts: DC=sub1,DC=w2k3,DC=vmnet1,DC=vm,DC=base
defaultNamingContext: DC=sub1,DC=w2k3,DC=vmnet1,DC=vm,DC=base
schemaNamingContext: CN=Schema,CN=Configuration,DC=w2k3,DC=vmnet1,DC=vm,DC=bas
 e
configurationNamingContext: CN=Configuration,DC=w2k3,DC=vmnet1,DC=vm,DC=base
rootDomainNamingContext: DC=w2k3,DC=vmnet1,DC=vm,DC=base
supportedControl: 1.2.840.113556.1.4.319
supportedControl: 1.2.840.113556.1.4.801
supportedControl: 1.2.840.113556.1.4.473
supportedControl: 1.2.840.113556.1.4.528
supportedControl: 1.2.840.113556.1.4.417
supportedLDAPVersion: 3
supportedLDAPVersion: 2
supportedLDAPPolicies: MaxPoolThreads
supportedLDAPPolicies: MaxDatagramRecv
supportedLDAPPolicies: MaxReceiveBuffer
supportedLDAPPolicies: InitRecvTimeout
highestCommittedUSN: 147481
supportedSASLMechanisms: GSSAPI
supportedSASLMechanisms: GSS-SPNEGO
supportedSASLMechanisms: EXTERNAL
supportedSASLMechanisms: DIGEST-MD5
dnsHostName: w2k3-104.sub1.w2k3.vmnet1.vm.base
ldapServiceName: w2k3.vmnet1.vm.base:w2k3-104$@SUB1.W2K3.VMNET1.VM.BASE
serverName: CN=W2K3-104,CN=Servers,CN=Standardname-des-ersten-Standorts,CN=Sit
 es,CN=Configuration,DC=w2k3,DC=vmnet1,DC=vm,DC=base
supportedCapabilities: 1.2.840.113556.1.4.800
supportedCapabilities: 1.2.840.113556.1.4.1670
supportedCapabilities: 1.2.840.113556.1.4.1791
isSynchronized: TRUE
isGlobalCatalogReady: FALSE
domainFunctionality: 0
forestFunctionality: 0
domainControllerFunctionality: 2
```

Figure 3.2. A typical RootDSE object (a pair of attribute values has been removed in order to give a better overview).

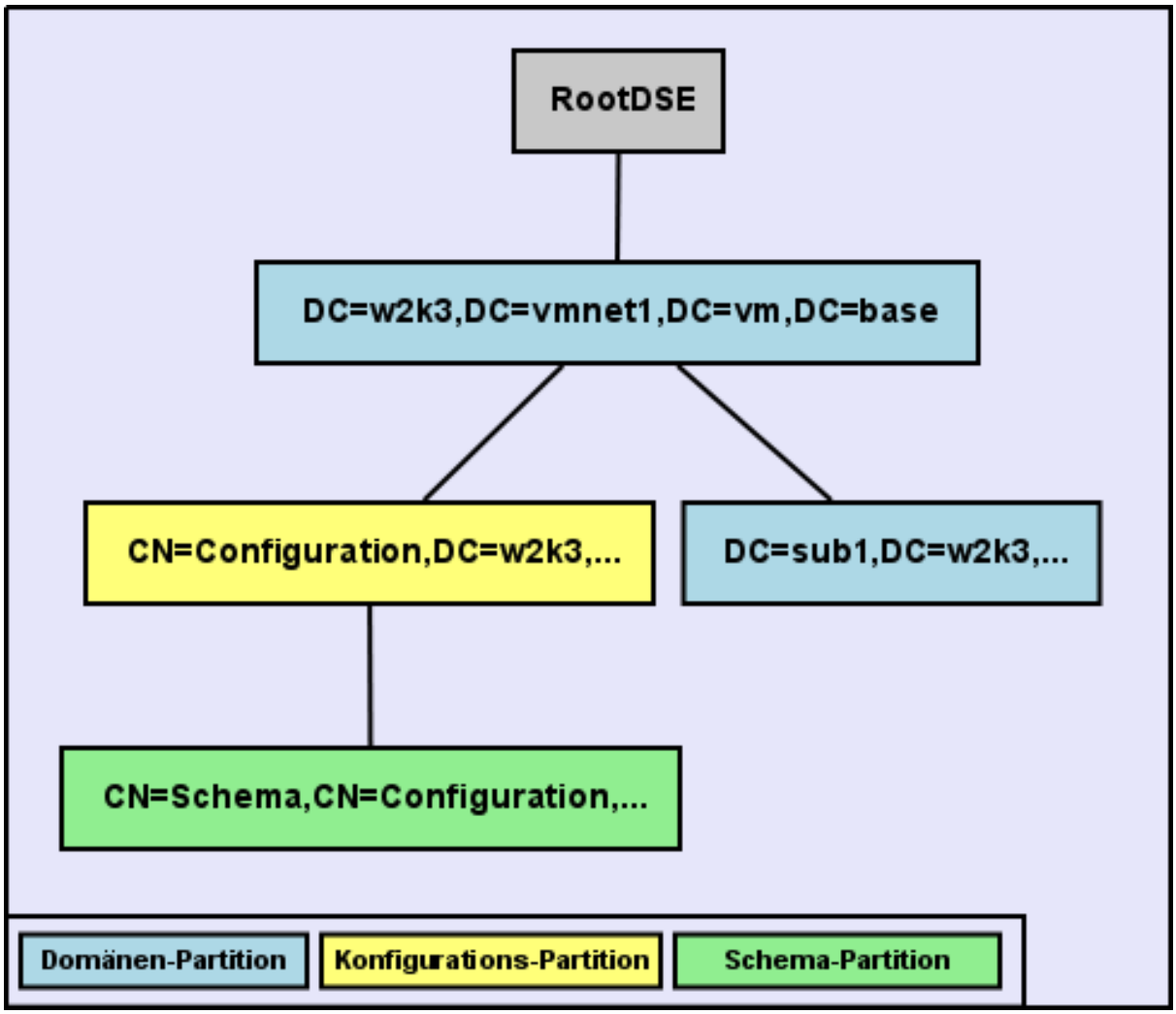

Figure 3.3. A typical domain forest within Windows 2003.

3.1.3. Directory objects

Figure 3.4, “A typical directory object with attributes.” gives an example of a directory object. The attribute “objectClass” plays a special part, since objects are instanced object classes. The directory schema contains the definitions of attributes and object classes, which in turn are instances of the object classes “attributeSchema” or “classSchema”. Object classes define, which attributes are mandatory for an object and which ones are optional. Furthermore, object classes can inherit properties of other classes, for instance each class is derived by the superclass “top”. For attributes, it is among other things defined, if several attribute values are possible for an object or which syntax requirements must be complied with by them. He who wants to know more about inheritance and other details of the directory schema used within Active Directory, can find a good description under [MSTN04] (also see source/setup/schema.ldif under [SaTe01]).

```
dn: OU=newtop,DC=sub1,DC=w2k3,DC=vmnet1,DC=vm,DC=base
objectClass: top
objectClass: organizationalUnit
ou: newtop
distinguishedName: OU=newtop,DC=sub1,DC=w2k3,DC=vmnet1,DC=vm,DC=base
instanceType: 4
whenCreated: 20070115100855.0Z
whenChanged: 20070115103112.0Z
uSNCreated: 131547
uSNChanged: 131648
name: newtop
objectGUID:: 8VdxB8zuv0e6A5dTyT4ucw==
objectCategory: CN=Organizational-Unit,CN=Schema,CN=Configuration,DC=w2k3,DC=v
 mnet1,DC=vm,DC=base
```

Figure 3.4. A typical directory object with attributes.

3.1.4. The partitions

The schema partition

Within the schema partition, all the information on the directory schema is stored (also see [MSTN05]).

The configuration partition

In the configuration partition, all the configuration information, that concern the entire domain forest, is stored (see also [MSTN05]). For example, information on all domain controllers and their sites are part of it. Furthermore, information on all directory partitions stored within the domain forest are stored in it.

The domain partition

Within the domain partition, all the data that is relevant to the corresponding domain is stored. (see also [MSTN05]). Part of this are user- and computer accounts as well as user groups. Furthermore, domain related configuration data is stored in it.

3.2. Fundamental protocols

Within Active-Directory, a set of fundamental protocols are used. The basis is created by the *Internet Protocol* (IP) [RFC791] as well as the *Transmission Control Protocol* (TCP) [RFC793] and the *User Datagram Protocol* (UDP) [RFC768]. For the resolution of names to IP-Addresses, the protocols *Domain Name Service* (DNS) [RFC1034, RFC1035] as well as *Netbios Name Service* [RFC1001, RFC1002] are used.

3.2.1. Authentication protocols

The interaction of the networking protocols is displayed in Figure 3.5, “ Overview over the networking protocols (compare “Replication and LDAP Client-Server Architecture” in [MSTN01]). ”. All protocols on a higher level than TCP use authenticated sessions, where *Kerberos 5* (KRB5) [RFC4120] and *New Technology Lan Manager Secure Service Provider* (NTLMSSP) [Gla2006] are being used. Partially, for the negotiation of the respectively used protocols, additional protocols, such as *Generic Security Service Application Program Interface* (GSSAPI) [RFC2473], *Simple and Protected Negotiation* (SPNEGO) [RFC4178] and/or *GSimple Authentication and Security Layer* (SASL) [RFC2222], are being used.

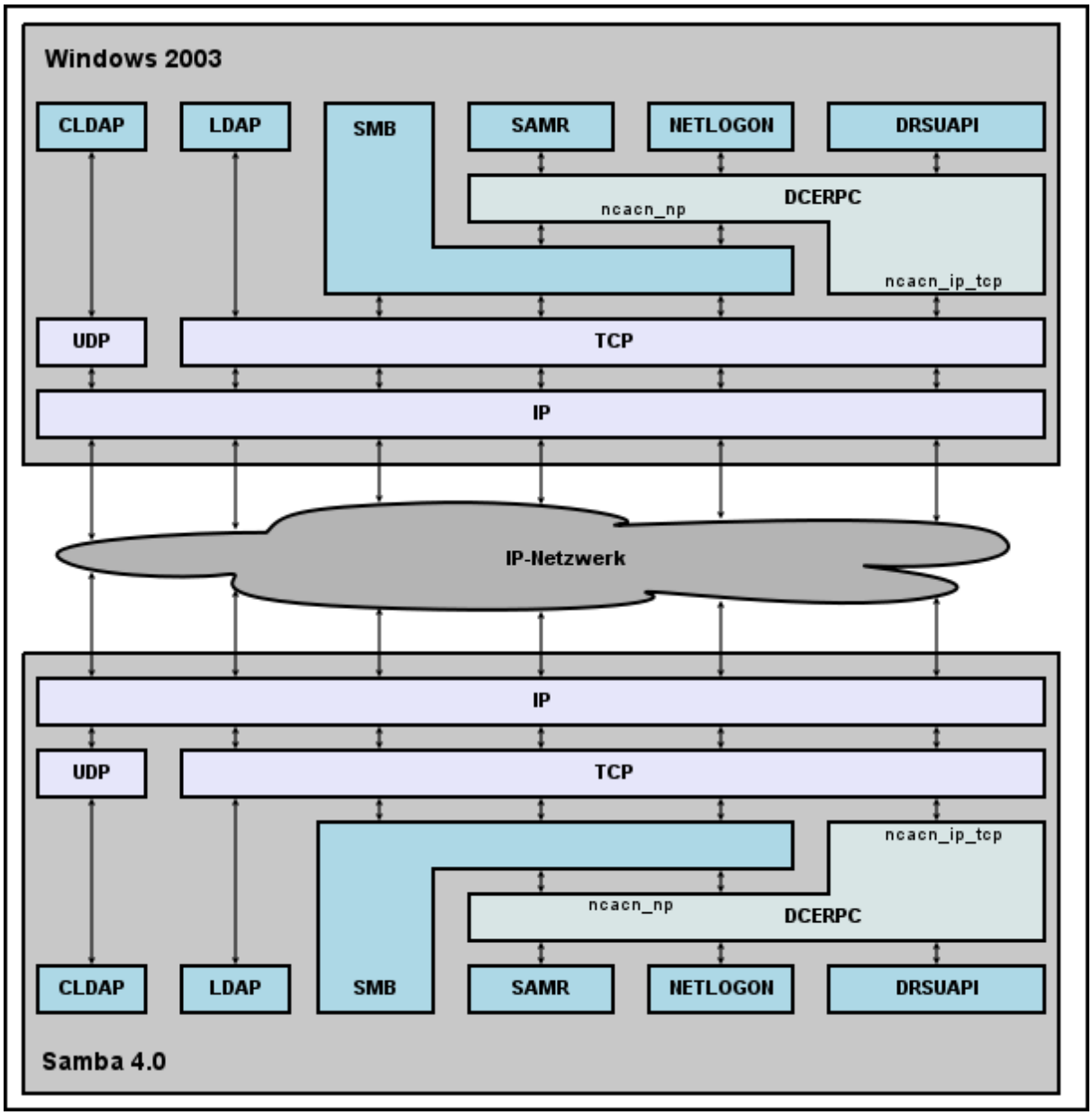

Figure 3.5. Overview over the networking protocols (compare "Replication and LDAP Client-Server Architecture" in [MSTN01]).

3.2.2. LDAP

The *Lightweight Directory Access Protocol* (LDAP) [RFC4510] is a protocol, that makes access to directory objects and their attributes possible. The protocol is based on TCP and uses the TCP port 389. Typically, a client has to authenticate itself with `ldap_bind()` on the server. Then, the functions `ldap_add()`, `ldap_modify()`, `ldap_modify_rdn()` and `ldap_delete()` are available for the processing of a specific directory object. Furthermore, the function `ldap_search()` serves the listing of directory objects. Through the Base-DN, the search is limited to a determined subtree within the directory hierarchy. For the further limitation of the search, a search area is given by `LDAP_SCOPE_BASE` (only Base-DN), `LDAP_SCOPE_ONELEVEL` (only direct child objects of Base-DN) or `LDAP_SCOPE_SUBTREE` (all objects under Base-DN and Base-DN itself). In order to only

list objects with specific attribute-values, the search filter determines, for example "(&(objectClass=user)(name=Administrator))", which objects the server should send to the client. Lastly, an attribute list determines, which attributes of the found objects should be delivered. In the search filter and attribute-list, "*" may be used as Wildcard.

3.2.3. CLDAP

The *Connection-less Lightweight Directory Access Protocol*(CLDAP) [RFC3352] is the variant of LDAP without connection capabilities. It is based on UDP and does not use any authentication. Hence, only search requests are supported.

3.2.4. SMB

The protocol *Server Message Block* (SMB) [4] [Her2003] is connection-oriented and based on TCP. Thereby TCP port 139 as well as TCP port 445 are being used. For port 139, between SMB and TCP lies formally still the *Netbios Session Service* [RFC1001,RFC1002], however, this can only be noted when establishing a connection by two additional protocol-messages. SMB provides clearances of data systems and printers. The special "IPC$"-Clearance enables furthermore communication through the means of *Named Pipes*.

3.2.5. DCERPC

Distributed Computing Environment / Remote Procedure Call (DCERPC) [OpGr01] is a generic protocol for the transmission of function invocations on a distant computer. DCERPC can use different transport-protocols. Within Active-Directory, TCP ("ncacn_ip_tcp") and SMB-Named-Pipes ("ncacn_np") are used as transport-protocols. With DCERPC, used functions are consolidated into interfaces. Those interfaces are specified with the aid of *Interface Definition Language* (IDL) [OpGr03]. The transmission of function parameters and return values is realised with aid of *Network Data Representation* (NDR) [OpGr02]. DCERPC handles session management including authentication and optional encryption. Server mostly do not use fixed TCP ports for "ncacn_ip_tcp". Instead, they register themselves at the *End-point-Mapper* (see [OpGr04]), there, clients may request the respectively used TCP ports.

3.2.6. DCERPC based interfaces/protocols

Within Active-Directory, a set of DCERPC-based protocols are used. However, the IDL-specifications used by Microsoft are not public. The best open, but incomplete IDL-specifications for those protocols are maintained by the Samba-Team and a group of other volunteers (see [SaTe03]).

3.2.7. SAMR

The SAMR-protocol provides functions for user- and group management (see samr.idl under [SaTe03]). The name comes from the *Security Account Manager* (SAM), used in Windows-NT 4.0.

[4] SMB is also known under the name *Common Internet File System* (CIFS).

3.2.8. NETLOGON

The NETLOGON-protocol provides the functions for authentication within Windows domains. These functions are used among others for NTLMSSP-authentication. Furthermore, the NETLOGON-interface provides functions for the SAM-replication used in Windows-NT 4.0 (see netlogon.idl in [SaTe03]).

3.2.9. DRSUAPI

The protocol *Directory Replication Service Update API* (DRSUAPI) provides functions, that are used for the Active-Directory-Replication (see drsuapi.idl in [SaTe03]). Furthermore, the function `DsCrackNames()` is provided, which converts names of directory objects from different naming formats.

3.3. Fundamental concepts of the replication system

3.3.1. Source-DSA vs. Destination-DSA

At the moment of replication, changes within the data stock have been transmitted from one server, the so-called Source-DSA, to another Server, the so-called Destination-DSA.

3.3.2. Singlemaster-Replication vs. Multimaster-Replication

For the replication of data stocks, it can be basically differentiated between the *Singlemaster*- and the *Multimaster-Model*.

Singlemaster-Replication

Within the relatively simple *Singlemaster-Replication*, write accesses are only possible to one server and all other servers only may provide read access, since otherwise the data stock could become inconsistent.

Multimaster-Replication

Within the more complex *Multimaster-Replication,* write accesses are possible to all servers. However, here inconsistencies have to be well established within the concept. Conflicts should thus be resolved by algorithms, so that each server can work on its own. That means a conflict is resolved on each server in the same manner, so that the data stock becomes consistent again.

In Active-Directory, the *Multimaster-Replication* is used, although here, several specific system-critical write operations are only permitted on one server, the *FSMO-Role-Owner*[6].

3.3.3. Push-Replication vs. Pull-Replication

Generally, when replicating data stocks, it is differentiated between *Push-Replication* and *Pull-Replication*.

Push-Replication

For *Push-Replication,* the server, on which the data has been changed, decides when and which data it has to send to its replication-partner. In order to maintain this system efficient, the Source-DSA has to keep track of which data has already been transmitted. Although if a connection is terminated or through other errors, it can happen, that data does not arrive correctly at the Destination-DSA, but the Source-DSA assumes a faultless transmission. In this case the Destination-DSA will never receive some changes.

Pull-Replication

In the *Pull-Replication*, each server knows, what data it already received. The Destination-DSA tells the Source-DSA, which data it has at each replication-request. After that, the Source-DSA can determine, what data it has to provide for the Destination-DSA, so that the data stock on both servers coincide. This procedure is by far not as sensitive to transmission errors, as the *Push-Replication*. The *Pull-Replication*is typically triggered in configurable intervals, since the Destination-DSA does not know, when changes are made on the Source-DSA. Additionally, a so-called *Change-Notify* mechanism is often used. Here, a Source-DSA sends notifications of changes to the Destination-DSA, which then reacts with an immediate *Pull-Replication*.

Active-Directory uses the *Pull-Replication* with *Change-Notify*.

3.3.4. Log-based vs. Status-based replications

In order to make data replication efficient, it is necessary to keep track of write accesses. Here, there exist two basic approaches, the log-based and the status-based.

Log-based replication

In log-based replication, each change is written into a log file. The advantage of this procedure is that each change can be retraced and old conditions can be recreated. However, the necessary storage place grows linearly to the number of changes and not the size of the respective current data stocks.

Status-based replication

The status-based replication goes a different way, that is, only the respective current data stock is saved. Here, a sequence number is used, which is incremented upon each change. At the moment of a change, the current sequence number is assigned to the changed data set. For the replication, it is only needed to know, until which sequence number the transmission was executed successfully, then, only data sets with a higher sequence number have to be transmitted. Here, the storage space is linear to the size of the current data stock.

Active-Directory uses the status-based replication.

3.3.5. Store-and-Forward-replication

For the replication with many participating servers, it is desirable, that each server does not have to replicate with each other server. The principle of the *Store-and-Forward-Replication* allows for an indirect transmission of changes between two servers, that do not have a direct network connection. The replication could look as follows: Server A replicates with Server B, Server B replicates with Server C. In such a *Replication-Topology*, changes on Server A are primarily replicated to Server B and from there to Server C. This topology can be extended arbitrarily. Each server knows, what changes it received already from which other server. The path (directly or indirectly), through which the changes were transmitted, is irrelevant.

In Active-Directory, the *Store-and-Forward-Principle* is being used. The structure of the used *Replication-Topology* (see [MSTN02]) is not explained further in this bachelor thesis.

3.4. Fundamental attributes and structures

In order to describe the replication used within Active-Directory, a set of fundamental attributes, structures and terms are necessary. Those will be explained shortly hereafter.

3.4.1. The “objectGUID” attribute

Since objects within a directory partition may be moved or renamed, the Object-DN is not constantly clearly defined. This is why, each directory object has a so-called *Global Unique Identifier* (GUID)[5], that never changes. Roughly seen, a GUID is a 128-bit (16 byte) random number. Some describe a GUID as an IDL-structure (see Figure 3.6, “The IDL-description of the *Directory-Object-Identifier* fundamental structures.”), whose NDR-coding is an array with 16 bytes. However, often a string-representation is used, for instance “077157f1-eecc-47bf-ba03-9753c93e2e73”. The GUID of the directory object is NDR-coded and saved within the attribute “objectGUID”.

3.4.2. The “objectSid” attribute

Domains, user-accounts and groups additionally contain a *Security Identifier* (SID)[6], which is used for the identification at the moment of authentication evaluation. The SID is as `struct dom_sid`[7] described in IDL (see Figure 3.6, “The IDL-description of the *Directory-Object-Identifier* fundamental structures.”). However, normally the string-representation is used, for example “S-1-5-21-4258141497-2604123113-439883119- 1286”. Here, the SID is composed of the Domain-SID and the *Relative Identifier* (RID) for the user or the group. In the example, “S-1-5-21-4258141497-2604123113-439883119” is the Domain-SID and “1286” the user-RID. The SID of a directory object is NDR-coded and saved within the attribute “objectSid”.

3.4.3. The structure “DsReplicaObjectIdentifier”

Within the DRSUAPI-protocol, a object is referenced by an *Directory-Object-Identifier*[8] Here, always one of the three items must have a valid value, the other two are optional and can also be initialised with NULL-bytes.

[5]Although the german term should be “the GUID”, in the literature it is always talked about “a GUID”. A guid is also known as *Universal Unique Identifier* (UUID).

[6]Here the corresponding literature also speaks of “a SID”.

[7]`struct dom_sid2` and `struct dom_sid28` differentiate themselves minimally in their NDR-coding. However, this difference can be neglected here.

[8]For the biggest part, `struct drsuapi_DsReplicaObjectIdentifier` is used (see appendix A2 starting at row 128). However, there also exist variations, for example `struct drsuapi_DsReplicaObjectIdentifier3`, which only are different on NDR-level.

```
/* aus misc.idl unter [Sate03] */
typedef [public,noprint,gensize,noejs] struct {
    uint32 time_low;
    uint16 time_mid;
    uint16 time_hi_and_version;
    uint8  clock_seq[2];
    uint8  node[6];
} GUID;

/* aus security.idl unter [SaTe03] */
typedef [public,gensize,noprint,noejs,nosize] struct {
    uint8  sid_rev_num;             /**< SID revision number */
    [range(0,15)] int8  num_auths;  /**< Number of sub-authorities */
    uint8  id_auth[6];              /**< Identifier Authority */
    uint32 sub_auths[num_auths];
} dom_sid;

/* aus drsuapi.idl unter [SaTe03], bzw. Anhang A2 ab Zeile 128 */
typedef [public,gensize] struct {
    [value(ndr_size_drsuapi_DsReplicaObjectIdentifier(r, ndr->flags)-4)] uint32 __ndr_size;
    [value(ndr_size_dom_sid28(&sid, ndr->flags))]  uint32 __ndr_size_sid;
    GUID guid;
    dom_sid28 sid;
    [flag(STR_SIZE4|STR_CHARLEN|STR_CONFORMANT)] string dn;
} drsuapi_DsReplicaObjectIdentifier;
```

Figure 3.6. The IDL-description of the *Directory-Object-Identifier* fundamental structures.

3.4.4. The object“NTDS Settings”

Each DSA saves its own configuration within a object of the class “nTDSDSA” and the RDN “CN=NTDS Settings”. An example-object is shown by Figure 3.7, “A typical “NTDS Settings” Object.”. Here, the attributes “objectGUID” and “invocationId” are especially relevant. Both contain a NDR-copied GUID. The so-called *DSA-GUID*, which represents the DSA unambiguously and never changes, is stored within the attribute “objectGUid”. The respective current directory data stock used by the DSA has its own GUID, the so-called *DSA-Invocation-Id*, which is stored within the attribute “invocationId”. The *DSA-Invocation-ID* is generated at the moment of installation. For the DSA that was firstly installed into the domain forest, the *DSA-GUID* and the *DSA-Invocation-Id* have identical values. Only in special cases, for instance when changing back to a backup, the *DSA-Invocation-Id* is changed.

```
dn: CN=NTDS Settings,CN=W2K3-104,CN=Servers,CN=Standardname-des-ersten-Standor
 ts,CN=Sites,CN=Configuration,DC=w2k3,DC=vmnet1,DC=vm,DC=base
objectClass: top
objectClass: applicationSettings
objectClass: nTDSDSA
cn: NTDS Settings
distinguishedName: CN=NTDS Settings,CN=W2K3-104,CN=Servers,CN=Standardname-des
 -ersten-Standorts,CN=Sites,CN=Configuration,DC=w2k3,DC=vmnet1,DC=vm,DC=base
instanceType: 4
whenCreated: 20040725122652.0Z
whenChanged: 20040725125936.0Z
hasMasterNCs: DC=sub1,DC=w2k3,DC=vmnet1,DC=vm,DC=base
hasMasterNCs: CN=Schema,CN=Configuration,DC=w2k3,DC=vmnet1,DC=vm,DC=base
hasMasterNCs: CN=Configuration,DC=w2k3,DC=vmnet1,DC=vm,DC=base
uSNCreated: 7080
dMDLocation: CN=Schema,CN=Configuration,DC=w2k3,DC=vmnet1,DC=vm,DC=base
invocationId:: wKUazBD0Gk6Q/C99YLlFZw==
uSNChanged: 16402
showInAdvancedViewOnly: TRUE
name: NTDS Settings
objectGUID:: +fTt+ugNgkW4tsR17+++Wg==
systemFlags: 33554432
serverReferenceBL: CN=W2K3-104,CN=Domain System Volume (SYSVOL share),CN=File
 Replication Service,CN=System,DC=sub1,DC=w2k3,DC=vmnet1,DC=vm,DC=base
objectCategory: CN=NTDS-DSA,CN=Schema,CN=Configuration,DC=w2k3,DC=vmnet1,DC=vm
 ,DC=base
dSCorePropagationData: 20040725123509.0Z
dSCorePropagationData: 16010101000001.0Z
msDS-Behavior-Version: 2
msDS-HasDomainNCs: DC=sub1,DC=w2k3,DC=vmnet1,DC=vm,DC=base
msDS-hasMasterNCs: DC=DomainDnsZones,DC=sub1,DC=w2k3,DC=vmnet1,DC=vm,DC=base
msDS-hasMasterNCs: DC=sub1,DC=w2k3,DC=vmnet1,DC=vm,DC=base
msDS-hasMasterNCs: DC=ForestDnsZones,DC=w2k3,DC=vmnet1,DC=vm,DC=base
msDS-hasMasterNCs: CN=Schema,CN=Configuration,DC=w2k3,DC=vmnet1,DC=vm,DC=base
msDS-hasMasterNCs: CN=Configuration,DC=w2k3,DC=vmnet1,DC=vm,DC=base
```

Figure 3.7. A typical “NTDS Settings” Object.

3.4.5. The attribute“highestCommitedUSN”

For Active-Directory, a 64-bit (“unsigned”) *Update Sequence Number* (USN) is used, which is incremented at each write access. This USN is local for the respective DSA. The respective currently highest USN is provided by each DSA within the attribute “highestCommitedUSN” of the RootDSE (see Figure 3.2, “A typical RootDSE object (a pair of attribute values has been removed in order to give a better overview).”).

3.4.6. The structures “DsReplicaCursor” and “DsReplicaCursorCtr”

In order to execute the *Store-and-Forward-Principle*, each change is described unambigously within Active-Directory by the DSA-Invocation-ID and the corresponding USN. This data is transmitted within the DRSUAPI-protocol as `struct drsuapi_DsReplicaCursorCtr`, that means as an array with `struct drsuapi_DsReplicaCursor` items (see Figure 3.8, “The IDL-description of the *Up-To-Date-Vector* fundamental structures.”). Those structures also exist with an additional timestamp, which indicates the time of the last successful replication. `struct drsuapi_DsReplicaCursorCtr` is also known as *Up-To-Date-Vector*.

```
/* aus drsuapi.idl unter [SaTe03] bzw. Anhang A2 ab Zeile 186
typedef [public] struct {
    GUID source_dsa_invocation_id; /* the 'invocationId' field of the CN=NTDS Settings object */
    hyper highest_usn;  /* (hyper == uint64) updated after a full replication cycle */
} drsuapi_DsReplicaCursor;

/* aus drsuapi.idl unter [SaTe03] bzw. Anhang A2 ab Zeile 1209
typedef struct {
    uint32 count;
    uint32 reserved;
    [size_is(count)] drsuapi_DsReplicaCursor cursors[];
} drsuapi_DsReplicaCursorCtr;
```

Figure 3.8. The IDL-description of the *Up-To-Date-Vector* fundamental structures.

3.4.7. The structure “DsReplicaHighWaterMark”

When directly replicating between two servers, a so-called *High-Water-mark* is exchanged. Figure 3.9, “The IDL-description of the *High-Water-Mark* structure.” shows its structure. The *High-Water-Mark* contains the USNs that are already known to the *Destination-DSA* by the *Source-DSA*.

```
/* aus drsuapi.idl unter [SaTe03] bzw. Anhang A2 ab Zeile 180 */
typedef [public] struct {
    /* (hyper == uint64) */
    hyper tmp_highest_usn; /* updated after each object update */
    hyper reserved_usn;
    hyper highest_usn; /* updated after a full replication cycle */
} drsuapi_DsReplicaHighWaterMark;
```

Figure 3.9. The IDL-description of the *High-Water-Mark* structure.

3.4.8. The attributes “repsFrom” and “repsTo”

The *High-Water-Mark* is saved together with some other information on the corresponding DSA within the attribute “repsFrom”. For each Source-DSA, from which is replicated, an own attribute-value exists, which contains the NDR-coded form of `struct repsFromToBlob` (Figure 3.10, “The IDL-description of the “repsFromToBlob” structure.”). For each Destination-DSA, that shall be informed about changes using *Change-Notify*, analogously an attribute value of the attribute “repsTo” is present. The DSAs, with which the local DSA replicates in any way, are also called *Replication-Neighbours*.

```
/* aus drsblobs.idl unter [SaTe03] bzw. Anhang A1 ab Zeile 82
 * repsFrom/repsTo
 * w2k  uses version 1
 * w2k3 uses version 1
 */
typedef [public,gensize] struct {
    asclstr dns_name;
} repsFromTo1OtherInfo;

typedef [public,gensize,flag(NDR_PAHEX)] struct {
    /* this includes the 8 bytes of the repsFromToBlob header */
    [value(ndr_size_repsFromTo1(r, ndr->flags)+8)] uint32 blobsize;
    uint32 consecutive_sync_failures;
    NTTIME_1sec last_success;
    NTTIME_1sec last_attempt;
    WERROR result_last_attempt;
    [relative] repsFromTo1OtherInfo *other_info;
    [value(ndr_size_repsFromTo1OtherInfo(other_info, ndr->flags))] uint32 other_info_length;
    drsuapi_DsReplicaNeighbourFlags replica_flags;
    uint8 schedule[84];
    uint32 reserved;
    drsuapi_DsReplicaHighWaterMark highwatermark;
    GUID source_dsa_obj_guid; /* the 'objectGuid' field of the CN=NTDS Settings object */
    GUID source_dsa_invocation_id; /* the 'invocationId' field of the CN=NTDS Settings object */
    GUID transport_guid;
} repsFromTo1;

typedef [nodiscriminant] union {
    [case(1)] repsFromTo1 ctr1;
} repsFromTo;

typedef [public] struct {
    uint32 version;
    uint32 reserved;
    [switch_is(version)] repsFromTo ctr;
} repsFromToBlob;

/* siehe auch struct drsuapi_DsReplicaNeighbour in drsuapi.idl unter [SaTe03] bzw. Anhang A2 ab Zeile 1186 */
```

Figure 3.10. The IDL-description of the “repsFromToBlob” structure.

```
/* drsuapi.idl unter [SaTe03] bzw. Anhang A2 ab Zeile 201 */
typedef [public] bitmap {
    /* the _WRITEABLE flag indicates a replication with all attributes
     *
     * --metze
     */
    DRSUAPI_DS_REPLICA_NEIGHBOUR_WRITEABLE                          = 0x00000010,
    DRSUAPI_DS_REPLICA_NEIGHBOUR_SYNC_ON_STARTUP                    = 0x00000020,
    DRSUAPI_DS_REPLICA_NEIGHBOUR_DO_SCHEDULED_SYNCS                 = 0x00000040,
    DRSUAPI_DS_REPLICA_NEIGHBOUR_USE_ASYNC_INTERSIDE_TRANSPORT      = 0x00000080,
    DRSUAPI_DS_REPLICA_NEIGHBOUR_TWO_WAY_SYNC                       = 0x00000200,
    DRSUAPI_DS_REPLICA_NEIGHBOUR_RETURN_OBJECT_PARENTS              = 0x00000800,
    DRSUAPI_DS_REPLICA_NEIGHBOUR_FULL_IN_PROGRESS                   = 0x00001000, /* was 0x00010000, */
    DRSUAPI_DS_REPLICA_NEIGHBOUR_FULL_NEXT_PACKET                   = 0x00002000, /* was 0x00020000, */
    DRSUAPI_DS_REPLICA_NEIGHBOUR_NEVER_SYNCED                       = 0x00200000,
    DRSUAPI_DS_REPLICA_NEIGHBOUR_PREEMPTED                          = 0x01000000,
    DRSUAPI_DS_REPLICA_NEIGHBOUR_IGNORE_CHANGE_NOTIFICATIONS        = 0x04000000,
    DRSUAPI_DS_REPLICA_NEIGHBOUR_DISABLE_SCHEDULED_SYNC             = 0x08000000,
    /*
     * the following NOTE applies to DsGetNCChangesRequest5:
     *  - the data is only compressed when 10 or more objects are replicated
     *  - but there could also be a size limit of 35 KBytes or something like that
     *  - the reply is DsGetNCChangesCtr2
     *  - maybe the same applies to DsGetNCChangesRequest8...
     *
     *  --metze
     */
    DRSUAPI_DS_REPLICA_NEIGHBOUR_COMPRESS_CHANGES                   = 0x10000000,
    DRSUAPI_DS_REPLICA_NEIGHBOUR_NO_CHANGE_NOTIFICATIONS            = 0x20000000,
    DRSUAPI_DS_REPLICA_NEIGHBOUR_PARTIAL_ATTRIBUTE_SET              = 0x40000000
} drsuapi_DsReplicaNeighbourFlags;
```

Figure 3.11. The Bit-flags “DsReplicaNeighbourFlags”.

3.4.9. ASN.1 “Object Identifier”

Attributes and object-classes are identified within the Schema unambigously by an *ASN.1 Object Identifier* (OID), for instance “1.3.6.1.4.1.7165.4.255.1” (see [ITUT01]). In the Active-Directory-Schema, the OID-value is stored as an ASCII-string. For attributes, the “attributeID” attribute and for classes, the “governsID” attribute is stored. The OID does have nothing in common with the “DsReplicaObjectIdentifier” structure!

3.4.10. The “Prefix-Mapping”

Within the DRSUAPI-protocol, attributes and object-classes are not referred to by their name, but by a 32-bit (unsigned) integer value. This 32-bit-value is derived from the OID-value. Here, the OID-string is separated in prefix and the last item at the point of the last occurrence of the character “.” (see Figure 3.12, “Example for the *Prefix-Mapping* including the basis-table used within Active-Directory. ”). Because the OIDs used within the Schema mostly only differ in the last item, a table with so-called *Prefix-Mappings* is used. Here, the OID-prefixes are depicted onto 16-bit (unsigned) integer values. The 16-bit-prefix is written to the 16 higher value bits of the 32-bit-value. The last item is also interpreted as 16-bit (unsigned) integer value and written into the lower value 16-bit of the 32-bit-value.Figure 3.12, “Example for the *Prefix-Mapping* including the basis-table used within Active-Directory. ” shows a list with the *Prefix-Mappings* used for Active-Directory. The 32-bit-value is also called *AttId* for attributes.

```
/* aus drsuapi.idl unter [SaTe03] bzw. Anhang A2 ab Zeile 327 */
 * In DRSUAPI all attributes with syntax 2.5.5.2
 * are identified by uint32 values
 *
 * the following table shows the mapping used between the two representations
 * e.g. - objectClass 'nTDSDSA' has governsID: 1.2.840.113556.1.5.7000.47
 *        and a UINT32-ID of '0x0017002F'.
 *      - so the OID 1.2.840.113556.1.5.7000.47 is splitted into a
 *        OID-prefix: 1.2.840.113556.1.5.7000
 *        and a value: 47 => 0x2F
 *      - the mapping table gives a UINT32-prefix: 0x00170000
 *      - and the UINT32-ID is 0x0017002F = 0x00170000 | 0x2F
 *
 * This prefix mapping table is replied in the drsuapi_DsReplicaOIDMapping_Ctr
 * array. The following are the default mappings of w2k3
 *
 * OID-prefix                   => UINT32-Id prefix
 *
 * 2.5.4.*                      => 0x00000000 (standard attributes RFC2256 core.schema)
 * 2.5.6.*                      => 0x00010000 (standard object classes RFC2256 core.schema)
 * 1.2.840.113556.1.2.*         => 0x00020000
 * 1.2.840.113556.1.3.*         => 0x00030000
 * 2.5.5.*                      => 0x00080000 (attributeSyntax OID's)
 * 1.2.840.113556.1.4.*         => 0x00090000
 * 1.2.840.113556.1.5.*         => 0x000A0000
 * 2.16.840.1.113730.3.*        => 0x00140000
 * 0.9.2342.19200300.100.1.*    => 0x00150000
 * 2.16.840.1.113730.3.1.*      => 0x00160000
 * 1.2.840.113556.1.5.7000.*    => 0x00170000
 * 2.5.21.*                     => 0x00180000 (attrs for SubSchema)
 * 2.5.18.*                     => 0x00190000 (createTimeStamp,modifyTimeStamp, SubSchema)
 * 2.5.20.*                     => 0x001A0000
 * 1.3.6.1.4.1.1466.101.119.*   => 0x001B0000 (dynamicObject, entryTTL)
 * 2.16.840.1.113730.3.2.*      => 0x001C0000
 * 1.3.6.1.4.1.250.1.*          => 0x001D0000
 * 1.2.840.113549.1.9.*         => 0x001E0000 (unstructuredAddress,unstructuredName)
 * 0.9.2342.19200300.100.4.*    => 0x001F0000
 */
```

Figure 3.12. Example for the *Prefix-Mapping* including the basis-table used within Active-Directory.

3.4.11. The structures “DsReplicaOIDMapping” and “DsReplicaOIDMapping_Ctr”

Within the DRSUAPI-protocol, the *Prefix-Mapping* table is transmitted as an array with `struct drsuapi_DsReplicaOIDMapping` items (see Figure 3.13, “The IDL-description of the *Prefix-Mapping* fundamental structures.”).

```
/* aus drsuapi.idl unter [SaTe03] bzw. Anhang A2 ab Zeile 327 */
typedef [nopush,nopull] struct {
    [range(0,10000),value(ndr_size_drsuapi_DsReplicaOID_oid(oid, 0))] uint32 __ndr_size;
    [size_is(__ndr_size),charset(DOS)] uint8 *oid; /* it's encoded with asn1_write_OID_String() */
} drsuapi_DsReplicaOID;

typedef struct {
    uint32 id_prefix;
    drsuapi_DsReplicaOID oid;
} drsuapi_DsReplicaOIDMapping;

typedef [public] struct {
    [range(0,0x100000)] uint32 num_mappings;
    [size_is(num_mappings)] drsuapi_DsReplicaOIDMapping *mappings;
} drsuapi_DsReplicaOIDMapping_Ctr;
```

Figure 3.13. The IDL-description of the *Prefix-Mapping* fundamental structures.

3.4.12. Originating-Updates vs. Replicating-Updates

Changes, that were generated by client applications, that is to say that they were not caused by the replication, are called *Originating-Updates*. Replicated changes are called *Replicating-Updates*.

3.4.13. The attribute “replPropertyMetaData”

The meta data necessary for the replication are stored within the “replPropertyMetaData” attribute for each directory object. In Active-Directory, not all attributes are replicated, and some attributes are dynamically created at the moment of a search-request. The attribute “replPropertyMetaData” is an attribute that is non replicable and only has meaning for the local DSA. The contents of this attribute is NDR-coded and described in IDL as `struct replPropertyMetaDataBlob` (see Figure 3.14, “The IDL-description of the “replProperyMetaData” and “DsReplicaMetaDataCtr” fundamental structures.”). Meta data for each replicated attribute are stored in an array made out of `struct replPropertyMetaData1` items. Attribute are referred to by the 32-bit *AttId*. At each *Originating-Update* of an attribute, the version field is incremented and the time of change, DSA-Invocation-Id and the USN (“originating” and “local”) are set At a *Replicating-Update*, all meta data items, except the local USN, are transmitted as `struct drsuapi_DsReplicaMetaDataCtr` from the Source-DSA to Destination-DSA. The local USN is then added at the moment off loading of the replicated changes by the Destination-DSA.

```
/* aus drsblobs.idl unter [SaTe03] bzw. Anhang A1 ab Zeile 14
 * replPropertyMetaData
 * w2k  uses version 1
 * w2k3 uses version 1
 */
/* (hyper == uint64) */
typedef struct {
    drsuapi_DsAttributeId attid; /* == uint32 */
    uint32 version;
    NTTIME_1sec originating_change_time; /* (uint64) in secs since 1601 on the wire, converted to 100 nanosec steps */
    GUID originating_invocation_id;
    hyper originating_usn; /* == uint64 */
    hyper local_usn; /* == uint64 */
} replPropertyMetaData1;

typedef struct {
    uint32 count;
    uint32 reserved;
    replPropertyMetaData1 array[count];
} replPropertyMetaDataCtr1;

typedef [nodiscriminant] union {
    [case(1)] replPropertyMetaDataCtr1 ctr1;
} replPropertyMetaDataCtr;

typedef [public] struct {
    uint32 version;
    uint32 reserved;
    [switch_is(version)] replPropertyMetaDataCtr ctr;
} replPropertyMetaDataBlob;

/* aus drsuapi.idl unter [SaTe03] bzw. Anhang A2 ab Zeile 481 */
typedef struct {
    uint32 version;
    NTTIME_1sec originating_change_time; /* (uint64) in secs since 1601 on the wire, converted to 100 nanosec steps */
    GUID originating_invocation_id;
    hyper originating_usn; /* == uint64 */
} drsuapi_DsReplicaMetaData;

typedef [public] struct {
    [range(0,1048576)] uint32 count;
    [size_is(count)] drsuapi_DsReplicaMetaData meta_data[];
} drsuapi_DsReplicaMetaDataCtr;
```

Figure 3.14. The IDL-description of the “replProperyMetaData” and “DsReplicaMetaDataCtr” fundamental structures.

3.4.14. The structure “DsReplicaObject”

In the DRSUAPI-protocol, directory objects are transmitted as `struct drsuapi_DsReplicaObject` (see Figure 3.15, “The IDL-description of the “DsReplicaObject” fundamental structure.”). An object consists of an *directory object identifier* and an attribute array made out of `struct drsuapi_DsReplicaAttribute` items. Each attribute is then again referred to by the *AttId*. Furthermore, each attribute contains a value-array made out of `struct drsuapi_DsAttributeValue` items. Each value is composed by an Byte-Array (`DATA_BLOB`).

```
/* aus drsuapi.idl unter [SaTe03] bzw. Anhang A2 ab Zeile 436 */
typedef struct {
    [range(0,10485760),value(ndr_size_DATA_BLOB(0,blob,0))] uint32 __ndr_size;
    DATA_BLOB *blob; /* struct data_blob { uint32 length; uint8 data[length]; } (byte-array) */
} drsuapi_DsAttributeValue;

typedef struct {
    [range(0,10485760)] uint32 num_values;
    [size_is(num_values)] drsuapi_DsAttributeValue *values; /* array! */
} drsuapi_DsAttributeValueCtr;

/* aus drsuapi.idl unter [SaTe03 bzw. Anhang A2 ab Zeile 465 */
typedef [public] struct {
    drsuapi_DsAttributeId attid; /* == uint32 */
    drsuapi_DsAttributeValueCtr value_ctr;
} drsuapi_DsReplicaAttribute;

typedef struct {
    [range(0,1048576)] uint32 num_attributes;
    [size_is(num_attributes)]  drsuapi_DsReplicaAttribute *attributes; /* array! */
} drsuapi_DsReplicaAttributeCtr;

typedef [public] struct {
    drsuapi_DsReplicaObjectIdentifier *identifier;
    uint32 unknown1;
    drsuapi_DsReplicaAttributeCtr attribute_ctr;
} drsuapi_DsReplicaObject;
```

Figure 3.15. The IDL-description of the “DsReplicaObject” fundamental structure.

3.4.15. The list-items “DsReplicaObjectListItem” and “DsReplicaObjectListItemEx”

In the DRSUAPI-protocol, directory objects are transmitted as simply concatenated lists. Here, there are two variants. Either the list-items contain only the `struct drsuapi_DsReplicaObject` or additionally the GUID of the parent-object as well as a meta data array, which has the same size as the attribute-array, whereby the array-items with the same index also belong together (see Figure 3.16, “The IDL-description of the “DsReplicaObjectListItem” and “DsReplicaObjectListItemEx” list items.”).

```
/* aus drsuapi.idl unter [SaTe03] bzw. Anhang A2 ab Zeile 1018 */
typedef [public,noprint] struct {
    drsuapi_DsReplicaObjectListItem *next_object;
    drsuapi_DsReplicaObject object;
} drsuapi_DsReplicaObjectListItem;

/* aus drsuapi.idl unter [SaTe03] bzw. Anhang A2 ab Zeile 493 */
typedef [public,noprint] struct {
    drsuapi_DsReplicaObjectListItemEx *next_object;
    drsuapi_DsReplicaObject object;
    uint32 unknown1;
    GUID *parent_object_guid;
    drsuapi_DsReplicaMetaDataCtr *meta_data_ctr;
} drsuapi_DsReplicaObjectListItemEx;
```

Figure 3.16. The IDL-description of the “DsReplicaObjectListItem” and “DsReplicaObjectListItemEx” list items.

3.4.16. The attributes “uSNCreate” and “uSNChanged”

The attributes “uSNCreate” and “uSNChanged” each contain the local USNs. “uSNCreate” is set at the moment of creation of the directory-object on the local DSA. “uSNChanged” contains the USn of the last attribute-change of the directory-object. Analogously to the USN-attributes, the attributes “whenCreated” and “whenChanged”, which contain timestamps in the form of “20070309000909”, exist.

3.4.17. The attribute “instanceType”

The attribute “instanceType”contains the bit-flags linked through disjunction. Each directory-object has this attribute. For writable objects, the `INSTANCE_TYPE_WRITE`bit-flag is set, furthermore, the root-object of the root-domain within the domain-forest has additionally set the `INSTANCE_TYPE_IS_NC_HEAD` bit-flag. All the other root-objects of a directory partition have additionally set the bit-flags `INSTANCE_TYPE_IS_NC_HEAD` and `INSTANCE_TYPE_NC_HEAD_ABOVE`. Figure 3.17, “The bit-flags for the “instance Type” attribute.” shows all the possible bit-flags.

```
/* aus source/dsdb/common/flags.h unter [SaTe01] */
#define INSTANCE_TYPE_IS_NC_HEAD        0x00000001
#define INSTANCE_TYPE_UNINSTANT         0x00000002
#define INSTANCE_TYPE_WRITE             0x00000004
#define INSTANCE_TYPE_NC_ABOVE          0x00000008
#define INSTANCE_TYPE_NC_COMING         0x00000010
#define INSTANCE_TYPE_NC_GOING          0x00000020
```

Figure 3.17. The bit-flags for the “instance Type” attribute.

3.4.18. The attribute “systemFlags”

The attribute “systemFlags”contains the bit-flags linked through disjunction. It is only present for system-critical directory objects, in order to, for example, prevent deleting. Figure 3.18, “The bit-flags for the “systemFlags” attribute.” shows all the possible bit-flags.

```
/* aus source/dsdb/common/flags.h unter [SaTe01] */
#define SYSTEM_FLAG_CR_NTDS_NC                  0x00000001
#define SYSTEM_FLAG_CR_NTDS_DOMAIN              0x00000002
#define SYSTEM_FLAG_CR_NTDS_NOT_GC_REPLICATED   0x00000004
#define SYSTEM_FLAG_SCHEMA_BASE_OBJECT          0x00000010
#define SYSTEM_FLAG_DISALLOW_MOVE_ON_DELETE     0x02000000
#define SYSTEM_FLAG_DOMAIN_DISALLOW_MOVE        0x04000000
#define SYSTEM_FLAG_DOMAIN_DISALLOW_RENAME      0x08000000
#define SYSTEM_FLAG_CONFIG_ALLOW_LIMITED_MOVE   0x10000000
#define SYSTEM_FLAG_CONFIG_ALLOW_MOVE           0x20000000
#define SYSTEM_FLAG_CONFIG_ALLOW_RENAME         0x40000000
#define SYSTEM_FLAG_DISALLOW_DELTE              0x80000000
```

Figure 3.18. The bit-flags for the “systemFlags” attribute.

3.4.19. The attribute “msDs-Behaviour-Version”

In order to add new functions to Active-Directory from release to release, Microsoft uses so-called function-levels (see [MSTN06]). Domain-controller, domains and domain-forests have each an assigned function-level, which is saved within the attribute “msDs-Behaviour-Version”. Figure 3.19, “The possible values for the “msDs-Behaviour-Version” attribute.” shows all the possible values.

```
/* aus source/dsdb/common/flags.h unter [SaTe01] */
#define DS_BEHAVIOR_WIN2000             0
#define DS_BEHAVIOR_WIN2003_INTERIM     1
#define DS_BEHAVIOR_WIN2003             2
```

Figure 3.19. The possible values for the “msDs-Behaviour-Version” attribute.

3.5. Originating-Updates

In order to execute the *status-basedStore-and-Forward-Replication* used within Active-Directory, a set of meta data has to be maintained by the DSA. After all the necessary structures, terms and attributes have been explained, the treatment of meta data with *Originating-Updates* will be evaluated.

Generally, for *Originating-Updates*, changes are validated by the directory-schema.

If the domain-forest is within the function-level `DS_BEHAVIOR_2000`, the smallest item that has to be replicated is an attribute as well as all of its attribute-values. The function-levels `DS_BEHAVIOR_2003_INTERIM` and `DS_BEHAVIOR_2003` allow for the independent replication of individual attribute-values, if so-called *Linked-Attributes* are present. However, *Linked-Attributes* will not be further explained in this bachelor thesis.

3.5.1. Meta data storage with Originating-Add

At the moment of creation of a new directory-object, the attributes used for replication are automatically generated by the DSA. For the attribute “objectGUID”, a new unique GUID is generated. The attribute “instanceType” contains the value `INSTANCE_TYPE_WRITE`. Furthermore, the attributes “whenCreated” and “whenChanged” are initialised with the current timestamp. The attributes “uSNCreated” and “uSNChanged” are initialised with the USN assigned to the change. Also, an attribute based upon the RDN of the new object is automatically generated, for instance “dc” or “cn”. Additionally, the attribute “name” is added and set to the attribute-value of the RDN-based attribute.

The attribute “replPropertyMetaData” stores an array with meta data for each attribute that is to be replicated [9] (see Figure 3.14, “The IDL-description of the “replProperyMetaData” and “DsReplicaMetaDataCtr” fundamental structures.”). Thereby each *AttId* is set using the attribute-name and the item `version` is initialised with `1`. The current timestamp is stored within the item `originating_change_time`. The local DSA-Invocation-ID is stored within the item `originating_invocation_id`. The USN assigned to the change is written into the items `originating_usn` and `local_usn`.

3.5.2. Storage of meta data for Originating-Modify

When changing a directory object, attributes are generally replaced, that means at the moment of attribute deletion, it will be replaced by an attribute with no attribute-value. However, previously it is checked, whether the attribute-values are really altered by the change. If this is not the case, the attribute-modification is ignored. If all attribute-modifications are ignored, the entire change is ignored.

If the change is not ignored, then the attributes “whenChanged” and “uSNChanged” are set to the current time or USN.

The meta data within the attribute “replPropertyMetaData” are adjusted to the changed or added attributes. Here, the item `version` will be incremented or initialised with `1` for newly added attributes. The items `originating_change_time`, `originating_invocation_id`, `originating_usn` and `local_usn` are set analogously to the *Originating-Add*.

3.5.3. Storage of meta data for Originating-Move

[9] Some attributes are only relevant for the local DSA, furthermore, some attributes are constructed automatically upon request

Microsoft TechNet alleges in the paragraph “Originating Move” under [MSTN01], that moving a directory-object within a directory-partition is treated like an *Originating-Modify* of the attribute “name”. This behaviour could not be verified by own experiments.

3.5.4. Storage of meta data for Originating-Delete

Since deleted directory-objects must be propagated to all domain-controller via the *Store-and-Forward-Principle*, objects are not really deleted with *Originating-Delete*, but changed into so-called *Tombstones*.

The attribute “isDeleted” is here set to “TRUE”, which forces a search request to ignore the object. Additionally, all attributes that do not uniquely identify the object, nor are used for replication, are replaced by an attribute without attribute-value. The RDN-value is replaced by a value, that cannot be used for an *Originating-Add* or *Originating-Move*. The RDN-value is attached by the string “\nDel:”, followed by the string-representation of the Object-GUID. If the bit-flag `SYSTEM_FLAG_DISALLOW_MOVE_ON_DELETE` is set within the attribute “systemFlags”, the object is moved to the hidden container “CN=Deleted Objects”, underneath the root-object within the directory-partition. Figure 3.20, “An example for a *Tombstone-Object*.” shows an example for a *Tombstone* Object.

```
dn: CN=smbtorturedc2\0ADEL:b9cf25c3-ffc3-4dd8-9bbd-5b3776d204c0,CN=Deleted Ob
 jects,DC=sub1,DC=w2k3,DC=vmnet1,DC=vm,DC=base
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: user
objectClass: computer
cn:: c21idG9ydHVyZWRjMgpERUw6YjljZjI1YzMtZmZjMy00ZGQ4LTliYmQtNWIzNzc2ZDIwNGMw
distinguishedName: CN=smbtorturedc2\0ADEL:b9cf25c3-ffc3-4dd8-9bbd-5b3776d204c0
 ,CN=Deleted Objects,DC=sub1,DC=w2k3,DC=vmnet1,DC=vm,DC=base
instanceType: 4
whenCreated: 20070223103122.0Z
whenChanged: 20070223103315.0Z
uSNCreated: 3401
isDeleted: TRUE
uSNChanged: 3401
name:: c21idG9ydHVyZWRjMgpERUw6YjljZjI1YzMtZmZjMy00ZGQ4LTliYmQtNWIzNzc2ZDIwNGM
 w
objectGUID:: wyXPucP/2E2bvVs3dtIEwA==
userAccountControl: 4096
objectSid:: AQUAAAAAAAUVAAAAORXO/enDN5tvFTgaGwUAAA==
sAMAccountName: smbtorturedc2$
lastKnownParent: CN=Computers,DC=sub1,DC=w2k3,DC=vmnet1,DC=vm,DC=base
```

Figure 3.20. An example for a *Tombstone-Object*.

3.6. Replication of directory data

Within Active Directory, the directory partitions are replicated independently of each other. That means, the DSA maintaints for each partition a separate “replUpToDateVector”, “repsFrom” and “repsTo” attribute (see Figure 3.8, “The IDL-description of the *Up-To-Date-Vector* fundamental structures.” and Figure 3.10, “The IDL-description of the “repsFromToBlob” structure.”). The attribute “repsFrom” contains information on all Source-DSA used by the DSA, whereby the attribute values are created automatically by so-called *Knowledge Consistency Checker* (KCC) at the moment of creation of the *Replication topology*. The attribute “repsFrom” containts Information on all Destination-DSA, that shall be notified about changes by the local DSA. More information on this topic is given in [MSTN01] and [MSTN02].

3.6.1. The function `DSGetNCChanges()`

The function `DsGetNCChanges()` implements the *Pull-Replication*. Oversimplified, this function expects a `struct drsuapi_DsGetNCChangesRequest8` (see Figure 3.21, “The IDL description of the “DsGetNCChangesRequest8” structure.”) as parameter and returns a `struct drsuapi_DsGetNCChangesCtr6` (see Figure 3.22, “The IDL description of the “DsGetNCChangesRequest8” structure.”). Whereby under Windows 2000 `struct drsuapi_DsGetNCChangesRequest5` and `struct drsuapi_DsGetNCChangesCtr1` were used, which each form a subset of the newer versions, and thus do not have to be analysed separately.

```
/* aus drsuapi.idl unter [SaTe03] bzw. Anhang A2 ab Zeile 402
typedef struct {
    GUID destination_dsa_guid;
    GUID source_dsa_invocation_id; /* the 'invocationId' field of the CN=NTDS Settings object */
    [ref] drsuapi_DsReplicaObjectIdentifier *naming_context;
    drsuapi_DsReplicaHighWaterMark highwatermark;
    drsuapi_DsReplicaCursorCtrEx *uptodateness_vector;
    drsuapi_DsReplicaNeighbourFlags replica_flags;
    uint32 max_object_count; /* w2k3 uses min(133,max(100,max_object_count)) */
    uint32 max_ndr_size; /* w2k3 seems to ignore this */
    uint32 unknown4; /* = 0, unknown */
    hyper h1; /* = 0 unknown */
    uint32 unique_ptr1; /* = 0, unknown */
    uint32 unique_ptr2; /* = 0, unknown */
    drsuapi_DsReplicaOIDMapping_Ctr mapping_ctr;
} drsuapi_DsGetNCChangesRequest8;
```

Figure 3.21. The IDL description of the “DsGetNCChangesRequest8” structure.

```
/* aus drsuapi.idl unter [SaTe03] bzw. Anhang A2 ab Zeile 519
 * if the DRSUAPI_DS_LINKED_ATTRIBUTE_FLAG_ACTIVE flag
 * isn't there it means the value is deleted
 */
typedef [public] bitmap {
    DRSUAPI_DS_LINKED_ATTRIBUTE_FLAG_ACTIVE = 0x00000001
} drsuapi_DsLinkedAttributeFlags;

typedef [public] struct {
    drsuapi_DsReplicaObjectIdentifier *identifier;
    drsuapi_DsAttributeId attid;
    drsuapi_DsAttributeValue value;
    drsuapi_DsLinkedAttributeFlags flags;
    NTTIME_1sec originating_add_time;
    drsuapi_DsReplicaMetaData meta_data;
} drsuapi_DsReplicaLinkedAttribute;

typedef [public,gensize] struct {
    GUID source_dsa_guid; /* the 'objectGUID' field of the CN=NTDS Settings object */
    GUID source_dsa_invocation_id; /* the 'invocationId' field of the CN=NTDS Settings object */
    drsuapi_DsReplicaObjectIdentifier *naming_context;
    drsuapi_DsReplicaHighWaterMark old_highwatermark;
    drsuapi_DsReplicaHighWaterMark new_highwatermark;
    drsuapi_DsReplicaCursor2CtrEx *uptodateness_vector;
    drsuapi_DsReplicaOIDMapping_Ctr mapping_ctr;
    uint32 total_object_count; /* only w2k sets it, but w2k3 gives 0 */
    uint32 object_count;
    /* this +55 is sometimes +56, so I don't know where this comes from... --metze */
    [value(ndr_size_drsuapi_DsGetNCChangesCtr6(r,ndr->flags)+55)] uint32 __ndr_size;
    drsuapi_DsReplicaObjectListItemEx *first_object;
    uint32 unknown4; /* = 0, unknown */
    uint32 unknown5; /* = 0, unknown */
    uint32 unknown6; /* = 0, unknown */
    [range(0,1048576)] uint32 linked_attributes_count;
    [size_is(linked_attributes_count)] drsuapi_DsReplicaLinkedAttribute *linked_attributes; /* array! */
    uint32 unknown7; /* = 0, unknown */
} drsuapi_DsGetNCChangesCtr6;
```

Figure 3.22. The IDL description of the “DsGetNCChangesRequest8” structure.

When fulfilling the “DsGetNCChangesRequest8” structure, the following has to be taken into account:

- The item `naming_context` has to be initialised with the DN of the directory partition.
- In order to also receive the attributes along with the passwords, the `DRSUAPI_DS_REPLICA_NEIGHBOUR_WRITEABLE` bit flag within the `replica_flags` item has to be set.
- The item `max_object_count` indicates, how many objects the Source DSA should return at once. Here, Windows 2003 uses a value of `133`.
- The item`max_ndr_size`is not of real importance, here the value `1336811` is being sent.
- The item `mapping_ctr`as well as every other (until now) unknown items are initialised with `0` or rather `NULL`.
- The Destination DSA should set all already known items to their correct values or to `0`.

The return values within the “DsGetNCChangesCTR6” structure give the following information:

- The item `source_dsa_guid` containts the DSA Guid of the Source DSA.
- The item `source_dsa_invocation_id` contains the current DSA invocation guid of the source DSA. It does not have to be requested by the Destination DSA.
- The item `old_highwatermark` is a copy of the *High-Water-Mark* sent by the Destina-

tion DSA. And thus is not necessarily related to the value of `source_dsa_invocation_id`.

- The item `new_highwatermark` is related to the value of `source_dsa_invocation_id`. The USN in `new_highwatermark.tmp_highest_usn` contains the highest USN of the objects for the current invocation of `DsGetNCChange()`. If the USN in `new_highwatermark.highest_usn` has the same value, then the Source DSA does not have further changes for the Destination DSA. But if the value of `new_highwatermark.highest_usn` is smaller than the one of `new_highwatermark.tmp_highest_usn`, then the Destination DSA must request further changes through an additional invocation of `DsGetNCChanges()`.
- The item `uptodateness_vector` is only returned by the Source DSA, if there are no further changes from the Source DSA for the Destination DSA (compare item `new_highwatermark`).
- The item `mapping_ctr` is of significant importance for the replication, it containts the *Prefix-Mappings* used within the directory schema. Besides, in the last array item there is no *Prefix-Mapping* given, but only the value of the attribute "schemaInfo", the root object of the schema partition. Except when replicating the schema partition, the Destination DSA has to compare the *Prefix-Mappings* and the attribute "schemaInfo" with its own values. If those do not coincide, then the Destination DSA has to discard the complete result of the current invocation of `DsGetNCChanges()` and primarily replicate the schema partition.
- The item `total_object_count` is set to "0" in Windows 2003. Windows 2000 gives the number of all objects in the replicated directory partition.
- The item `object_count` contains the number of objects contained within the current invocation of `DsGetNCChanges`.
- The item `first_object` is the head of a simply chained list of `struct drsuapi_DsReplicaObjectListItemEx` items. Each item contains a directory object including the attributes and their meta data that are to be replicated If the `DRSUAPI_DS_REPLICA_NEIGHBOUR_RETURN_PARENT_OBJECTS` bit flag was set by the Destination DSA in the item `replica_flags`, each list item contains also the GUID of its respective parent object. Specifically how the individual list items should be interpreted, is explained in the next paragraph.
- The item `linked_attributes_count` contains the number of replicated *Linked-Attributes*.
- The item `linked_attributes` is an array consisting of `struct drsuapi_DsReplicaLinkedAttribute` items. *Linked-Attributes* are not explained further in this bachelor thesis. The basic principle of the replication of *Linked-Attributes*, is given in [MSTN01].

The `DRSUAPI_DS_REPLICA_NEIGHBOUR_COMPRESS_CHANGES` bit flag can be set by the Destination DSA in the `replica_flags` item, so that the Source DSA compresses the entire "DsGetNCChangesCtr6" structure. The used algorithms are called either "MSZIP" or "XPRESS". "MSZIP" is based on the "Deflate" algorithm [RFC1951]. The so called "XPRESS" algorithm is as of this date unknown. Since Samba manages the "MSZIP" decompression transparently for the programmer, it will not be explained further. The implementation can be found in source/librpc/ndr/ndr_compression.c and source/lib/compression/mszip.c under [SaTe01].

3.6.2. Interpretation of replicated directory objects

For the interpretation of the replicated directory objects, the directory schema together with the *Prefix-Mappings* is strongly needed. `struct drsuapi_DsReplicaObjectListItemEx *li` depicts an individual directory object

that needs to be replicated (see Figure 3.21, “The IDL description of the “DsGetNCChangesRequest8” structure.” and Figure 3.16, “The IDL-description of the “DsReplicaObjectListItem” and “DsReplicaObjectListItemEx” list items.”). Which logical steps have to be taken for the interpretation, will be explained below:

- The Object-DN is available as string through `li->object.identifier->dn`.
- From the Object-DN, the on the RDN based attribute is then constructed (also called RDN-attribute).
- The Object-GUID is through `li->object.identifier->guid` available as `struct GUID`, from it the attribute “objectGUID” has to be constructed with the NDR-coded GUID.
- The attribute “when Changed” will be constructed from the highest of all the `li->meta_data_ctr->meta_data[i].originating_change_time` values.
- Some attributes contain security-critical password information, for example “unicodePwd”, “dBCSPwd” or “supplementalCredentials”. The attribute values of those attribute will be encrypted specifically for each session. Before the attribute values can be used, the encryption has to be revoked. I have found the used method by systematic trial and error with known algorithms. Here, the following algorithms are used:
  - *Message Digest* (MD5) [RFC1321], a cryptographic *Hash-Function* (also see source/lib/crypto/md5.c under [SaTe01]).
  - *Cyclic Redundancy Check 32* (CRC32) [RFC1952] (see paragraph “8. Appendix: Sample CRC Code”), a *checksum-function*. (also see source/lib/crypto/md5.c under [SaTe01]).
  - ARCFOUR[10][WikiPe03],[GoGr01], a *stream ciphering function*. (also see source/lib/crypto/md5.c under [SaTe01]).

  In the Kerberos- or NTLMSSP-authentication on DCERPC-level, a session key is negotiated, that flows into the encryption. The session-specific encryption is done as follows:
  - A 16 byte random number is generated, this is called *Confounder*.
  - From the session key, followed by the *Confounder* , an encryption key is generated with the aid of the MD5-Hash-Function.
  - The password data, with prefixed CRC32-checksum, are encrypted using the encryption key and the ARCFOUR-Function.
  - For the transmission, the *Confounder* is placed in front of the encrypted data.

  Figure 3.23, “The session-specific encryption of password information.” clarifies this procedure. (see `dsdb_decrypt_attribute_value()` in source/dsdb/repl/replicated_objects.c under [SaTe01] or Appendix A12 starting at row 33).
- Successively, all attributes `li->object.attribute_ctr.attributes[i]` are converted in a loop. Here, the 32-bit *AttId* is converted with the aid of the *Prefix-Mapping* and the loaded Schema-Cache into the Attribute-Name. Using the Attribute-Definition in the Schema-Cache, then the so-called *Attribute-Syntax* will be determined, for instance “Boolean”, “Integer”, “String(UNICODE” or “Object(DS-DN)”. Each *Attribute-Syntax* has an assigned conversion-function, which converts the transmitted Attribute-Values into the internally used representation. Samba stores Attribute-Values, as they are used within the LDAP-Protocol. Integer-Values are for instance stored in the String-Representation. Each possible *Attribute-Syntaxes* and their respective conversion-functions are implemented under source/dsdb/schema/schema_syntax.c under [SaTe01] or in appendix A11. Starting from row 1113, the *Attribute-Syntaxes* are defined as array items from

[10]The name “RC4” is a trademark of RSA Data Security, Inc. That is why the name “ARCFOUR” is used more commonly.

`struct dsdb_syntax` (see source/dsdb/schema/schema.h under [SaTe01] or appendix A4 starting from row 30).

- The meta data array has to be converted from the `struct drsuapi_DsReplicaMetaData` items into an array of `struct replPropertyMetaData1` elements. Here, another `struct replPropertyMetaData1` item is attached for the RDN-Attribute, the meta data is copied from the attribute “name”. Then, the meta data is saved within the attribute “replPropertyMetaData”. Also see Figure 3.14, “The IDL-description of the “replProperyMetaData” and “DsReplicaMetaDataCtr” fundamental structures.”.

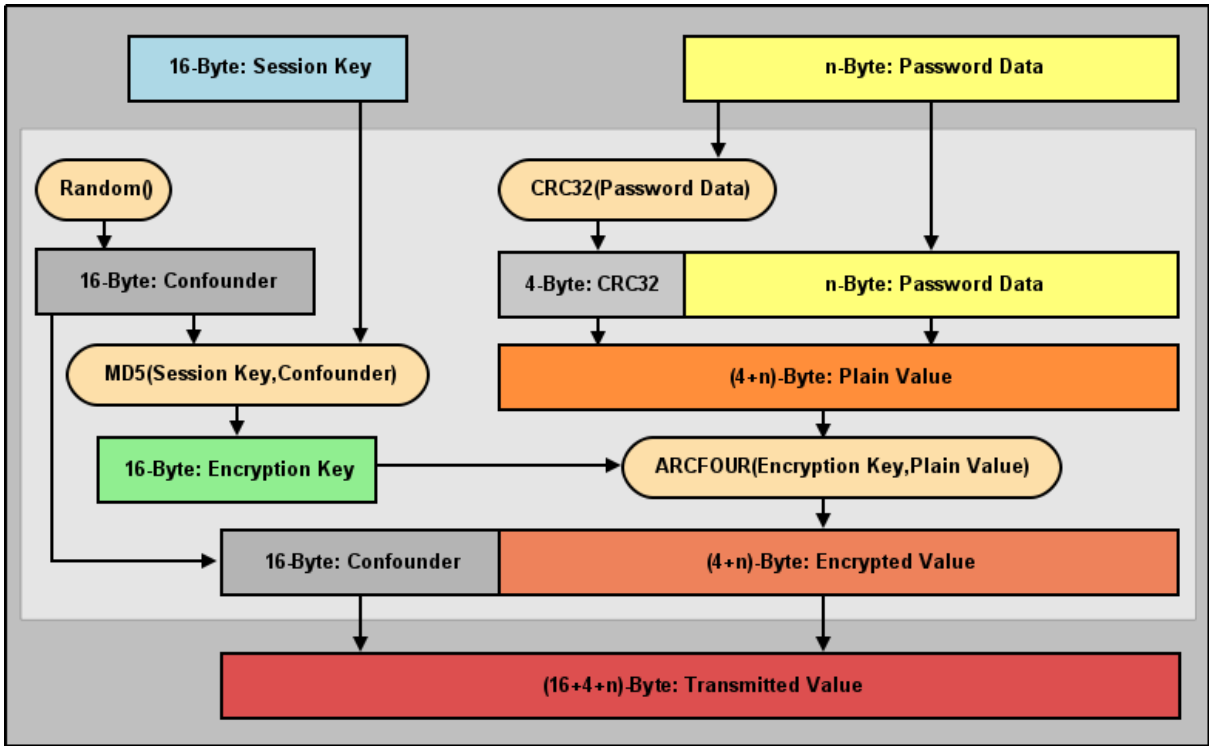

Figure 3.23. The session-specific encryption of password information.

3.6.3. Methods for the filtering of data that is to be replicated

The Source-DSA filters the data that is sent to the Destination-DSA. By this, only the changes are transmitted that the Destination-DSA did not already replicate from another Source-Dsa.

Filtering using the High-Water-Mark

During a replication request, the Source-DSA has to decide, using the *High-Water-Mark*, which directory objects have to be send to the Destination-DSA. For this, it simply lists all the objects, whose “uSNChanged” value is higher than the `highest_usn` value within the *High-Water-Mark*. Thereby, a maximum of 133 objects, sorted in ascending order of “uSNChanged” values, are selected for transmission to the Destination-DSA. The use of the *High-Water-Mark* makes sure, that changes are transmitted only once between a pair of Source-DSA and Destination-DSA.

Filtering using the Up-To-Date-Vector

Before the Source-DSA transmits the by the *High-Water-Mark* generated object list to the Destination-DSA, it is checked, whether the data, that has to be replicated, can be further minimised. The *Up-To-Date-Vector* sent by the Destination-DSA contains a list of all to the Destination-DSA known DSA-Invocation-Ids with their respective highest known USN. That is why, for each attribute it is checked using its meta data (`originating_invocation_id` and `originating_usn`) as well as the *Up-To-Date-Vector*, whether the attribute is already contained within the Destination-DSA. If this examination results in that the attribute does not have to be transmitted, it will be removed from the object. If there are no attributes that have to be transmitted left within an object, then the entire object will be removed from the list. Eventually, another object succeeds it.

3.6.4. Algorithm for the conflict resolution of Multimaster-Replication

When bringing in *Replicating-Updates*, there generally can be conflicts in the *Multimaster-Replication*. Those are resolved in Active-Directory through an algorithm, which without fail returns the same result for each DSA. Along the lines of: “The last write access wins”.

Conflict on Attribute-Level

Before an attribute including all of its Attribute-Values is replaced by a *Replicating-Update*, the old and new meta data (`struct replPropertyMetaData1`) is compared. Primarily, the `version` item is compared. If it is the same, the `originating_change_time` item is compared. In the statistically almost impossible case, that the same attribute is changed on two DSAs within the same second, the `originating_invocation_id` item is compared, which will always resolve the conflict definitively. If the replicated attribute has the “higher” meta data, the old attribute will be replaced, otherwise the replicated attribute will be discarded.

Conflicts on DN-Level

Using *Multimaster-Replication*, it can also happen, that objects are created, whose parent objects have been deleted on another DSA. According to [MSTN01] (paragraph “Replication Conflict Resolution”), those objects are moved to the “CN=Lost And Found” Container underneath the root-object of the directory-partition. However, this aspect has not been explicitly examined by the Samba-Team.

Using *Multimaster-Replication*, it can also happen, that objects with the same Object-DN are created on two different DSAs. According to [MSTN01] (paragraph “Replication Conflict Resolution”), the RDN (effectively the attribute “name”) of the two objects, undergo the comparison-algorithm that is valid for attribute-conflicts, whereby the object with the “higher” meta data may keep its Object-DN. For the object with the “lower” meta data, the String “*CNF:” is attached to the RDN, followed by the String-Representation of the Object-GUID. For example “CN=NewUser, CN=Users, ...” would become “CN=NewUser*CNF:077157f1-eecc-47bf-ba03-9753c93e2e73, CN=Users, ...” However, this aspect has not been explicitly examined by the Samba-Team.

3.6.5. Change-Notify via DsReplicaSync()

It is desirable, to propagate changes with the lowest possible latency. In Active-Directory, with the aid of the function `DsReplicaSync()`, a mechanism that is called *Change-Notify* is implemented. For each *Originating-Update*, the DSA sets a timer of 15 seconds by default. After that, all to the DSA known Destination-DSAs are notified about the change in intervals of 3 seconds. The used function `DsReplicaSync()` expects a `struct drsuapi_DsReplicaSyncRequest1` as parameter (see Figure 3.24, "The IDL description of the "DsReplicaSyncRequest1" structure."). The Source-DSA transfers the DN to the directory-partition and its DSA-GUID, so that the Destination-DSA knows, for which Source-DSA it shall replicate which directory-partition. By default, the item `other_info` is initialised with `NULL` and the item `options` with `0`.

```
/* aus drsuapi.idl unter [SaTe03] bzw. Anhang A2 ab Zeile 136 */
typedef [public] bitmap {
    DRSUAPI_DS_REPLICA_SYNC_ASYNCHRONOUS_OPERATION  = 0x00000001,
    DRSUAPI_DS_REPLICA_SYNC_WRITEABLE               = 0x00000002,
    DRSUAPI_DS_REPLICA_SYNC_PERIODIC                = 0x00000004,
    DRSUAPI_DS_REPLICA_SYNC_INTERSITE_MESSAGING     = 0x00000008,
    DRSUAPI_DS_REPLICA_SYNC_ALL_SOURCES             = 0x00000010,
    DRSUAPI_DS_REPLICA_SYNC_FULL                    = 0x00000020,
    DRSUAPI_DS_REPLICA_SYNC_URGENT                  = 0x00000040,
    DRSUAPI_DS_REPLICA_SYNC_NO_DISCARD              = 0x00000080,
    DRSUAPI_DS_REPLICA_SYNC_FORCE                   = 0x00000100,
    DRSUAPI_DS_REPLICA_SYNC_ADD_REFERENCE           = 0x00000200,
    DRSUAPI_DS_REPLICA_SYNC_NEVER_COMPLETED         = 0x00000400,
    DRSUAPI_DS_REPLICA_SYNC_TWO_WAY                 = 0x00000800,
    DRSUAPI_DS_REPLICA_SYNC_NEVER_NOTIFY            = 0x00001000,
    DRSUAPI_DS_REPLICA_SYNC_INITIAL                 = 0x00002000,
    DRSUAPI_DS_REPLICA_SYNC_USE_COMPRESSION         = 0x00004000,
    DRSUAPI_DS_REPLICA_SYNC_ABANDONED               = 0x00008000,
    DRSUAPI_DS_REPLICA_SYNC_INITIAL_IN_PROGRESS     = 0x00010000,
    DRSUAPI_DS_REPLICA_SYNC_PARTIAL_ATTRIBUTE_SET   = 0x00020000,
    DRSUAPI_DS_REPLICA_SYNC_REQUEUE                 = 0x00040000,
    DRSUAPI_DS_REPLICA_SYNC_NOTIFICATION            = 0x00080000,
    DRSUAPI_DS_REPLICA_SYNC_ASYNCHRONOUS_REPLICA    = 0x00100000,
    DRSUAPI_DS_REPLICA_SYNC_CRITICAL                = 0x00200000,
    DRSUAPI_DS_REPLICA_SYNC_FULL_IN_PROGRESS        = 0x00400000,
    DRSUAPI_DS_REPLICA_SYNC_PREEMPTED               = 0x00800000
} drsuapi_DsReplicaSyncOptions;

typedef struct {
    drsuapi_DsReplicaObjectIdentifier *naming_context;
    GUID source_dsa_guid;
    astring *other_info; /* I assume this is related to the repsFromTo1OtherInfo dns_name */
    drsuapi_DsReplicaSyncOptions options;
} drsuapi_DsReplicaSyncRequest1;
```

Figure 3.24. The IDL description of the "DsReplicaSyncRequest1" structure.

3.6.6. Periodic pull-replication

Additionally to the *Change-Notify*, each DSA executes by default every 15 minutes a *Pull-Replication* with each known Source-DSA.

Chapter 4. Implementation of some aspects, based on Samba 4.0

In this chapter, the logical steps and relations of the partial aspects that are to be implemented shall be roughly described. Because of the about 10 thousand source code rows, the explanation of details would exceed the scope of this bachelor thesis by far. That is why direct references are made to the functions in the source code (see appendix).

4.1. Implementation concepts within Samba 4.0

4.1.1. Programming languages

The programming language “C”

The main part of Samba is written in the programming language “C”. Part of this are a set of client libraries, that among other things, are relevant to all Active Directory protocols. Furthermore, the server program `smdb` and many small tools, that execute several different tasks, are written in C. The testing program `smbtorture` constitutes a big part, with it, protocol aspects as well as internal interfaces are tested.

The programming language “EJS”

Another part of Samba is written in the language “JavaScript”, this seems untypically at first view, since this language is more commonly known within the area of client-side web-applications. The reason for its use within Samba is the built-in web server, that offers an administration interface. Here, server sides, the variant “embedded javascript” (EJS) and client sides the standard “javascript” is being used. For this, the EJS interpreter from the appweb project (see [AppWeb01]) was integrated.

An in-between layer permits the invocation of C functions from EJS programs. Furthermore, the program `smbscript` provides the possibility for command line scripts, similar to Perl or Python. Conversely, the invocation of EJS programs from within C source code is possible as well.

The descriptive language “IDL”

The definition of the DCERPC interfaces has been realised in the language “IDL”. From that, Sambas IDL compiler “pidl” creates C functions, that execute the NDR coding and decoding of the C structures. Furthermore, DCERPC client functions for “C” and “EJS” are being generated. Through the automatic generation of C source code, much error-prone manual work is saved (momentarily about 280 thousand rows).

4.1.2. Operating systems

Samba may be used on a number of Unix operating systems. Examples are AIX, FreeBSD, HPUX, IRIX, Linux, MacOS, NetBSD, OpenBSD, Solaris and Tru64.

It is of importance, that the operating system supports the function `select()` (see [Man01]) or `epoll()` (see [Man02]). With it, it is possible to monitor a number of file- and socket descriptors for events. This function is then the only one, that can put the Unix process “asleep”. Thereupon, all I/O operations will be always executed in an unblocking manner. This is called *non blocking I/O*.

4.1.3. TALLOC the “tree allocator”

Samba does not use the functions `malloc()`, `strdup()` and `free()`. Samba uses a hierarchical storage management system called “TALLOC” (see source/lib/talloc/talloc_guide.txt under [SaTe01]). It supports storage references and destructors. Furthermore, type security is given. Figure 4.1, “An exemplary use of “TALLOC”.” shows an exemplary use.

```
#include <stdio.h>
#include <talloc.h>

struct foo {
    char *name;
};

static int foo_destructor(struct foo *f)
{
    printf("destructor called\n");
}

int main(void)
{
    void *top;
    void *child;
    struct foo *f;

    /* create the toplevel element */
    top = talloc_new(NULL);

    /* create child of the toplevel element */
    child = talloc_new(top);

    /* create a struct foo as child of 'child' */
    f = talloc(child, struct foo);

    /* allocate the name as child of 'f' */
    f->name = talloc_strdup(f, "foo-name");

    /* set the destructor for 'f' */
    talloc_set_destructor(f, foo_destructor);

    /* steal 'f' into the 'top' context */
    talloc_steal(top, f);

    /* an explicit free on 'child' only frees 'child' */
    talloc_free(child);

    /* an explicit free on 'top' also frees 'f' and 'f->name' */
    printf("before destructor\n");
    talloc_free(top);
    printf("after destructor\n");

    return 0;
}
```

Figure 4.1. An exemplary use of “TALLOC”.

4.1.4. The subsystem “EVENTS”

Samba is generally dimensioned for asynchronous work and *non blocking I/O*. For this, the abstraction layer *EVENTS* (see source/lib/events/events.h under [SaTe01]) was designed, which hides operating system specific details. Generally, the following events may be monitored:

- `fd_event`: File and socket descriptors, that become read- or writable.
- `timed_event`: Moments, that elapse.
- `aio_event`: asynchronous file system accesses (within Linux).
- `signal_event`: The appearance of Unix signals (their use should be prevented if possible).

Figure 4.2, “An exemplary use of functions of the subsystem “EVENTS”.” shows an exemplary use.

```
#include <stdio.h>
#include <stdint.h>
#include <sys/time.h>
#include <talloc.h>
#include <events.h>

static void stdin_read_handler(struct event_context *ev_ctx, struct fd_event *fde,
                               uint16_t flags, void *private_data)
{
    char c = '\0';
    /* read 1 byte from stdin */
    read(0, &c, 1);
    /* print the byte to stdout */
    printf("%c", c);
}

static void timeout_handler(struct event_context *ev, struct timed_event *te,
                            struct timeval tval, void *private_data)
{
    struct fd_event *fde = talloc_get_type(private_data, struct fd_event); /* typesafe cast */
    /* remove the fd_event, which means event_loop_wait() will return */
    talloc_free(fde);
    printf("timeout\n");
}

/* over 15 seconds all input bytes on stdin are copied to stdout */
int main(void)
{
    struct event_context *ev;
    struct fd_event *fde;
    struct timeval tval;

    /* create the event context (is also a valid talloc context) */
    ev = event_context_init(NULL);

    /* add a fd_event and wait for data in the readable from the pipe */
    fde = event_add_fd(ev, (TALLOC_CTX *)ev,
                       0/* stdin */, EVENT_FD_READ,
                       stdin_read_handler, /* callback function */
                       NULL); /* private pointer */

    /* add a timed_event that will be triggered 15 seconds later */
    gettimeofday(&tval, NULL);
    tval.tv_sec += 15;
    event_add_timed(ev, (TALLOC_CTX *)ev, tval, /* timeout */
                    timeout_handler, /* callback function */
                    fde); /* private pointer */

    /* wait for events, until all event handlers are remove from the event_context */
    event_loop_wait(ev);

    talloc_free(ev);
    return 0;
}
```

Figure 4.2. An exemplary use of functions of the subsystem “EVENTS”.

4.1.5. Asynchronous function calls

Generally, all functions, that communicate via *sockets* or *pipes* to other processes, are programmed asynchronously within Samba. Here, there always exist a C function (for example `foo_send()`), that "sends" function parameters and a C function (for example `foo_recv()`), that "receives" the return values. In doing so, the sending function expects among other things a `struct event_context` as parameter and returns a *request handle*. If the calling function wants to await the incoming return value actively, it calls the receiving function and transmits to it the *request handle*. For simplified use within testing programs this is often realised by the means of a synchronous C-function (for example `foo()`). If the calling function does not want to wait actively for the incoming return value, it can deposit a *callback function* in the *request handle* in turn terminate itself. Then, the main program typically procedes with the processing of other events within the function `event_loop_wait()` or puts the process "asleep" until new events arrive. If sometime later the return values of the function are available, the *callback function* that was deposited within the *request handle* will be called. The *callback function* in turn has to call the receiving function, in order to acquire the return values. Through the invocation of the receiving function, the *request handle* is cleared again. Figure 4.3, "An example of an asynchronously called function." shows an example.

```
#include <stdio.h>
#include <stdint.h>
#include <sys/time.h>
#include <talloc.h>
#include <events.h>

struct foo_request {
    void (*async_fn)(struct foo_request *); /* function pointer for callback function */
    void *async_private;                    /* private data for the callback function */
};

static void _foo_timed_handler(struct event_context *ev, struct timed_event *te,
                               struct timeval tval, void *private_data)
{
    struct foo_request *req = talloc_get_type(private_data, struct foo_request);
    if (req->async_fn) req->async_fn(req); /* notify the caller by calling the callback function */
}

static struct foo_request *foo_send(struct event_context *ev, int delay)
{
    struct foo_request *req = talloc_zero(ev, struct foo_request);
    struct timeval tval;
    gettimeofday(&tval, NULL);
    tval.tv_sec += delay;   /* wait a bit before giving a result to the caller */
    event_add_timed(ev, req, tval, _foo_timed_handler, req);
    return req;
}

static int foo_recv(struct foo_request *req)
{
    if (!req) return -1;    /* error */
    talloc_free(req);       /* free the request handle */
    return 0;               /* no error */
}

static void setup_task_end(struct foo_request *req)
{
    struct fd_event *fde = req->async_private;
    int ret = foo_recv(req);        /* function foo() finished */
    printf("foo() returned %d\n", ret);
    talloc_free(fde); /* remove dummy fd_event to exit the event_loop_wait() */
}

static void setup_task_start(struct event_context *ev, struct timed_event *te,
                  struct timeval tval, void *private_data)
{
    struct foo_request *req;
    req = foo_send(ev, 5);                  /* function foo() started */
    req->async_fn           = setup_task_end; /* function foo() will be finished in setup_task_end() */
    req->async_private      = private_data;   /* pass the private_data to setup_task_end() */
}

int main(void)
{
    struct timeval tval = { .tv_sec = 0, .tv_usec = 0 }; /* directly trigger the event */
    struct event_context *ev = event_context_init(NULL);
    struct fd_event *fde;
    fde = event_add_fd(ev, ev, 0, 0, NULL, NULL); /* event_loop_wait needs a dummy fd_event */
    event_add_timed(ev, ev, tval, setup_task_start, fde);
    event_loop_wait(ev);
    talloc_free(ev);
    return 0;
}
```

Figure 4.3. An example of an asynchronously called function.

4.1.6. TDB, the “trivial database”

The *trivial database* (TDB) used within samba, provides a very efficient *hash table*, based on *shared memory* for the storage of *key value pairs*. TDB supports *multitasking* and *transactions* (see source/lib/tdb/docs/README under [SaTe01]).

4.1.7. LDB, the “light weight database”

The *lightweight database* (LDB) [SaTe05] used within Samba, provides the following functions:

- The programming interface (LDB-API) and the data model are very similar to LDAP.
- The data is either stored within a TDB file or in a real LDAP server. Furthermore, an experimental “sqlite3” LDB backend exists.
- When using a TDB file, no separate server process is necessary. Here, the often complicated setup of a LDAP server can be skipped. Furthermore, access times are optimised.
- The LDB-API can also be used as an alternative to common LDAP-API.
- By default, no schema is used.
- Very simple administration of indexed attributes.
- Through LDB modules, additional functions can be implemented. For example support of a directory schema.
- Easy to manage administration programs exist.
- Import and export in the LDIF format is possible.
- LDB controls and LDB extended operations are supported analogously to *LDAP controls* and *LDAP extended operations* (see also [RFC4510]).

4.2. The subsystem “DSDB” (or “SAMDB”)

Within Samba, the C functions, necessary for the implementation of the directory service, are gathered within the subsystem *Directory Service Database* (DSDB). In some instances, the term *Security Account Manager Database* (SAMDB) is used indifferently. However, preference should gradually be given to the DSDB.

When implementing a directory service, several layers that are built on top of each other, overtake the task of storing the directory objects in a secondary storage means.

4.2.1. DSA operational layer

The *DSA operational layer* takes care of the semantics of the functions that are provided by the directory service, these are “Add”, “Modify”, “Delete” and “Search”. That means, write accesses will be validated by the directory schema and the meta data, necessary for replication, maintained. Furthermore, authorizations are analysed for write- and read accesses. The *DSA operational layer* is independent to the networking protocol used for the access. The server components of the respective networking protocols, for instance LDAP, DRSUAPI, SAMR and NETLOGON, are based on the *DSA operational layer*. Figure 4.4, “Layer comparison between Windows 2003 and Samba 4.0 (compare “Replication Subsystem Components” in [MSTN01]).” shows an overview of the used layers.

According to [MSTN01] (in the paragraph “Replication Subsystem”), the *DSA operational layer* is implemented in Windows 2003 within the file `ntdsa.dll`. Here, Samba uses a set of LDB modules (see Figure 4.4, “Layer comparison between Windows 2003 and Samba 4.0 (compare “Replication Subsystem Components” in [MSTN01]).”).

4.2.2. Database layer

The *Database layer* takes care of the way that directory objects are stored. Furthermore attribute indices are managed within this layer.

According to [MSTN01] (in the paragraph “Replication Subsystem”), the *database layer* is implemented in Windows 2003 within the file `ntdsa.dll`. In Samba, the tasks pertaining to the *database layer* are executed within the LDB BACKEND “ldb_tdb” (see Figure 4.4, “Layer comparison between Windows 2003 and Samba 4.0 (compare “Replication Subsystem Components” in [MSTN01]).”).

4.2.3. Storage layer

The *storage layer* takes care of the way that the data is finally managed on the secondary storage means. The implementation of transactions is also a part of this.

According to [MSTN01] (in the paragraph “Replication Subsystem”), the *storage layer* is implemented in Windows 2003 within the file `Esent.dll`. The data is stored within the file system in the file “Ntds.dit”. In Samba, the tasks pertaining to the *storage layer* are executed by TDB. In the file system, the data is stored within the files `samdb.ldb`, `domain.ldb`, `config.ldb` and `schema.ldb` (see Figure 4.4, “Layer comparison between Windows 2003 and Samba 4.0 (compare “Replication Subsystem Components” in [MSTN01]).”).

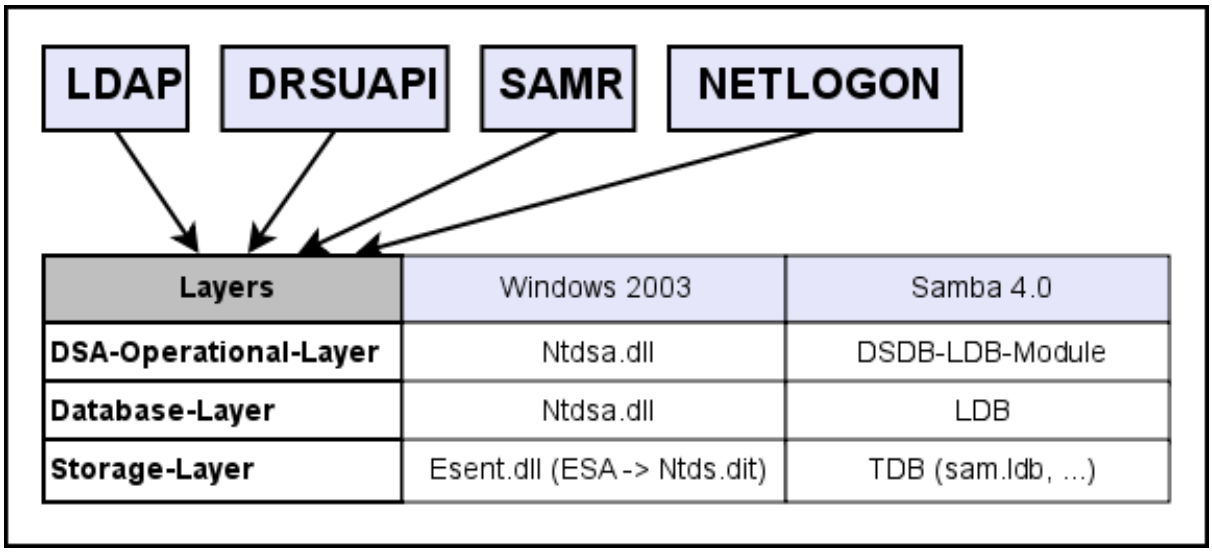

Figure 4.4. Layer comparison between Windows 2003 and Samba 4.0 (compare “Replication Subsystem Components” in [MSTN01]).

4.2.4. DSDB LDB modules

In Samba, the LDB API represents the protocol independent interface of the *DSA operational layer*. The tasks pertaining to the *DSA operational layer* are are acquired by LDB modules.In the following Figure 4.5, “Overview over LDB modules, that are used in the subsystem “DSDB”.”, the interaction of the LDB modules is shown. Subsequently, the respective tasks of the LDB modules are shortly described.

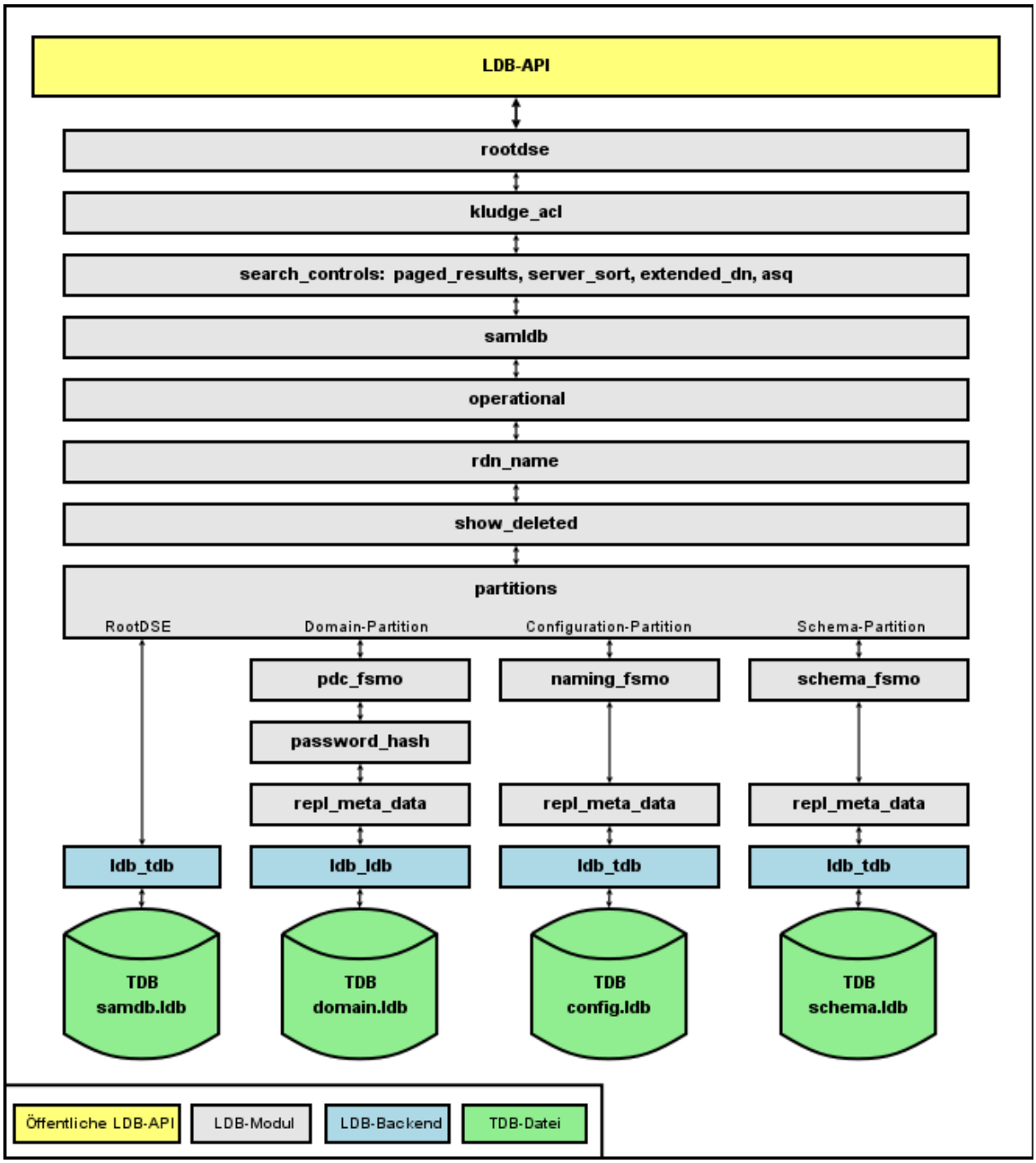

Figure 4.5. Overview over LDB modules, that are used in the subsystem “DSDB”.

LDB module “rootdse”

The LDB module “rootdse” implements the characteristics of the RootDSE object (see source/dsdb/samdb/ldb_modules/rootdse.c under [SaTe01]). For example, the attributes “highestCommitedUsn” and “currentTime” give current values for each search request.

LDB module “kludge_acl”

The LDB module “kludge_acl” gives a trivial implementation for the enforcement of authorizations (see source/dsdb/samdb/ldb_modules/kludge_acl.c under [SaTe01]). Here, adminis-

trators may read and write to all objects and attributes. All other users may read all objects and almost all attributes, only attributes with password information are restrained. At a later point, this LDB module shall be replaced by a module with far greater function coverage.

LDB module “paged_results”, “server_sort”, “extended_dn” and “asq”

The LDB modules “paged_results”, “server_sort”, “extended_dn” and “asq” each implement a *LDAP control* (see [RFC4510]). Since they do not have big importance for the replication, they will here not be further explained (see source/dsdb/samdb/ldb_modules/ and lib/ldb/modules/ under [SaTe01]).

LDB module “samldb”

The LDB module “samldb” takes care of the assignment of a SID to user- and group objects for the *originating add*, furthermore some attributes are initialised with standard values (see source/dsdb/samdb/ldb_modules/samldb.c under [SaTe01]).

LDB module “operational”

The LDB module “operational” constructs constructed attributes upon search requests, such as “createTimestamp”, “modifyTimestamp”, “structuralObjectClass” and “canonicalName” (see source/lib/ldb/modules/operational.c under [SaTe01]). On a later point, this LDB module must be extended by use of the schema API.

LDB module“rdn_name”

The LDB module “rdn_name” enforces, that the RDN attribute and the attribute “name” always contain the same value (see source/lib/ldb/modules/rdn_name.c under [SaTe01]).

LDB module“show_deleted”

The LDB module “show_deleted” enforces, that objects, whose attribute “isDeleted” contain the value “TRUE”, are ignored by a search request. The LDAP control “SHOW_DELETED” (see [MSDN01]) is also implemented and enforces, that *tombstones* are not being ignored.

This LDB module has been programmed within the scope of this bachelor thesis (see source/dsdb/samdb/ldb_modules/show_deleted.c under [SaTe01] or appendix A15).

LDB module “partition”

The LDB module “partition” enforces, that requests are redirected to the correct directory partition (see source/dsdb/samdb/ldb_modules/partition.c under [SaTe01]). Here, the LDB control `DSDB_CONTROL_CURRENT_PARTITION_OID` including `struct dsdb_control_current_partition` is attached to the request, in order to transmit information on current directory partitions to the partition specific LDB modules (see source/dsdb/samdb/samdb.h under [SaTe01]).

LDB modules“pdc_fsmo” and “naming_fsmo”

The LDB modules “pdc_fsmo” and “naming_fsmo” do currently have no effect (see source/dsdb/samdb/ldb_modules/ under [SaTe01]). At a later point, the *PDC emulator master* or rather *domain naming master* functionalities shall be implemented (see [MSTN03]).

LDB module “password_hash”

The LDB module “password_hash” enforces, that the password hash attributes “unicodePwd”, “dBCSPwd” and “supplementalCredentials” are generated upon password

modifications and the plaintext password is not stored by default. Furthermore, the attributes “ntPwdHistory”, “ImPwdHistory” and “pwdLastSet” are being maintained (see source/dsdb/samdb/ldb_modules/password_hash.c under [SaTe01]).

LDB module“schema_fsmo”

The LDB module “schema_fsmo” enforces the loading of the directory schema with help from the schema API. On a later point, the *Schema master* functionality shall be implemented by it as well (see [MSTN03]).

This LDB module has been programmed within the scope of this bachelor thesis (see source/dsdb/samdb/ldb_modules/schema_fsmo.c under [SaTe01] or appendix A15).

LDB module “repl_meta_data”

The LDB module “repl_meta_data” enforces, that the meta data necessary for replication is maintained for all *originating updates*. Currently, this is only exemplary implemented for the case of *originating add*, later other cases have to be taken into account as well. Furthermore, the “repl_meta_data” module provides the LDB extended operation `DSDB_EXTENDED_REPLICATED_OBJECTS_OID`, through which *replicating updates* are executed (see source/dsdb/samdb/samdb.h under [SaTe01]).

This LDB module has been programmed within the scope of this bachelor thesis (see source/dsdb/samdb/ldb_modules/repl_meta_data.c under [SaTe01] or appendix A13).

4.3. Change of server role to domain controller role

Within the scope of this bachelor thesis, the function `libnet_BecomeDC()` was implemented. A rough description of its functionality is given hereafter. References are made to parts within the source code (in the appendix), where further details are given. See source/libnet/libnet_become_dc.h and source/libnet/libnet_become_dc.c under [SaTe01] or appendix A5 and appendix A8.

For the sake of completeness, the function `libnet_UnbecomeDC()` was developed analogously to `libnet_BecomeDC()`, it downgrades a domain controller to domain member. However, in this text mere references to the source code are given. See source/libnet/libnet_unbecome_dc.h and source/libnet/libnet_unbecome_dc.c under [SaTe01] or appendix A6 and appendix A9.

4.3.1. The prerequisites for `libnet_BecomeDC()`

As prerequisite, it is expected from `libnet_BecomeDC()`, that a computer account for a domain member is already present in Active Directory.

4.3.2. The input parameters of `libnet_BecomeDc()`

The input parameters are summarised in `struct libnet_BecomeDC`:

- The DNS name of the domain has to be transmitted within the item `in.domain_dns_name`. Internally, sometimes `realm` (usually in capitals) is interchangeably used with `domain_dns_name` (usually in lower case).
- The NETBIOS name of the domain has to be transmitted within the item `in.domain_netbios_name`.
- The domain SID is transmitted within the item `in.domain_sid`.
- The IP address is transmitted as string in the item `in.source_dsa_address`.
- The NETBIOS name of the computer account, which is to be upgraded to domain controller, is transmitted within the item `in.dest_dsa_netbios_name`. Internally, sometimes `hostname` (usually in lower case) is interchangeably used with `domain_dns_name` (usually in capitals). The DNS name is constructed from `hostname` and `domain_dns_name`.
- The function `libnet_BecomeDC()` is dimensioned in a way that permits that after each logical step, already determined results can be transmitted to the calling function via *callback functions* (see `struct libnet_BecomeDC_Callbacks`). Here it is also possible, that the *callback function* enforces, that the function `libnet_BecomeDC()` aborts execution and notifies the calling function of an error. The individual `callback functions` will be explained at a later point.

4.3.3. The logical steps of `libnet_BecomeDC()`

The function `libnet_BecomeDC` handles a complex sequence of network-requests (LDAP and DRSUAPI). For this, the individual asynchronous requests are consolidated into a single asynchronous function, thus the complexity is hidden for the caller. Within Samba, the subsystem *COMPOSITE* provides helper functions for the concatenation of functions (see source/libcli/composite/composite.c under [SaTe01]). A detailed description of the logical steps and partial tasks is provided as comment within source/libnet/libnet_become_dc.c under [SaTe01] or appendix A8 starting from row 37. Roughly seen, the following partial tasks are processed:

- Primarily, a set of information on domain, domain forest and source DSA is gathered. Subsequently, this information is transmitted as `struct libnet_BecomeDC_CheckOptions` to the *callback function* `check_options()`.
- After that, the configuration of the new domain controller is created. Among other things, the "NTDS settings" is created with aid of the function `DsAddEntry` (see also drsuapi.idl under [SaTe03] or appendix A2 starting from row 1018). Subsequently, the information on domain, domain forest, Source DSA and Destination DSA is transmitted to the *callback function* `prepare_db()` as `struct libnet_BecomeDC_PrepareDB`. With this, the calling function (or rather its *callback function*) has sufficient information in order to create the files necessary for the storage of directory data.
- In the next step, the schema partition is replicated. Here, the function `DsGetNCChanges()` is looped, until the partition has been transmitted entirely. After each successful `DsGetNCChanges()` invocation, the result (`struct drsuapi_DsGetNCChangesCtr1` or `struct drsuapi_DsGetNCChangesCtr6`) together with the session key used for encryption and the information on domain, domain forest, source DSA, destination DSA and the directory partition is transmitted as `struct libnet_BecomeDC_StoreChunk` to the *callback function*`schema_chunk()`. The calling function (or rather its *callback function*) can process each replicated directory object (typically in blocks of 133 objects) and store them.
- Afterwards, the configuration partition is replicated. Here, it is proceeded analogously to the schema partition. However, the *callback function* `config_chunk()` is used.
- In a next step, the computer account of the destination DSA is upgraded to domain controller.
- Now, the domain partition is replicated. Here, it is proceeded analogously to the schema partition. However, the *callback function* `domain_chunk()` is used.
- Finally, the function `DsReplicaUpDateRefs()` (also see drsuapi.idl under [SaTe03] or appendix A2 starting from row 633) is called successively for the schema partition, the configuration partition and the domain partition. I suspect, that the function `DsReplicaUpDateRefs()` generates respectively one value of the attribute "repsTo", so that the source DSA can notify the destination DSA about changes at a later point. However, this still needs further exploration.

4.4. Access to schema information

In order to simplify the access to information on the directory schema within Samba, within the scope of this bachelor thesis a *schema API* as well as a *schema cache* was developed. Hereafter the used structures and concepts are introduced briefly and references to paragraphs in the source codes are given, where additional details can be read. See source/dsdb/schema/schema.h, source/dsdb/schema/schema_init.c and source/dsdb/schema/schema_syntax.c under [SaTe01] or appendix A4, A10, A11.

4.4.1. Schema cache

The *schema cache* is realised via `struct dsdb_schema` (see source/dsdb/schema/schema.h under [SaTe01] or appendix A4). There, the following information is made available:

- An array of `struct dsdb_schema_oid_prefix` items contains the *prefix mappings* used within the schema.
- The contents of the attribute “schemaInfo” of the root object of the schema partition is made available as hex string. The hex string representation is used, because that is what Sambas NDR layer expects.
- A dobule chained list of `struct dsdb_attribute` items contains the definitions of all attributes that are present within the schema. Because C macros for the easy handling of double chained lists are given within source/lib/util/dlinklist.h (under [SaTe01]), those are used here.
- A dobule chained list of `struct dsdb_class` items contains the definitions of all object classes that are present within the schema.

Definition of the attributes

The attribute definitions within directory objects of the object class “attributeSchema” are stored persistently within the schema partition. The `struct dsdb_attribute` contains all attribute definition information relevant to the schema in the schema cache. A description of all items of the attribute definition can be consulted under [MSTN04], hereafter only the items relevant to replication are described:

- The RDN of the directory object is stored as string within the item `cn`.
- Within LDAP, an attribute is identified by a name, which can be found within the item `IDAPDisplayName`.
- Each attribute definition has a worldwide unique OID, which is saved as string representation within the attribute “attributeID”. In `struct dsdb_attribute` it is stored within the item `attributID_oid`. The item `attributeID_id` contains the 32 bit ID of the OID, generated by the *prefix mapping*.
- The item `systemFlags` is a 32 bit integer and contains the following bit flags:
  - 0x00000001 means, the attribute will not be replicated.
  - 0x00000002 means, the attribute has to be mandatorily replicated to the *Global Catalog* (GC). Further information on the GC are available under [MSTN07].
  - 0x00000004 means, the attribute is not stored but constructed upon requests.
- Each attribute is assigned to an attribute syntax. Here, the used syntax is determined by the combination of attributes “attributeSyntax” “oMSyntax” and “oMObjectClass”. The list with the attribute syntaxes used within Active Directory is described in the paragraph “Valid Syntaxes for Attributes in the Active Directory Schema” in [MSTN04].
- In Samba, attribute syntaxes are represented by the `struct dsdb_syntax` (see source/

dsdb/schema/schema.h under [SaTe1] or appendix A4 starting with row 30). The item `syntax` in `struct dsdb_attribute` contains references to the used syntax. Each attribute syntax has assigned conversion functions for the conversion between DRSUAPI and LDB representations, which have been implemented in source/ dsdb/schema/schema_syntaxes.c under [SaTe01] or appendix A11.

Definition of object classes

Class definitions in directory objects of the object class "classSchema" are stored persistently within the schema partition. The `struct dsdb_class` contains all attribute definition information relevant to the schema in the schema cache. A description of all items of the class definition can be consulted under [MSTN04], hereafter only the items relevant to replication are described:

- The RDN of the directory object is stored as string within the item `cn`.
- Within LDAP, an object class is identified by a name, which can be found within the item `IDAPDisplayName`.
- Each class definition has a worldwide unique OID, which is saved as string representation within the attribute "governsID". In `struct dsdb_class` it is stored within the item `governsID_oid`. The item `governsID_id` contains the 32 bit ID of the OID, generated by the *prefix mapping*.

4.4.2. The schema API functions for the creation of the schema cache

In order to load the directory schema, a set of functions were developed. At the moment of creation of the schema cache, 3 fundamental steps are necessary, for each, two function variants, one for information in DRSUAPI structures and one for information in LDB structures, are given:

- Primarily, the *prefix mapping* and the attributes "schemaInfo" have to be loaded. See `dsdb_load_oid_mappings_ldb()` and `dsb_load_oid_mappings_drsuapi()` in source/dsdb/schema/schema_init.c under [SaTe01] or rather appendix A10 starting from row 33. For the LDB variant, it has to be considered, that the *prefix mappings* are stored within the attribute "prefixMap", which is not replicated. Since it is not possible with Windows 2003, to request the attribute value via LDAP, a suitable format has been selected (see drsblobs.idl under [SaTe03] or rather appendix A1 starting from row 147).
- In the second step, all attribute definitions have to be loaded into the schema cache. See `dsdb_attribute_from_ldb()` and `dsdb_attribute_from_drsuapi()` in source/dsdb/schema/schema_init.c under [SaTe01] or rather appendix A10 starting from row 365 as well as from row 709.
- In the last step, all class definitions have to be loaded into the schema cache. See `dsdb_class_from_ldb()` and `dsdb_class_from_drsuapi()` in source/ dsdb/schema/schema_init.c under [SaTe01] or rather appendix A10 starting from row 436 as well as from row 771.

The use of the DRSUAPI function variants only serves the replication of the schema partition and thus is only programmed for this intention. Here, only the most necessary information is stored in the schema cache.

The loading of class definition is incomplete in the present implementation and only covers

the functionalities used within this bachelor thesis. Especially, attribute lists and relations between object classes are still missing.

The completely loaded schema cache is bound to a LDB connection with the function `dsdb_set_schema()` (see source/dsdb/schema/schema_init.c under [SaTe01] or rather appendix A10 starting from row 952).

4.4.3. The schema API functions for the use of the schema cache

The used schema cache of a LDB connection can be accessed via the function `dsdb_get_schema()` (see source/dsdb/schema/schema_init.c under [SaTe01] or rather appendix A10 starting from row 952). Generally, the information within the schema cache should only be accessed via the functions, that were intended for this.

Access to the prefix mappings

The functions `dsdb_get_oid_mappings_drsuapi()` and `dsdb_get_oid_mappings_ldb()` give access to the prefix mappings (see source/ dsdb/schema/schema_init.c under [SaTe01] or appendix A10 starting from row 123 and from row 160). The function `dsdb_verify_oid_mappings_drsuapi()` is used for the pull replication in order to evaluate the schema version between source DSA and destination DSA (see source/dsdb/schema/schema_init.c under [SaTe01] or appendix A10 starting from row 188).

Search of attribute and class definitions

IDAPDisplayName The schema API provides functions, that allow the access to attribute and class definitions per OID, 32 bit ID or LDAP attribute name. These are in particular the functions:

- `dsdb_attribute_by_attributeID_id()`
  (see source/dsdb/schema/schema_init.c under [SaTe01] or appendix A10 starting from row 822)
- `dsdb_attribute_by_attributeID_oid()`
  (see source/dsdb/schema/schema_init.c under [SaTe01] or appendix A10 starting from row 843)
- `dsdb_attribute_by_lDAPDisplayName()`
  (see source/dsdb/schema/schema_init.c under [SaTe01] or appendix A10 starting from row 860)
- `dsdb_class_by_governsID_id()`
  (see source/dsdb/schema/schema_init.c under [SaTe01] or appendix A10 starting from row 877)
- `dsdb_class_by_governsID_oid()`
  (see source/dsdb/schema/schema_init.c under [SaTe01] or appendix A10 starting from row 898)
- `dsdb_class_by_lDAPDisplayName()`
  (see source/dsdb/schema/schema_init.c under [SaTe01] or appendix A10 starting from row 915)

In order to search for LDAP attribute name with the 32 bit ID, the function `dsdb_IDAPDisplayName_by_id()` is provided, it is unimportant, if an attribute or a class is identified by the 32 bit ID.

Conversion of attributes between DRSUAPI and LDB representations

At the moment of loading of the attribute definitions, each attribute is assigned a `struct dsdb_syntax` with aid of the function `dsdb_syntax_for_attribute()` (see source/dsdb/schema/schema_syntax.c under [saTe01] or appendix A11 starting from row 1294). With aid of the functions `dsdb_attribute_drsuapi_to_ldb()` and `dsdb_attribute_ldb_to_drsuapi()`, attributes may be converted specifically for the attribute syntax between DRSUAPI representation and LDB representation (see source/dsdb/schema/schema_syntax.c under [SaTe01] or appendix A11 starting from row 1319 and from row 1334). Particularly, `struct drsuapi_DsReplicaAttribute` (see drsuapi.idl under [SaTe03] or appendix A2 starting from row 467) and `struct ldb_message_element` (see source/lib/ldb/include/ldb.h under [SaTe01] starting from row 143) are meant.

4.4.4. LDB module for the loading of the schema cache

Within the scope of this bachelor thesis, the LDB module “schema_fsmo” was developed. Figure 4.4, “Layer comparison between Windows 2003 and Samba 4.0 (compare “Replication Subsystem Components” in [MSTN01]).” shows, in which context it can be used with other LDB modules. At the moment of connection establishment, it loads the schema cache automatically and binds it with the function `dsdb_set_schema()` to the LDB connection, subsequently, other modules may access it. The source code is provided under source/dsdb/samdb/ldb_modules/schema_fsmo.c under [SaTe01] or appendix A14.

4.5. Storage of the replicated data including meta data

In order to interpret the data that has been replicated by a source DSA and to incorporate it into the local data stock, a complex sequence of operations is necessary. Within the scope of this bachelor thesis, the function `dsdb_extended_replicated_objects_commit()` has been developed (see source/dsdb/repl/replicated_objects.c under [SaTe01] or appendix A12 starting from row 338). The function is basically divided into two phases. In the first phase, the received data, is interpreted and conditioned as described in paragraph 3.6.2. In the second phase, conflicts are resolved and it is persistently stored, as described in paragraph 3.6.4.

4.5.1. First phase: Interpretation and conditioning

Since the interpretation and conditioning of the data is very complex, and uses a non-trivial part of the CPU time, related to the entire function, the first phase happens “above” the LDB-API. Therefore, not within a LDB transaction. Concretely, the following steps are processed:

- Primarily, using the loaded directory schema and the *prefix mapping* sent by the source DSA, it is evaluated, if the same schema-version is present on source DSA and destination DSA. If this is not the case, it is aborted with an error message.
- Then, in a loop, all directory objects are converted from DRSUAPI representation to LDB representation. This is done by the helper function `dsdb_convert_object()` (see source/dsdb/repl/replicated_objects.c under [SaTe01] or appendix A12 starting from row 190).
- For each directory object, all attributes are encrypted if necessary, this is implemented using the functions `dsdb_decrypt_attribute()` and `dsdb_decrypt_attribute_value()` (see source/dsdb/repl/replicated_objects.c under [SaTe01] or appendix A12 starting from row 132 and from row 33). Afterwards, the attribute values are brought into the LDB representation with aid from the function `dsdb_attribute_drsuapi_to_ldb()` (see source/dsdb/schema/schema_syntax.c under [SaTe01] or appendix A11 starting from row 1319).

Here, the data is conditioned into a `struct dsdb_extended_replicated_objects`, where the directory objects are made available as array of `struct dsdb_extended_replicated_object` items, furthermore contained are the DN of the directory partition, the *Up to date vector* of the source DSA and the information for the attribute “repsFrom” relating to the source DSA (see source/dsdb/samdb/samdb.h under [SaTe01] or appendix A3 starting from row 55). With use of the *LDB extended operation* `DSDB_EXTENDED_REPLICATION_OBJECTS_OID`, the `struct dsdb_extended_replicated_object` is transmitted to the LDB API.

4.5.2. Second phase: Conflict resolution and storage

The second phase contains the resolution of possible conflicts and finally the storage of the replicated directory objects as well as the attributes “replUpToDateVector” and “repsFrom”. All of this is done within one LDB transaction, that means, it is guaranteed that all changes are saved in their entirety or not at all. The *LDB extended operation*`DSDB_EXTENDED_REPLICATED_OBJECTS_OID` is implemented by the function `replmd_extended_replicated_objects()` of the LDB module “repl_meta_data” (see source/dsdb/samdb/ldb_modules/repl_meta_data.c under [SaTe01] or appendix A13 starting from row 55). Where the LDB module “repl_meta_data” can be found in relation to other LDB modules, is shown by Figure 4.4, “Layer comparison between Windows 2003 and Samba 4.0 (compare “Replication Subsystem Components” in [MSTN01]).”.

Within the function `replmd_extended_replicated_objects()`, the following logic is applied to each replicated directory object within a loop:

- In the functions `replmd_replicated_apply_search()`, each object is searchd using its objectGUID (see source/dsdb/samdb/ldb_modules/repl_meta_data.c under [SaTe01] or appendix A13 starting from row 1095).
- If the object is not found, the function `replmd_replicated_apply_add()` sets the still local USN for the object and then saves it (see source/dsdb/samdb/ldb_modules/repl_meta_data.c under [SaTe01] or appendix A13 starting from row 723).
- Is the object is already present, the function `replmd_replicated_apply_merge()` consolidates the data of the old and new version of the object. This is done using the conflict resolution at attribute level, implemented in `replmd_replPropertyMetaData1_conflict_compare()`. Then, the local USN is set in the necessary places and the object is saved (see source/dsdb/samdb/ldb_modules/repl_meta_data.c under [SaTe01] or appendix A13 starting from row 865 and from row 817).
- As a last subtask, the attributes “replUpToDateVector” and “repsFrom” are updated. The functions `replmd_replicated_uptodate_search()` and `replmd_replicated_uptodate_modify()` handle this task (see source/dsdb/samdb/ldb_modules/repl_meta_data.c undre [SaTe01] or appendix A13 starting from row 1512 and from row 1180).

Conflicts on DN level are still not implemented, because these conflicts do not occur at the initial replication. Furthermore, there is extended need for research in order to implement these cases correctly.

4.5.3. Originating add

The LDB module “repl_meta_data” is additionally responsible for the maintenance of meta data for *originating updates*. Within the scope of this bachelor thesis, the case of *originating add* has been implemented. The function `replmd_add_originating()` handles the tasks described in paragraph 3.5.1 (see source/dsdb/samdb/ldb_modules/repl_meta_data.c under [SaTe01] or appendix A13 starting from row 290). In order to implement other cases of *originating updates* correctly, the still have to be researched further.

4.6. The “NET API BECOME DC” test

In order to test the function that was implemented within this bachelor thesis, Sambas test program `smbtorture` has been extended by the test “NET API BECOME DC”. This test is implemented into the function `torture_net_become_dc()` (see source/torture/libnet/libnet_BecomeDC.c under [SaTe01] or appendix A7 starting from row 729).

4.6.1. Creation of a computer account for testing

In `smbtorture`, the functions `torture_join_domain()` and `torture_leave_domain()` are available for the creation or deletion of a test computer account (see source/torture/rpc/testjoin.c under [SaTe01] starting from row 297 and from row 491). These functions are called at the start or end of the function `torture_net_become_dc()`.

4.6.2. Upgrade to domain controller

With use of the temporary computer account, the function `libnet_BecomeDC()` is being tested. All the data necessary for the test is gathered in the function `struct test_become_dc_state` (see source/torture/libnet/libnet_BecomeDC.c under [SaTe01] starting from row 97) and all helper functions are provided. Herafter, it is explained, which individual tasks are handled by the *callback functions* given to `libnet_BecomeDC()`.

Callback functions for `check_options()`

The function `test_become_dc_check_options()` displays the information received from `libnet_BecomeDC()` on the screen (see source/torture/libnet/libnet_BecomeDC.c under [SaTe01] starting from row 97).

Callback function for `prepare_db()`

The function `test_become_dc_prepare_db()` conditions the LDB files and configures, which LDB modules are to be used. The so-called *provisioning* is implemented within the programming language EJS. What is why the function `test_become_dc_prepare_db()` constructs within a string an EJS program and executes with it the EJS function `provision_become_dc()`. After that, the prepared LDB is connected (see source/torture/libnet/libnet_BecomeDC.c under [SaTe01] starting from row 149 and source/scripting/libjs/provision.js under [SaTe01] starting from row 454).

Callback function for `schema_chunk()`

Schema The function `test_become_dc_schema_chunk()` receives the replicated directory objects of the schema partition. These are temporarily stored within `struct test_become_dc_state` (see source/torture/libnet/libnet_BecomeDC.c under [SaTe01] or appendix A7 starting from row 519). If all objects of the schema partition have been received, a helper schema is created in the function `test_apply_schema()` with aid of the functions `dsdb_attribute_from_drsuapi()` and `dsdb_class_from_drsuapi()`, which subsequently is bound to the LDB connection with use of the function `dsdb_set_schema()`. After that, the directory objects are stored in the LDB via a call of the function `dsdb_extended_replicated_objects_commit()` (see source/torture/libnet/libnet_BecomeDC.c under [SaTe01] or appendix A7 starting from row 285). Thereupon, the *prefix mappings* are written into the non-replicated attribute “prefixMap”. After that, the LDB connection is closed and re-established, within this procedure the helper

schema is discarded and the entirely replicated directory schema is loaded from the LDB module “schema_fsmo”.

Callback function for `config_chunk()` and `domain_chunk()`

The replicated directory objects of the configuration partition and the domain partition are received by the function `test_become_dc_store_chunk()`, which thereupon are saved to the LDB with aid of the function `dsdb_extended_replicated_objects_commit()` (see source/torture/libnet/libnet_BecomeDC.c under [SaTe01] or appendix A7 starting from row 590).

After al the data has been replicated, the LDB connection is closed and re-established for testing purposes.

4.6.3. Downgrading and deletion of the computer account

At the end of the “NET API BECOME DC” test, the computer account is downgraded to domain member by use of the function `libnet_UnbecomeDC()` and subsequently deleted with the function `torture_leave_domain()`.

4.6.4. Execution of “NET API BECOME DC” via `smbtorture`

Attention: `smbtorture` should be used exclusively in environments for testing purposes, important data could get lost!

The installation of Samba 4.0 is described in appendix A17.

The “NET API BECOME DC” test can be executed as follows:

```
#>bin/smbtorture ncacn_np:w2k3-21.w2k3-thesis.vmnet24.vm.base \
        -W W2K3-THESIS -TODO-realm w2k3-thesis.vmnet24.vm.base \
        -U administrator%test \
        NET-API-BECOME-DC
```

With an additional parameter, all replicated data can be displayed with additional detail.
Attention: The use of `less` is important, since around 500 thousand rows are being generated!:

```
#>bin/smbtorture ncacn_np:w2k3-21.w2k3-thesis.vmnet24.vm.base \
        -W W2K3-THESIS -TODO-realm w2k3-thesis.vmnet24.vm.base \
        -U administrator%test \
        NET-API-BECOME-DC \
        --option="become dc:dump objects=yes" | less
```

Chapter 5. Conclusion

The objectives set in chapter “Objectives” have been successfully elaborated.

5.1. What was achieved

The most important aspects of Active-Directory-Replication have been investigated and documented, including:

- The necessary steps for the creation of a computer account for a Domain-Controller in Active-Directory.
- The necessary steps for the replication of all Directory-Data from a Windows-2003-Server.
- The necessary steps for the interpretation of the replicated Directory-Data. In particular, password information can be interpreted as well.

Furthermore, the implemented functions of this bachelor thesis create a solid foundation for the future development of Samba 4.0.

5.2. What is still to be done

Since Samba 4.0 shall become a full-value Domain-Controller for Active-Directory-Domains, there are naturally still many details to be explored and implemented.

The following points have yet to be explored:

- Conflict resolution on DN-level.
- Handling of *Linked-Attributes*.
- The entire subject area of *Replication-Topology*.
- The execution of Schema-Restrictions.
- The execution of access limits.

The following points still have to be implemented:

- Meta data have to be maintained for all *Originating-Updates*.
- The *Pull-Replication* has to be implemented within Sambas Server-Program `smbd` for the roles Source-DSA and Destination-DSA.
- And of course all the aspects, that are still to be explored.

Bibliography

[AppWeb01] Mbedthis Software LLC: Embedded JavaScript; http://www.appwebserver.org/products/ejs/ejs.html; Mbedthis Software LLC, 2003-2006 [last visit: 16.03.2007].

[FSF01] Free Software Foundation: GNU General Public License; http://www.gnu.org/licenses/gpl.html; Free Software Foundation, 1991 [last visit: 16.03.2007].

[FSF02] Free Software Foundation: The Free Software Definition; http://www.gnu.org/philosophy/free-sw.html; Free Software Foundation, 1996-2006 [last visit: 16.03.2007].

[Gla2006] E. Glass: The NTLM Authentication Protocol and Security Support Provider; http://davenport.sourceforge.net/ntlm.html; 2003,2006 [last visit: 16.03.2007].

[GoGr01] D. Sterndark: RC4 Algorithm revealed; http://groups.google.com/group/sci.crypt/msg/10a300c9d21afca0; 1994 [last visit: 16.03.2007].

[Her2003] C. Hertel: Implementing CIFS; http://ubiqx.org/cifs/SMB.html; 1999-2003 [last visit: 16.03.2007].

[ITUT01] INTERNATIONAL TELECOMMUNICATION UNION: Abstract Syntax Notation One (ASN.1) and ASN.1 Encoding Rules; http://www.itu.int/ITU-T/studygroups/com17/languages/X.680-X.693-0207w.zip; INTERNATIONAL TELECOMMUNICATION UNION, 2002 [last visit: 16.03.2007].

[KoMe2005] M. Kohlgraf; S. Metzmacher: Active Directory Replikation; (Student research paper in the subject data networks, WS04/05; Adviso Prof. Dr.-Ing. Andreas Grebe) FH Köln, Institut für Nachrichtentechnik 2005, (see. a. [KoMe2005W]).

[KoMe2005W] M. Kohlgraf; S. Metzmacher: Active Directory Replikation; http://samba.org/~metze/presentations/2005/Datennetze1/WS0405_Acitve_Directory_Replication.pdf; 2005 [last visit: 16.03.2007] (vgl. a. [KoMe2005]).

[Man01] Linux Programmer's Manual: select(2) - Linux man page; http://www.linuxmanpages.com/man2/select.2.php; Linux Programmer's Manual, 2001 [last visit: 16.03.2007].

[Man01] Linux Programmer's Manual: epoll(4) - Linux man page; http://www.linuxmanpages.com/man4/epoll.4.php; Linux Programmer's Manual, 2002 [last visit: 16.03.2007].

[MSDN01] Microsoft MSDN: LDAP_SERVER_SHOW_DELETED_OID; http://msdn2.microsoft.com/en-us/library/aa366989.aspx; Microsoft Corporation, 2007 [last visit: 16.03.2007].

[MSTN01] Microsoft TechNet: How the Active Directory Replication Model Works; http://technet2.microsoft.com/WindowsServer/en/Library/1465d773-b763-45ec-b971-c23cdc27400e1033.mspx; Microsoft Corporation, 2006 [last visit: 16.03.2007].

[MSTN02] Microsoft TechNet: How the Active Directory Topologie Works; http://technet2.microsoft.com/WindowsServer/en/library/c238f32b-4400-4a0c-b4fb-7b0febecfc731033.mspx; Microsoft Corporation, 2005 [last visit: 16.03.2007].

[MSTN03] Microsoft TechNet: Operations master roles; http://technet2.microsoft.com/WindowsServer/en/library/9a353810-8e3a-4023-a557-db1a686d8ec81033.mspx; Microsoft Corporation, 2005 [last visit: 16.03.2007].

[MSTN04] Microsoft TechNet: How the Active Directory Schema Works; http://technet2.microsoft.com/WindowsServer/en/library/e3525d00-a746-4466-bb87-140acb44a6031033.mspx; Microsoft Corporation, 2005 [last visit: 16.03.2007].

[MSTN05] Microsoft TechNet: How the Data Store Works; http://technet2.microsoft.com/WindowsServer/en/library/54094485-71f6-4be8-8ebf-faa45bc5db4c1033.mspx; Microsoft Corporation, 2005 [last visit: 16.03.2007].

[MSTN06] Microsoft TechNet: How Active Directory Functional Levels Work; http://technet2.microsoft.com/WindowsServer/en/library/c3aa5e55-42c0-4386-93ba-66e5c538b0ac1033.mspx; Microsoft Corporation, 2003 [last visit: 16.03.2007].

[MSTN07] Microsoft TechNet: How the Global Catalog Works; http://technet2.microsoft.com/WindowsServer/en/library/440e44ab-ea05-4bd8-a68c-12cf8fb1af501033.mspx; Microsoft Corporation, 2006 [last visit: 16.03.2007].

[OpGr01] The Open Group: DCE 1.1: Remote Procedure Call; http://www.opengroup.org/onlinepubs/9629399/toc.htm; The Open Group, 1997 [last visit: 16.03.2007].

[OpGr02] The Open Group: Transfer Syntax NDR; http://www.opengroup.org/onlinepubs/9629399/chap14.htm; The Open Group, 1997 [last visit: 16.03.2007].

[OpGr03] The Open Group: Interface Definition Language; http://www.opengroup.org/onlinepubs/9629399/chap4.htm; The Open Group, 1997 [last visit: 16.03.2007].

[OpGr04] The Open Group: Remote Procedure Call Model - Endpoints and the Endpoint Mapper; http://www.opengroup.org/onlinepubs/9629399/chap6.htm#tagcjh_11_02_02; The Open Group, 1997 [last visit: 16.03.2007].

[RFC768] University of Southern California - Information Sciences Institute: USER DATAGRAM PROTOCOL; http://www.ietf.org/rfc/rfc768.txt; The Internet Engineering Task Force, 1981 [last visit: 16.03.2007].

[RFC791] J. Postel: INTERNET PROTOCOL; http://www.ietf.org/rfc/rfc791.txt; The Internet Engineering Task Force, 1980 [last visit: 16.03.2007].

[RFC793] University of Southern California - Information Sciences Institute: TRANSMISSION CONTROL PROTOCOL; http://www.ietf.org/rfc/rfc793.txt; The Internet Engineering Task Force, 1981 [last visit: 16.03.2007].

[RFC1001] Network Working Group: PROTOCOL STANDARD FOR A NetBIOS SER-

VICE ON A TCP/UDP TRANSPORT: CONCEPTS AND METHODS; http://www.ietf.org/rfc/rfc1001.txt; The Internet Engineering Task Force, 1987 [last visit: 16.03.2007].

[RFC1002] Network Working Group: PROTOCOL STANDARD FOR A NetBIOS SERVICE ON A TCP/UDP TRANSPORT: DETAILED SPECIFICATIONS http://www.ietf.org/rfc/rfc1002.txt; The Internet Engineering Task Force, 1987 [last visit: 16.03.2007].

[RFC1034] P. Mockapetris: DOMAIN NAMES - CONCEPTS AND FACILITIES; http://www.ietf.org/rfc/rfc1034.txt; The Internet Engineering Task Force, 1987 [last visit: 16.03.2007].

[RFC1035] P. Mockapetris: DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION; http://www.ietf.org/rfc/rfc1035.txt; The Internet Engineering Task Force, 1987 [last visit: 16.03.2007].

[RFC1321] R. Rivest: The MD5 Message-Digest Algorithm; http://www.ietf.org/rfc/rfc1321.txt; The Internet Engineering Task Force, 1992 [last visit: 16.03.2007].

[RFC1951] P. Deutsch: DEFLATE Compressed Data Format Specification version 1.3; http://www.ietf.org/rfc/rfc1951.txt; The Internet Engineering Task Force, 1996 [last visit: 16.03.2007].

[RFC1952] P. Deutsch: GZIP file format specification version 4.3; http://www.ietf.org/rfc/rfc1952.txt; The Internet Engineering Task Force, 1996 [last visit: 16.03.2007].

[RFC2222] J. Myers: Simple Authentication and Security Layer; http://www.ietf.org/rfc/rfc2222.txt; The Internet Engineering Task Force, 1997 [last visit: 16.03.2007].

[RFC2743] J. Linn: Generic Security Service Application Program Interface Version 2, Update 1; http://www.ietf.org/rfc/rfc2743.txt; The Internet Engineering Task Force, 2000 [last visit: 16.03.2007].

[RFC2849] G. Good: The LDAP Data Interchange Format (LDIF) - Technical Specification; http://www.ietf.org/rfc/rfc2849.txt; The Internet Engineering Task Force, 2000 [last visit: 16.03.2007].

[RFC3352] K. Zeilenga: Connection-less Lightweight Directory Access Protocol (CLDAP) to Historic Status; http://www.ietf.org/rfc/rfc3352.txt; The Internet Engineering Task Force, 2003 [last visit: 16.03.2007].

[RFC4120] C. Neuman; T. Yu; S. Hartman; K. Raeburn: The Kerberos Network Authentication Service (V5); http://www.ietf.org/rfc/rfc4120.txt; The Internet Engineering Task Force, 2005 [last visit: 16.03.2007].

[RFC4178] L. Zhu; P. Leach; K. Jaganathan; W. Ingersoll: The Simple and Protected Generic Security Service Application Program Interface (GSS-API) Negotiation Mechanism; http://www.ietf.org/rfc/rfc4178.txt; The Internet Engineering Task Force, 2005 [last visit: 16.03.2007].

[RFC4510] K. Zeilenga: Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map; http://www.ietf.org/rfc/rfc4510.txt; The Internet Engineering Task Force, 2006 [last visit: 16.03.2007].

[RFC4648] S. Josefsson: The Base16, Base32, and Base64 Data Encodings; ht-

tp://www.ietf.org/rfc/rfc4648.txt; The Internet Engineering Task Force, 2006; [last visit: 16.03.2007].

[SaTe01] Samba-Team: Samba 4.0 - Quelltext (Subversion Revision 21859); http://viewcvs.samba.org/cgi-bin/viewcvs.cgi/branches/SAMBA_4_0/?rev=21859; Samba-Team, 2007 [last visit: 16.03.2007].

[SaTe02] Samba-Team: Samba Development; http://devel.samba.org/; Samba-Team, 2007 [last visit: 16.03.2007].

[SaTe03] Samba-Team: Samba IDL-Dateien (Subversion Revision 21859); http://viewcvs.samba.org/cgi-bin/viewcvs.cgi/branches/SAMBA_4_0/source/librpc/idl/?rev=21859; Samba-Team, 2007 [last visit: 16.03.2007].

[SaTe04] Samba-Team: Samba - Opening Windows to a Wider World; http://www.samba.org/; Samba-Team, 2007 [last visit: 16.03.2007].

[SaTe05] Samba-Team: LDB; http://ldb.samba.org/; Samba-Team, 2007 [last visit: 16.03.2007].

[Tridge01] A. Tridgell: How Samba was written; http://samba.org/ftp/tridge/misc/french_cafe.txt; 2003 [last visit: 16.03.2007].

[Tridge02] A. Tridgell: Network Analysis Techniques; http://us4.samba.org/samba/ftp/slides/net_analysis.pdf; 2003 [last visit: 16.03.2007].

[WikiPe01] Wikimedia Foundation Inc.: OSI-Modell; http://de.wikipedia.org/wiki/OSI-Modell; Wikimedia Foundation Inc., 2007 [last visit: 16.03.2007].

[WikiPe02] Wikimedia Foundation Inc.: Linux; http://de.wikipedia.org/wiki/Linux; Wikimedia Foundation Inc., 2007 [last visit: 16.03.2007].

[WikiPe03] Wikimedia Foundation Inc.: RC4; http://de.wikipedia.org/wiki/RC4; Wikimedia Foundation Inc., 2007 [last visit: 16.03.2007].

List of abbreviations

BDC	Backup-Domain-Controller
CIFS	Common Internet File System
CLDAP	Connection-less Lightweight Directory Access Protocol
C	Programming language C
CRC32	Cyclic Redundancy Check 32
DC	Domain-Controllers
DCERPC	Distributed Computing Environment / Remote Procedure Call
DIT	Directory Information Tree
DN	Distinguished Name
DNS	Domain Name Service
DRSUAPI	Directory Replication Service Update API
DSA	Directory Service Agent
DSDB	Directory Service Database
EJS	Embedded-Javascript
FSMO	Flexible Service Operation Master
GC	Global Catalog
GSSAPI	Generic Security Service Application Program Interface
GUID	Global Unique Identifier
IDL	Interface Definition Language
IP	Internet Protocol
KCC	Knowledge Consistency Checker
KDC	Kerberos Key Distribution Center
KRB5	Kerberos 5
LDAP	Lightweight Directory Access Protocol
LDB	Lightweight Database
LDIF	LDAP Data Interchange Format
MD5	Message Digest 5
NC	“Naming Context”
NDR	Network Data Representation
NTDS	NT Directory Service
NTLMSSP	New Technology Lan Manager Secure Service Provider
OID	ASN.1 Object Identifier
PDC	Primary-Domain-Controller
RDN	Relative Distinguished Name

RID	Relative Identifier
RootDSE	Root DSA-specific Entry
SAMDB	Security Account Manager Database
SAM	Security Account Manager
SASL	Simple Authentication and Security Layer
SID	Security Identifier
SMB	Server Message Block
SPNEGO	Simple and Protected Negotiation
TCP	Transmission Control Protocol
TDB	Trivial Database
UDP	User Datagram Protocol
USN	Update Sequence Number
UUID	Universal Unique Identifier
W2K3	Windows-2003
W2K	Windows-2000

Index

Symbols

A

B

C

D

E

F

G

H

I

K

L

T

U

W

X

Chapter 9. Appendix

A1. source/librpc/idl/drsblobs.idl

source/librpc/idl/drsblobs.idl:

```
   1|#include "idl_types.h"
   2|
   3|import "drsuapi.idl";
   4|
   5|[
   6|  uuid("12345778-1234-abcd-0001-00000001"),
   7|  version(0.0),
   8|  pointer_default(unique),
   9|  helpstring("Active Directory Replication LDAP Blobs")
  10|]
  11|interface drsblobs {
  12|        declare bitmap drsuapi_DsReplicaSyncOptions;
  13|        declare bitmap drsuapi_DsReplicaNeighbourFlags;
  14|        declare [v1_enum] enum drsuapi_DsAttributeId;
  15|
  16|        /*
  17|         * replPropertyMetaData
  18|         * w2k  uses version 1
  19|         * w2k3 uses version 1
  20|         */
  21|        typedef struct {
  22|                drsuapi_DsAttributeId attid;
  23|                uint32 version;
  24|                NTTIME_1sec originating_change_time;
  25|                GUID originating_invocation_id;
  26|                hyper originating_usn;
  27|                hyper local_usn;
  28|        } replPropertyMetaData1;
  29|
  30|        typedef struct {
  31|                uint32 count;
  32|                uint32 reserved;
  33|                replPropertyMetaData1 array[count];
  34|        } replPropertyMetaDataCtr1;
  35|
  36|        typedef [nodiscriminant] union {
  37|                [case(1)] replPropertyMetaDataCtr1 ctr1;
  38|        } replPropertyMetaDataCtr;
  39|
  40|        typedef [public] struct {
  41|                uint32 version;
  42|                uint32 reserved;
  43|                [switch_is(version)] replPropertyMetaDataCtr ctr;
  44|        } replPropertyMetaDataBlob;
  45|
  46|        void decode_replPropertyMetaData(
  47|                [in] replPropertyMetaDataBlob blob
  48|                );
  49|
  50|        /*
  51|         * replUpToDateVector
  52|         * w2k  uses version 1
  53|         * w2k3 uses version 2
  54|         */
  55|        typedef struct {
  56|                uint32 count;
  57|                uint32 reserved;
  58|                drsuapi_DsReplicaCursor cursors[count];
  59|        } replUpToDateVectorCtr1;
  60|
  61|        typedef struct {
  62|                uint32 count;
  63|                uint32 reserved;
  64|                drsuapi_DsReplicaCursor2 cursors[count];
  65|        } replUpToDateVectorCtr2;
  66|
  67|        typedef [nodiscriminant] union {
  68|                [case(1)] replUpToDateVectorCtr1 ctr1;
  69|                [case(2)] replUpToDateVectorCtr2 ctr2;
  70|        } replUpToDateVectorCtr;
  71|
  72|        typedef [public] struct {
  73|                uint32 version;
  74|                uint32 reserved;
  75|                [switch_is(version)] replUpToDateVectorCtr ctr;
```

```
  76|        } replUpToDateVectorBlob;
  77|
  78|        void decode_replUpToDateVector(
  79|                [in] replUpToDateVectorBlob blob
  80|                );
  81|
  82|        /*
  83|         * repsFrom/repsTo
  84|         * w2k  uses version 1
  85|         * w2k3 uses version 1
  86|         */
  87|        typedef [public,gensize] struct {
  88|                asclstr dns_name;
  89|        } repsFromTo1OtherInfo;
  90|
  91|        typedef [public,gensize,flag(NDR_PAHEX)] struct {
  92|                /* this includes the 8 bytes of the repsFromToBlob header */
  93|                [value(ndr_size_repsFromTo1(r, ndr->flags)+8)] uint32 blobsize;
  94|                uint32 consecutive_sync_failures;
  95|                NTTIME_1sec last_success;
  96|                NTTIME_1sec last_attempt;
  97|                WERROR result_last_attempt;
  98|                [relative] repsFromTo1OtherInfo *other_info;
  99|                [value(ndr_size_repsFromTo1OtherInfo(other_info, ndr->flags))] uint32
->    other_info_length;
 100|                drsuapi_DsReplicaNeighbourFlags replica_flags;
 101|                uint8 schedule[84];
 102|                uint32 reserved;
 103|                drsuapi_DsReplicaHighWaterMark highwatermark;
 104|                GUID source_dsa_obj_guid; /* the 'objectGuid' field of the CN=NTDS
->    Settings object */
 105|                GUID source_dsa_invocation_id; /* the 'invocationId' field of the CN=NTDS
->    Settings object */
 106|                GUID transport_guid;
 107|        } repsFromTo1;
 108|
 109|        typedef [nodiscriminant] union {
 110|                [case(1)] repsFromTo1 ctr1;
 111|        } repsFromTo;
 112|
 113|        typedef [public] struct {
 114|                uint32 version;
 115|                uint32 reserved;
 116|                [switch_is(version)] repsFromTo ctr;
 117|        } repsFromToBlob;
 118|
 119|        void decode_repsFromTo(
 120|                [in] repsFromToBlob blob
 121|                );
 122|
 123|        /*
 124|         * partialAttributeSet
 125|         * w2k  uses version 1
 126|         * w2k3 uses version 1
 127|         */
 128|        typedef struct {
 129|                uint32 count;
 130|                drsuapi_DsAttributeId array[count];
 131|        } partialAttributeSetCtr1;
 132|
 133|        typedef [nodiscriminant] union {
 134|                [case(1)] partialAttributeSetCtr1 ctr1;
 135|        } partialAttributeSetCtr;
 136|
 137|        typedef [public] struct {
 138|                uint32 version;
 139|                uint32 reserved;
 140|                [switch_is(version)] partialAttributeSetCtr ctr;
 141|        } partialAttributeSetBlob;
 142|
 143|        void decode_partialAttributeSet(
 144|                [in] partialAttributeSetBlob blob
 145|                );
 146|
 147|        /*
 148|         * prefixMap
 149|         * w2k  unknown
 150|         * w2k3 unknown
 151|         * samba4 uses 0x44544442 'DSDB'
 152|         *
 153|         * as we windows don't return the prefixMap attribute when you ask for
 154|         * we don't know the format, but the attribute is not replicated
 155|         * so that we can choose our own format...
 156|         */
 157|        typedef [v1_enum] enum {
 158|                PREFIX_MAP_VERSION_DSDB = 0x44544442
 159|        } prefixMapVersion;
```

```
 160|
 161|        typedef [nodiscriminant] union {
 162|                [case(PREFIX_MAP_VERSION_DSDB)] drsuapi_DsReplicaOIDMapping_Ctr dsdb;
 163|        } prefixMapCtr;
 164|
 165|        typedef [public] struct {
 166|                prefixMapVersion version;
 167|                uint32 reserved;
 168|                [switch_is(version)] prefixMapCtr ctr;
 169|        } prefixMapBlob;
 170|
 171|        void decode_prefixMap(
 172|                [in] prefixMapBlob blob
 173|                );
 174|
 175|        /*
 176|         * the cookie for the LDAP dirsync control
 177|         */
 178|        typedef [nodiscriminant,gensize] union {
 179|                [case(0)];
 180|                [default] replUpToDateVectorBlob uptodateness_vector;
 181|        } ldapControlDirSyncExtra;
 182|
 183|        typedef struct {
 184|                [value(3)] uint32 u1;
 185|                NTTIME time;
 186|                uint32 u2;
 187|                uint32 u3;
 188|                [value(ndr_size_ldapControlDirSyncExtra(&extra,
->    extra.uptodateness_vector.version, 0))]
 189|                        uint32 extra_length;
 190|                drsuapi_DsReplicaHighWaterMark highwatermark;
 191|                GUID guid1;
 192|                [switch_is(extra_length)] ldapControlDirSyncExtra extra;
 193|        } ldapControlDirSyncBlob;
 194|
 195|        typedef [public,relative_base] struct {
 196|                [charset(DOS),value("MSDS")] uint8 msds[4];
 197|                [subcontext(0)] ldapControlDirSyncBlob blob;
 198|        } ldapControlDirSyncCookie;
 199|
 200|        void decode_ldapControlDirSync(
 201|                [in] ldapControlDirSyncCookie cookie
 202|                );
 203|
 204|        typedef struct {
 205|                [value(2*strlen_m(name))] uint16 name_len;
 206|                [value(strlen(data))] uint16 data_len;
 207|                uint16 unknown1; /* 2 for name = 'Packages', 1 for name = 'Primary:*' */
 208|                [charset(UTF16)] uint8 name[name_len];
 209|                /*
 210|                 * the data field contains data as HEX strings
 211|                 *
 212|                 * 'Packages':
 213|                 *   data contains the list of packages
 214|                 *   as non termiated UTF16 strings with
 215|                 *   a UTF16 NULL byte as separator
 216|                 *
 217|                 * 'Primary:Kerberos':
 218|                 *    ...
 219|                 *
 220|                 * 'Primary:WDigest':
 221|                 *    ...
 222|                 *
 223|                 * 'Primary:CLEARTEXT':
 224|                 *    data contains the cleartext password
 225|                 *    as UTF16 string encoded as HEX string
 226|                 */
 227|                [charset(DOS)] uint8 data[data_len];
 228|        } supplementalCredentialsPackage;
 229|
 230|        /* this are 0x30 (48) whitespaces (0x20) followed by 'P' (0x50) */
 231|        const string SUPPLEMENTAL_CREDENTIALS_PREFIX = "
->                P";
 232|
 233|        typedef [gensize] struct {
 234|                [value(SUPPLEMENTAL_CREDENTIALS_PREFIX),charset(UTF16)] uint16
->    prefix[0x31];
 235|                uint16 num_packages;
 236|                supplementalCredentialsPackage packages[num_packages];
 237|        } supplementalCredentialsSubBlob;
 238|
 239|        typedef [public] struct {
 240|                [value(0)] uint32 unknown1;
 241|                [value(ndr_size_supplementalCredentialsSubBlob(&sub, ndr->flags))] uint32
->    __ndr_size;
 242|                [value(0)] uint32 unknown2;
```

```
243|                [subcontext(0),subcontext_size(__ndr_size)]
->    supplementalCredentialsSubBlob sub;
244|                [value(0)] uint8 unknown3;
245|        } supplementalCredentialsBlob;
246|
247|        void decode_supplementalCredentials(
248|                [in] supplementalCredentialsBlob blob
249|                );
250|
251|        typedef [public] struct {
252|                [flag(STR_NOTERM|NDR_REMAINING)] string_array names;
253|        } package_PackagesBlob;
254|
255|        void decode_Packages(
256|                [in] package_PackagesBlob blob
257|                );
258|
259|        typedef struct {
260|                [value(2*strlen_m(string))] uint16 length;
261|                [value(2*strlen_m(string))] uint16 size;
262|                [relative,subcontext(0),subcontext_size(size),flag(STR_NOTERM|NDR_REMAINI
->    NG)] string *string;
263|        } package_PrimaryKerberosString;
264|
265|        typedef struct {
266|                uint32 keytype;
267|                [value((value?value->length:0))] uint32 value_len;
268|                [relative,subcontext(0),subcontext_size(value_len),flag(NDR_REMAINING)]
->    DATA_BLOB *value;
269|                [value(0)] uint32 unknown1;
270|                [value(0)] uint32 unknown2;
271|        } package_PrimaryKerberosKey;
272|
273|        typedef struct {
274|                uint16 num_keys;
275|                uint16 num_old_keys;
276|                package_PrimaryKerberosString salt;
277|                [value(0)] uint32 unknown1;
278|                [value(0)] uint32 unknown2;
279|                package_PrimaryKerberosKey keys[num_keys];
280|                package_PrimaryKerberosKey old_keys[num_old_keys];
281|                udlong unknown3[num_keys];
282|                udlong unknown3_old[num_old_keys];
283|        } package_PrimaryKerberosCtr3;
284|
285|        typedef [nodiscriminant] union {
286|                [case(3)] package_PrimaryKerberosCtr3 ctr3;
287|        } package_PrimaryKerberosCtr;
288|
289|        typedef [public] struct {
290|                [value(3)] uint32 version;
291|                [switch_is(version)] package_PrimaryKerberosCtr ctr;
292|        } package_PrimaryKerberosBlob;
293|
294|        void decode_PrimaryKerberos(
295|                [in] package_PrimaryKerberosBlob blob
296|                );
297|
298|        typedef [public] struct {
299|                [flag(STR_NOTERM|NDR_REMAINING)] string cleartext;
300|        } package_PrimaryCLEARTEXTBlob;
301|
302|        void decode_PrimaryCLEARTEXT(
303|                [in] package_PrimaryCLEARTEXTBlob blob
304|                );
305|
306|        typedef [flag(NDR_PAHEX)] struct {
307|                uint8 hash[16];
308|        } package_PrimaryWDigestHash;
309|
310|        typedef [public] struct {
311|                [value(0x31)] uint16 unknown1;
312|                [value(0x01)] uint8  unknown2;
313|                uint8  num_hashes;
314|                [value(0)] uint32 unknown3;
315|                [value(0)] udlong uuknown4;
316|                package_PrimaryWDigestHash hashes[num_hashes];
317|        } package_PrimaryWDigestBlob;
318|
319|        void decode_PrimaryWDigest(
320|                [in] package_PrimaryWDigestBlob blob
321|                );
322|
323|        typedef struct {
324|                NTTIME time1;
325|                uint32 unknown1;
326|                /*
```

```
327|                 * the secret value is encoded as UTF16 if it's a string
328|                 * but krb5 trusts have random bytes here, so converting to UTF16
329|                 * mayfail...
330|                 *
331|                 * TODO: We should try handle the case of a random buffer in all places
332|                 *       we deal with cleartext passwords from windows
333|                 *
334|                 * so we don't use this:
335|                 *
336|                 * uint32 value_len;
337|                 * [charset(UTF16)] uint8 value[value_len];
338|                 */
339|                DATA_BLOB value;
340|                [flag(NDR_ALIGN4)] DATA_BLOB _pad;
341|        } trustAuthInOutSecret1;
342|
343|        typedef struct {
344|                [relative] trustAuthInOutSecret1 *value1;
345|                [relative] trustAuthInOutSecret1 *value2;
346|        } trustAuthInOutCtr1;
347|
348|        typedef struct {
349|                NTTIME time1;
350|                uint32 unknown1;
351|                DATA_BLOB value;
352|                NTTIME time2;
353|                uint32 unknown2;
354|                uint32 unknown3;
355|                uint32 unknown4;
356|                [flag(NDR_ALIGN4)] DATA_BLOB _pad;
357|        } trustAuthInOutSecret2V1;
358|
359|        typedef struct {
360|                NTTIME time1;
361|                uint32 unknown1;
362|                DATA_BLOB value;
363|                NTTIME time2;
364|                uint32 unknown2;
365|                uint32 unknown3;
366|                [flag(NDR_ALIGN4)] DATA_BLOB _pad;
367|        } trustAuthInOutSecret2V2;
368|
369|        typedef struct {
370|                [relative] trustAuthInOutSecret2V1 *value1;
371|                [relative] trustAuthInOutSecret2V2 *value2;
372|        } trustAuthInOutCtr2;
373|
374|        typedef [nodiscriminant] union {
375|                [case(1)] trustAuthInOutCtr1 ctr1;
376|                [case(2)] trustAuthInOutCtr2 ctr2;
377|        } trustAuthInOutCtr;
378|
379|        typedef [public] struct {
380|                uint32 version;
381|                [switch_is(version)] trustAuthInOutCtr ctr;
382|        } trustAuthInOutBlob;
383|
384|        void decode_trustAuthInOut(
385|                [in] trustAuthInOutBlob blob
386|                );
387|
388|        typedef [public] struct {
389|                uint32 marker;
390|                DATA_BLOB data;
391|        } DsCompressedChunk;
392|
393|        typedef [public] struct {
394|                DsCompressedChunk chunks[5];
395|        } DsCompressedBlob;
396|
397|        void decode_DsCompressed(
398|                [in] DsCompressedBlob blob
399|                );
400|}
```

A2. source/librpc/idl/drsuapi.idl

source/librpc/idl/drsuapi.idl:

```
   1|#include "idl_types.h"
   2|
   3|import "security.idl", "misc.idl", "samr.idl";
   4|
   5|[
   6|  uuid("e3514235-4b06-11d1-ab04-00c04fc2dcd2"),
   7|  version(4.0),
   8|  endpoint("ncacn_np:[\\pipe\\lsass]","ncacn_np:[\\pipe\\protected_storage]",
->    "ncacn_ip_tcp:", "ncalrpc:"),
   9|  authservice("ldap"),
  10|  helpstring("Active Directory Replication"),
  11|  helper("librpc/ndr/ndr_drsuapi.h"),
  12|  pointer_default(unique)
  13|]
  14|interface drsuapi
  15|{
  16|        declare bitmap samr_GroupAttrs;
  17|
  18|        /*****************/
  19|        /* Function 0x00 */
  20|        typedef [bitmap32bit] bitmap {
  21|                DRSUAPI_SUPPORTED_EXTENSION_BASE                        = 0x00000001,
  22|                DRSUAPI_SUPPORTED_EXTENSION_ASYNC_REPLICATION           = 0x00000002,
  23|                DRSUAPI_SUPPORTED_EXTENSION_REMOVEAPI                   = 0x00000004,
  24|                DRSUAPI_SUPPORTED_EXTENSION_MOVEREQ_V2                  = 0x00000008,
  25|                DRSUAPI_SUPPORTED_EXTENSION_GETCHG_COMPRESS             = 0x00000010,
  26|                DRSUAPI_SUPPORTED_EXTENSION_DCINFO_V1                   = 0x00000020,
  27|                DRSUAPI_SUPPORTED_EXTENSION_RESTORE_USN_OPTIMIZATION    = 0x00000040,
  28|                DRSUAPI_SUPPORTED_EXTENSION_00000080                    = 0x00000080,
  29|                DRSUAPI_SUPPORTED_EXTENSION_KCC_EXECUTE                 = 0x00000100,
  30|                DRSUAPI_SUPPORTED_EXTENSION_ADDENTRY_V2                 = 0x00000200,
  31|                DRSUAPI_SUPPORTED_EXTENSION_LINKED_VALUE_REPLICATION    = 0x00000400,
  32|                DRSUAPI_SUPPORTED_EXTENSION_DCINFO_V2                   = 0x00000800,
  33|                DRSUAPI_SUPPORTED_EXTENSION_INSTANCE_TYPE_NOT_REQ_ON_MOD= 0x00001000,
  34|                DRSUAPI_SUPPORTED_EXTENSION_CRYPTO_BIND                 = 0x00002000,
  35|                DRSUAPI_SUPPORTED_EXTENSION_GET_REPL_INFO               = 0x00004000,
  36|                DRSUAPI_SUPPORTED_EXTENSION_STRONG_ENCRYPTION           = 0x00008000,
  37|                DRSUAPI_SUPPORTED_EXTENSION_DCINFO_V01                  = 0x00010000,
  38|                DRSUAPI_SUPPORTED_EXTENSION_TRANSITIVE_MEMBERSHIP       = 0x00020000,
  39|                DRSUAPI_SUPPORTED_EXTENSION_ADD_SID_HISTORY             = 0x00040000,
  40|                DRSUAPI_SUPPORTED_EXTENSION_POST_BETA3                  = 0x00080000,
  41|                DRSUAPI_SUPPORTED_EXTENSION_00100000                    = 0x00100000,
  42|                DRSUAPI_SUPPORTED_EXTENSION_GET_MEMBERSHIPS2            = 0x00200000,
  43|                DRSUAPI_SUPPORTED_EXTENSION_GETCHGREQ_V6                = 0x00400000,
  44|                DRSUAPI_SUPPORTED_EXTENSION_NONDOMAIN_NCS               = 0x00800000,
  45|                DRSUAPI_SUPPORTED_EXTENSION_GETCHGREQ_V8                = 0x01000000,
  46|                DRSUAPI_SUPPORTED_EXTENSION_GETCHGREPLY_V5              = 0x02000000,
  47|                DRSUAPI_SUPPORTED_EXTENSION_GETCHGREPLY_V6              = 0x04000000,
  48|                /*
  49|                 * the following 3 have the same value
  50|                 * repadmin.exe /bind says that
  51|                 */
  52|                DRSUAPI_SUPPORTED_EXTENSION_ADDENTRYREPLY_V3            = 0x08000000,
  53|                DRSUAPI_SUPPORTED_EXTENSION_GETCHGREPLY_V7              = 0x08000000,
  54|                DRSUAPI_SUPPORTED_EXTENSION_VERIFY_OBJECT               = 0x08000000,
  55|                DRSUAPI_SUPPORTED_EXTENSION_XPRESS_COMPRESS             = 0x10000000,
  56|                DRSUAPI_SUPPORTED_EXTENSION_20000000                    = 0x20000000,
  57|                DRSUAPI_SUPPORTED_EXTENSION_40000000                    = 0x40000000,
  58|                DRSUAPI_SUPPORTED_EXTENSION_80000000                    = 0x80000000
  59|        } drsuapi_SupportedExtensions;
  60|
  61|        /* this is used by w2k */
  62|        typedef struct {
  63|                drsuapi_SupportedExtensions supported_extensions;
  64|                GUID site_guid;
  65|                uint32 u1;
  66|        } drsuapi_DsBindInfo24;
  67|
  68|        /* this is used by w2k3 */
  69|        typedef struct {
  70|                drsuapi_SupportedExtensions supported_extensions;
  71|                GUID site_guid;
  72|                uint32 u1;
  73|                uint32 repl_epoch;
  74|        } drsuapi_DsBindInfo28;
  75|
  76|        typedef struct {
  77|                [flag(NDR_REMAINING)] DATA_BLOB info;
  78|        } drsuapi_DsBindInfoFallBack;
```

```
  79|
  80|        typedef [nodiscriminant] union {
  81|                [case(24)][subcontext(4)] drsuapi_DsBindInfo24 info24;
  82|                [case(28)][subcontext(4)] drsuapi_DsBindInfo28 info28;
  83|                [default][subcontext(4)] drsuapi_DsBindInfoFallBack FallBack;
  84|        } drsuapi_DsBindInfo;
  85|
  86|        /* the drsuapi_DsBindInfoCtr was this before
  87|         * typedef [flag(NDR_PAHEX)] struct {
  88|         *      [range(1,10000)] uint32 length;
  89|         *      [size_is(length)] uint8 data[];
  90|         * } drsuapi_DsBindInfo;
  91|         *
  92|         * but we don't want the caller to manually decode this blob,
  93|         * so we're doing it here
  94|         */
  95|
  96|        typedef struct {
  97|                [range(1,10000)] uint32 length;
  98|                [switch_is(length)] drsuapi_DsBindInfo info;
  99|        } drsuapi_DsBindInfoCtr;
 100|
 101|        /* this is a magic guid you need to pass to DsBind to make
->    drsuapi_DsWriteAccountSpn() work
 102|         *
 103|         * maybe the bind_guid could also be the invocation_id see
->    drsuapi_DsReplicaConnection04
 104|         */
 105|        const char *DRSUAPI_DS_BIND_GUID = "e24d201a-4fd6-11d1-a3da-0000f875ae0d";
 106|        /*
 107|         * this magic guid are needed to fetch the whole tree with
->    drsuapi_DsGetNCChanges()
 108|         * as administrator and this values are also used in the destination_dsa_guid
->    field
 109|         * of drsuapi_DsGetNCChangesReq5/8 and the source_dsa_guid is zero.
 110|         */
 111|        const char *DRSUAPI_DS_BIND_GUID_W2K    = "6abec3d1-3054-41c8-a362-5a0c5b7d5d71";
->
 112|        const char *DRSUAPI_DS_BIND_GUID_W2K3   = "6afab99c-6e26-464a-975f-f58f105218bc";
->
 113|
 114|        [public] WERROR drsuapi_DsBind(
 115|                [in,unique]         GUID *bind_guid,
 116|                [in,out,unique]    drsuapi_DsBindInfoCtr *bind_info,
 117|                [out]   policy_handle *bind_handle
 118|                );
 119|
 120|        /*****************/
 121|        /* Function 0x01 */
 122|        WERROR drsuapi_DsUnbind(
 123|                [in,out] policy_handle *bind_handle
 124|                );
 125|
 126|        /*****************/
 127|        /* Function 0x02 */
 128|        typedef [public,gensize] struct {
 129|                [value(ndr_size_drsuapi_DsReplicaObjectIdentifier(r, ndr->flags)-4)]
->    uint32 __ndr_size;
 130|                [value(ndr_size_dom_sid28(&sid, ndr->flags))]  uint32 __ndr_size_sid;
 131|                GUID guid;
 132|                dom_sid28 sid;
 133|                [flag(STR_SIZE4|STR_CHARLEN|STR_CONFORMANT)] string dn;
 134|        } drsuapi_DsReplicaObjectIdentifier;
 135|
 136|        typedef [public] bitmap {
 137|                DRSUAPI_DS_REPLICA_SYNC_ASYNCHRONOUS_OPERATION  = 0x00000001,
 138|                DRSUAPI_DS_REPLICA_SYNC_WRITEABLE               = 0x00000002,
 139|                DRSUAPI_DS_REPLICA_SYNC_PERIODIC                = 0x00000004,
 140|                DRSUAPI_DS_REPLICA_SYNC_INTERSITE_MESSAGING     = 0x00000008,
 141|                DRSUAPI_DS_REPLICA_SYNC_ALL_SOURCES             = 0x00000010,
 142|                DRSUAPI_DS_REPLICA_SYNC_FULL                    = 0x00000020,
 143|                DRSUAPI_DS_REPLICA_SYNC_URGENT                  = 0x00000040,
 144|                DRSUAPI_DS_REPLICA_SYNC_NO_DISCARD              = 0x00000080,
 145|                DRSUAPI_DS_REPLICA_SYNC_FORCE                   = 0x00000100,
 146|                DRSUAPI_DS_REPLICA_SYNC_ADD_REFERENCE           = 0x00000200,
 147|                DRSUAPI_DS_REPLICA_SYNC_NEVER_COMPLETED         = 0x00000400,
 148|                DRSUAPI_DS_REPLICA_SYNC_TWO_WAY                 = 0x00000800,
 149|                DRSUAPI_DS_REPLICA_SYNC_NEVER_NOTIFY            = 0x00001000,
 150|                DRSUAPI_DS_REPLICA_SYNC_INITIAL                 = 0x00002000,
 151|                DRSUAPI_DS_REPLICA_SYNC_USE_COMPRESSION         = 0x00004000,
 152|                DRSUAPI_DS_REPLICA_SYNC_ABANDONED               = 0x00008000,
 153|                DRSUAPI_DS_REPLICA_SYNC_INITIAL_IN_PROGRESS     = 0x00010000,
 154|                DRSUAPI_DS_REPLICA_SYNC_PARTIAL_ATTRIBUTE_SET   = 0x00020000,
 155|                DRSUAPI_DS_REPLICA_SYNC_REQUEUE                 = 0x00040000,
 156|                DRSUAPI_DS_REPLICA_SYNC_NOTIFICATION            = 0x00080000,
 157|                DRSUAPI_DS_REPLICA_SYNC_ASYNCHRONOUS_REPLICA    = 0x00100000,
 158|                DRSUAPI_DS_REPLICA_SYNC_CRITICAL                = 0x00200000,
```

```
 159|                DRSUAPI_DS_REPLICA_SYNC_FULL_IN_PROGRESS        = 0x00400000,
 160|                DRSUAPI_DS_REPLICA_SYNC_PREEMPTED               = 0x00800000
 161|        } drsuapi_DsReplicaSyncOptions;
 162|
 163|        typedef struct {
 164|                drsuapi_DsReplicaObjectIdentifier *naming_context;
 165|                GUID source_dsa_guid;
 166|                astring *other_info; /* I assume this is related to the
->    repsFromTo1OtherInfo dns_name */
 167|                drsuapi_DsReplicaSyncOptions options;
 168|        } drsuapi_DsReplicaSyncRequest1;
 169|
 170|        typedef [switch_type(int32)] union {
 171|                [case(1)] drsuapi_DsReplicaSyncRequest1 req1;
 172|        } drsuapi_DsReplicaSyncRequest;
 173|
 174|        WERROR drsuapi_DsReplicaSync(
 175|                [in] policy_handle *bind_handle,
 176|                [in] int32 level,
 177|                [in,switch_is(level)] drsuapi_DsReplicaSyncRequest req
 178|                );
 179|
 180|        /*****************/
 181|        /* Function 0x03 */
 182|        typedef [public] struct {
 183|                hyper tmp_highest_usn; /* updated after each object update */
 184|                hyper reserved_usn;
 185|                hyper highest_usn; /* updated after a full replication cycle */
 186|        } drsuapi_DsReplicaHighWaterMark;
 187|
 188|        typedef [public] struct {
 189|                GUID source_dsa_invocation_id; /* the 'invocationId' field of the CN=NTDS
->    Settings object */
 190|                hyper highest_usn;  /* updated after a full replication cycle */
 191|        } drsuapi_DsReplicaCursor;
 192|
 193|        typedef struct {
 194|                uint32 u1;
 195|                uint32 u2;
 196|                [range(0,0x100000)] uint32 count;
 197|                uint32 u3;
 198|                [size_is(count)] drsuapi_DsReplicaCursor cursors[];
 199|        } drsuapi_DsReplicaCursorCtrEx;
 200|
 201|        typedef [public] bitmap {
 202|                /* the _WRITEABLE flag indicates a replication with all attributes
 203|                 *
 204|                 * --metze
 205|                 */
 206|                DRSUAPI_DS_REPLICA_NEIGHBOUR_WRITEABLE                          =
->    0x00000010,
 207|                DRSUAPI_DS_REPLICA_NEIGHBOUR_SYNC_ON_STARTUP                    =
->    0x00000020,
 208|                DRSUAPI_DS_REPLICA_NEIGHBOUR_DO_SCHEDULED_SYNCS                 =
->    0x00000040,
 209|                DRSUAPI_DS_REPLICA_NEIGHBOUR_USE_ASYNC_INTERSIDE_TRANSPORT      =
->    0x00000080,
 210|                DRSUAPI_DS_REPLICA_NEIGHBOUR_TWO_WAY_SYNC                       =
->    0x00000200,
 211|                DRSUAPI_DS_REPLICA_NEIGHBOUR_RETURN_OBJECT_PARENTS              =
->    0x00000800,
 212|                DRSUAPI_DS_REPLICA_NEIGHBOUR_FULL_IN_PROGRESS                   =
->    0x00001000, /* was 0x00010000, */
 213|                DRSUAPI_DS_REPLICA_NEIGHBOUR_FULL_NEXT_PACKET                   =
->    0x00002000, /* was 0x00020000, */
 214|                DRSUAPI_DS_REPLICA_NEIGHBOUR_NEVER_SYNCED                       =
->    0x00200000,
 215|                DRSUAPI_DS_REPLICA_NEIGHBOUR_PREEMPTED                          =
->    0x01000000,
 216|                DRSUAPI_DS_REPLICA_NEIGHBOUR_IGNORE_CHANGE_NOTIFICATIONS        =
->    0x04000000,
 217|                DRSUAPI_DS_REPLICA_NEIGHBOUR_DISABLE_SCHEDULED_SYNC             =
->    0x08000000,
 218|                /*
 219|                 * the following NOTE applies to DsGetNCChangesRequest5:
 220|                 *  - the data is only compressed when 10 or more objects are replicated
 221|                 *  - but there could also be a size limit of 35 KBytes or something like
->    that
 222|                 *  - the reply is DsGetNCChangesCtr2
 223|                 *  - maybe the same applies to DsGetNCChangesRequest8...
 224|                 *
 225|                 *  --metze
 226|                 */
 227|                DRSUAPI_DS_REPLICA_NEIGHBOUR_COMPRESS_CHANGES                   =
->    0x10000000,
 228|                DRSUAPI_DS_REPLICA_NEIGHBOUR_NO_CHANGE_NOTIFICATIONS            =
->    0x20000000,
```

```
 229|                DRSUAPI_DS_REPLICA_NEIGHBOUR_PARTIAL_ATTRIBUTE_SET             =
->    0x40000000
 230|        } drsuapi_DsReplicaNeighbourFlags;
 231|
 232|        typedef struct {
 233|                GUID destination_dsa_guid;
 234|                GUID source_dsa_invocation_id; /* the 'invocationId' field of the CN=NTDS
->    Settings object */
 235|                [ref] drsuapi_DsReplicaObjectIdentifier *naming_context;
 236|                drsuapi_DsReplicaHighWaterMark highwatermark;
 237|                drsuapi_DsReplicaCursorCtrEx *uptodateness_vector;
 238|                drsuapi_DsReplicaNeighbourFlags replica_flags;
 239|                uint32 max_object_count; /* w2k3 uses min(133,max(100,max_object_count))
->    */
 240|                uint32 max_ndr_size; /* w2k3 seems to ignore this */
 241|                uint32 unknown4;
 242|                hyper h1;
 243|        } drsuapi_DsGetNCChangesRequest5;
 244|
 245|        /*
 246|         * In DRSUAPI all attributes with syntax 2.5.5.2
 247|         * are identified by uint32 values
 248|         *
 249|         * the following table shows the mapping used between the two representations
 250|         * e.g. - objectClass 'nTDSDSA' has governsID: 1.2.840.113556.1.5.7000.47
 251|         *        and a UINT32-ID of '0x0017002F'.
 252|         *      - so the OID 1.2.840.113556.1.5.7000.47 is splitted into a
 253|         *        OID-prefix: 1.2.840.113556.1.5.7000
 254|         *        and a value: 47 => 0x2F
 255|         *      - the mapping table gives a UINT32-prefix: 0x00170000
 256|         *      - and the UINT32-ID is 0x0017002F = 0x00170000 | 0x2F
 257|         *
 258|         * This prefix mapping table is replied in the drsuapi_DsReplicaOIDMapping_Ctr
 259|         * array. The following are the default mappings of w2k3
 260|         *
 261|         * OID-prefix                  => UINT32-Id prefix
 262|         *
 263|         * 2.5.4.*                     => 0x00000000 (standard attributes RFC2256
->    core.schema)
 264|         * 2.5.6.*                     => 0x00010000 (standard object classes RFC2256
->    core.schema)
 265|         * 1.2.840.113556.1.2.*         => 0x00020000
 266|         * 1.2.840.113556.1.3.*         => 0x00030000
 267|         * 2.5.5.*                     => 0x00080000 (attributeSyntax OID's)
 268|         * 1.2.840.113556.1.4.*         => 0x00090000
 269|         * 1.2.840.113556.1.5.*         => 0x000A0000
 270|         * 2.16.840.1.113730.3.*        => 0x00140000
 271|         * 0.9.2342.19200300.100.1.*    => 0x00150000
 272|         * 2.16.840.1.113730.3.1.*      => 0x00160000
 273|         * 1.2.840.113556.1.5.7000.*    => 0x00170000
 274|         * 2.5.21.*                     => 0x00180000 (attrs for SubSchema)
 275|         * 2.5.18.*                     => 0x00190000 (createTimeStamp,modifyTimeStamp,
->    SubSchema)
 276|         * 2.5.20.*                     => 0x001A0000
 277|         * 1.3.6.1.4.1.1466.101.119.*   => 0x001B0000 (dynamicObject, entryTTL)
 278|         * 2.16.840.1.113730.3.2.*      => 0x001C0000
 279|         * 1.3.6.1.4.1.250.1.*          => 0x001D0000
 280|         * 1.2.840.113549.1.9.*         => 0x001E0000 (unstructuredAddress,unstructuredNa
->   me)
 281|         * 0.9.2342.19200300.100.4.*    => 0x001F0000
 282|         *
 283|         * Here's a list of used 'attributeSyntax' OID's
 284|         *
 285|         * 2.5.5.1      => Object(DS-DN) string
 286|         *                   struct drsuapi_DsObjectIdentifier3
 287|         *
 288|         * 2.5.5.2      => OID-string
 289|         *                => all values are represented as uint32 values in drsuapi
 290|         *                => governsID, attributeID and attributeSyntax returned as
->    OID-Strings in LDAP
 291|         *                => mayContain, mustContain and all other attributes with 2.5.5.2
->    syntax
 292|         *                   are returned as attribute names
 293|         *
 294|         * 2.5.5.4      => String(Teletex) case-insensitive string with teletex charset
 295|         *
 296|         * 2.5.5.5      => String(IA5) case-sensitive string
 297|         *
 298|         * 2.5.5.6      => String(Numeric)
 299|         *                => eg. internationalISDNNumber
 300|         *
 301|         * 2.5.5.7      => Object(DN-Binary) B:<byte count>:<bytes>:<object DN>
 302|         *                => e.g. wellKnownObjects
 303|         *
 304|         * 2.5.5.8      => BOOL
 305|         *
 306|         * 2.5.5.9      => int32
```

```
 307|         *
 308|         * 2.5.5.10     => DATA_BLOB
 309|         *                => struct GUID
 310|         *
 311|         * 2.5.5.11     => LDAP timestring
 312|         *                => NTTIME_1sec
 313|         *
 314|         * 2.5.5.12     => String(Unicode) case-insensitive string
 315|         *                => 'standard strings'
 316|         *
 317|         * 2.5.5.13     => Object(Presentation-Address) string
 318|         *                => used in objectClass applicationEntity
 319|         *
 320|         * 2.5.5.14     => Object(DN-String) S:<char count>:<string>:<object DN>
 321|         *                => not used
 322|         *
 323|         * 2.5.5.15     => ntSecurityDescriptor
 324|         *
 325|         * 2.5.5.16     => int64
 326|         *
 327|         * 2.5.5.17     => dom_sid
 328|         */
 329|        typedef [nopush,nopull] struct {
 330|                [range(0,10000),value(ndr_size_drsuapi_DsReplicaOID_oid(oid, 0))] uint32
->    __ndr_size;
 331|                [size_is(__ndr_size),charset(DOS)] uint8 *oid; /* it's encoded with
->    asn1_write_OID_String() */
 332|        } drsuapi_DsReplicaOID;
 333|
 334|        typedef struct {
 335|                uint32 id_prefix;
 336|                drsuapi_DsReplicaOID oid;
 337|        } drsuapi_DsReplicaOIDMapping;
 338|
 339|        typedef [public] struct {
 340|                [range(0,0x100000)] uint32 num_mappings;
 341|                [size_is(num_mappings)] drsuapi_DsReplicaOIDMapping *mappings;
 342|        } drsuapi_DsReplicaOIDMapping_Ctr;
 343|
 344|        typedef [flag(NDR_PAHEX),v1_enum] enum {
 345|                DRSUAPI_OBJECTCLASS_top                 = 0x00010000,
 346|                DRSUAPI_OBJECTCLASS_classSchema         = 0x0003000d,
 347|                DRSUAPI_OBJECTCLASS_attributeSchema     = 0x0003000e
 348|        } drsuapi_DsObjectClassId;
 349|
 350|        typedef [flag(NDR_PAHEX),v1_enum,public] enum {
 351|                DRSUAPI_ATTRIBUTE_objectClass                   = 0x00000000,
 352|                DRSUAPI_ATTRIBUTE_description                   = 0x0000000d,
 353|                DRSUAPI_ATTRIBUTE_member                        = 0x0000001f,
 354|                DRSUAPI_ATTRIBUTE_instanceType                  = 0x00020001,
 355|                DRSUAPI_ATTRIBUTE_whenCreated                   = 0x00020002,
 356|                DRSUAPI_ATTRIBUTE_hasMasterNCs                  = 0x0002000e,
 357|                DRSUAPI_ATTRIBUTE_governsID                     = 0x00020016,
 358|                DRSUAPI_ATTRIBUTE_attributeID                   = 0x0002001e,
 359|                DRSUAPI_ATTRIBUTE_attributeSyntax               = 0x00020020,
 360|                DRSUAPI_ATTRIBUTE_isSingleValued                = 0x00020021,
 361|                DRSUAPI_ATTRIBUTE_rangeLower                    = 0x00020022,
 362|                DRSUAPI_ATTRIBUTE_rangeUpper                    = 0x00020023,
 363|                DRSUAPI_ATTRIBUTE_dMDLocation                   = 0x00020024,
 364|                DRSUAPI_ATTRIBUTE_objectVersion                 = 0x0002004c,
 365|                DRSUAPI_ATTRIBUTE_invocationId                  = 0x00020073,
 366|                DRSUAPI_ATTRIBUTE_showInAdvancedViewOnly        = 0x000200a9,
 367|                DRSUAPI_ATTRIBUTE_adminDisplayName              = 0x000200c2,
 368|                DRSUAPI_ATTRIBUTE_adminDescription              = 0x000200e2,
 369|                DRSUAPI_ATTRIBUTE_oMSyntax                      = 0x000200e7,
 370|                DRSUAPI_ATTRIBUTE_ntSecurityDescriptor          = 0x00020119,
 371|                DRSUAPI_ATTRIBUTE_searchFlags                   = 0x0002014e,
 372|                DRSUAPI_ATTRIBUTE_lDAPDisplayName               = 0x000201cc,
 373|                DRSUAPI_ATTRIBUTE_name                          = 0x00090001,
 374|                DRSUAPI_ATTRIBUTE_currentValue                  = 0x0009001b,
 375|                DRSUAPI_ATTRIBUTE_objectSid                     = 0x00090092,
 376|                DRSUAPI_ATTRIBUTE_schemaIDGUID                  = 0x00090094,
 377|                DRSUAPI_ATTRIBUTE_dBCSPwd                       = 0x00090037,/* lmPwdHash
->    */
 378|                DRSUAPI_ATTRIBUTE_unicodePwd                    = 0x0009005a,/* ntPwdHash
->    */
 379|                DRSUAPI_ATTRIBUTE_ntPwdHistory                  = 0x0009005e,
 380|                DRSUAPI_ATTRIBUTE_priorValue                    = 0x00090064,
 381|                DRSUAPI_ATTRIBUTE_supplementalCredentials       = 0x0009007d,
 382|                DRSUAPI_ATTRIBUTE_trustAuthIncoming             = 0x00090081,
 383|                DRSUAPI_ATTRIBUTE_trustAuthOutgoing             = 0x00090087,
 384|                DRSUAPI_ATTRIBUTE_lmPwdHistory                  = 0x000900a0,
 385|                DRSUAPI_ATTRIBUTE_sAMAccountName                = 0x000900dd,
 386|                DRSUAPI_ATTRIBUTE_fSMORoleOwner                 = 0x00090171,
 387|                DRSUAPI_ATTRIBUTE_systemFlags                   = 0x00090177,
 388|                DRSUAPI_ATTRIBUTE_serverReference               = 0x00090203,
 389|                DRSUAPI_ATTRIBUTE_serverReferenceBL             = 0x00090204,
```

```
390|                DRSUAPI_ATTRIBUTE_initialAuthIncoming           = 0x0009021b,
391|                DRSUAPI_ATTRIBUTE_initialAuthOutgoing           = 0x0009021c,
392|                DRSUAPI_ATTRIBUTE_wellKnownObjects              = 0x0009026a,
393|                DRSUAPI_ATTRIBUTE_isMemberOfPartialAttributeSet = 0x0009027f,
394|                DRSUAPI_ATTRIBUTE_objectCategory                = 0x0009030e,
395|                DRSUAPI_ATTRIBUTE_gPLink                        = 0x0009037b,
396|                DRSUAPI_ATTRIBUTE_msDS_Behavior_Version         = 0x000905b3,
397|                DRSUAPI_ATTRIBUTE_msDS_KeyVersionNumber         = 0x000906f6,
398|                DRSUAPI_ATTRIBUTE_msDS_HasDomainNCs             = 0x0009071c,
399|                DRSUAPI_ATTRIBUTE_msDS_hasMasterNCs             = 0x0009072c
400|        } drsuapi_DsAttributeId;
401|
402|        typedef struct {
403|                GUID destination_dsa_guid;
404|                GUID source_dsa_invocation_id; /* the 'invocationId' field of the CN=NTDS
->    Settings object */
405|                [ref] drsuapi_DsReplicaObjectIdentifier *naming_context;
406|                drsuapi_DsReplicaHighWaterMark highwatermark;
407|                drsuapi_DsReplicaCursorCtrEx *uptodateness_vector;
408|                drsuapi_DsReplicaNeighbourFlags replica_flags;
409|                uint32 max_object_count; /* w2k3 uses min(133,max(100,max_object_count))
->    */
410|                uint32 max_ndr_size; /* w2k3 seems to ignore this */
411|                uint32 unknown4;
412|                hyper h1;
413|                uint32 unique_ptr1;
414|                uint32 unique_ptr2;
415|                drsuapi_DsReplicaOIDMapping_Ctr mapping_ctr;
416|        } drsuapi_DsGetNCChangesRequest8;
417|
418|        typedef [switch_type(int32)] union {
419|                [case(5)] drsuapi_DsGetNCChangesRequest5 req5;
420|                [case(8)] drsuapi_DsGetNCChangesRequest8 req8;
421|        } drsuapi_DsGetNCChangesRequest;
422|
423|        typedef [public] struct {
424|                GUID source_dsa_invocation_id; /* the 'invocationId' field of the CN=NTDS
->    Settings object */
425|                hyper highest_usn;  /* updated after a full replication cycle */
426|                NTTIME last_sync_success;
427|        } drsuapi_DsReplicaCursor2;
428|
429|        typedef struct {
430|                uint32 u1;
431|                uint32 u2;
432|                [range(0,0x100000)] uint32 count;
433|                uint32 u3;
434|                [size_is(count)] drsuapi_DsReplicaCursor2 cursors[];
435|        } drsuapi_DsReplicaCursor2CtrEx;
436|
437|        /* Generic DATA_BLOB values */
438|        typedef struct {
439|                [range(0,10485760),value(ndr_size_DATA_BLOB(0,blob,0))] uint32
->    __ndr_size;
440|                DATA_BLOB *blob;
441|        } drsuapi_DsAttributeValue;
442|
443|        typedef struct {
444|                [range(0,10485760)] uint32 num_values;
445|                [size_is(num_values)] drsuapi_DsAttributeValue *values;
446|        } drsuapi_DsAttributeValueCtr;
447|
448|        /* DN String values */
449|        typedef [public,gensize] struct {
450|                [value(ndr_size_drsuapi_DsReplicaObjectIdentifier3(r, ndr->flags))]
->    uint32 __ndr_size;
451|                [value(ndr_size_dom_sid28(&sid,ndr->flags))]  uint32 __ndr_size_sid;
452|                GUID guid;
453|                dom_sid28 sid;
454|                [flag(STR_SIZE4|STR_CHARLEN)] string dn;
455|        } drsuapi_DsReplicaObjectIdentifier3;
456|
457|        typedef [public,gensize] struct {
458|                [value(ndr_size_drsuapi_DsReplicaObjectIdentifier3Binary(r, ndr->flags))]
->    uint32 __ndr_size;
459|                [value(ndr_size_dom_sid28(&sid,ndr->flags))]  uint32 __ndr_size_sid;
460|                GUID guid;
461|                dom_sid28 sid;
462|                [flag(STR_SIZE4|STR_CHARLEN)] string dn;
463|                [value(binary.length + 4)] uint32 __ndr_size_binary;
464|                [flag(NDR_REMAINING)] DATA_BLOB binary;
465|        } drsuapi_DsReplicaObjectIdentifier3Binary;
466|
467|        typedef [public] struct {
468|                drsuapi_DsAttributeId attid;
469|                drsuapi_DsAttributeValueCtr value_ctr;
470|        } drsuapi_DsReplicaAttribute;
```

```
471|
472|        typedef struct {
473|                [range(0,1048576)] uint32 num_attributes;
474|                [size_is(num_attributes)]  drsuapi_DsReplicaAttribute *attributes;
475|        } drsuapi_DsReplicaAttributeCtr;
476|
477|        typedef [public] struct {
478|                drsuapi_DsReplicaObjectIdentifier *identifier;
479|                uint32 unknown1;
480|                drsuapi_DsReplicaAttributeCtr attribute_ctr;
481|        } drsuapi_DsReplicaObject;
482|
483|        typedef struct {
484|                uint32 version;
485|                NTTIME_1sec originating_change_time;
486|                GUID originating_invocation_id;
487|                hyper originating_usn;
488|        } drsuapi_DsReplicaMetaData;
489|
490|        typedef [public] struct {
491|                [range(0,1048576)] uint32 count;
492|                [size_is(count)] drsuapi_DsReplicaMetaData meta_data[];
493|        } drsuapi_DsReplicaMetaDataCtr;
494|
495|        typedef [public,noprint] struct {
496|                drsuapi_DsReplicaObjectListItemEx *next_object;
497|                drsuapi_DsReplicaObject object;
498|                uint32 unknown1;
499|                GUID *parent_object_guid;
500|                drsuapi_DsReplicaMetaDataCtr *meta_data_ctr;
501|        } drsuapi_DsReplicaObjectListItemEx;
502|
503|        typedef [public,gensize] struct {
504|                GUID source_dsa_guid; /* the 'objectGUID' field of the CN=NTDS Settings
->    object */
505|                GUID source_dsa_invocation_id; /* the 'invocationId' field of the CN=NTDS
->    Settings object */
506|                drsuapi_DsReplicaObjectIdentifier *naming_context;
507|                drsuapi_DsReplicaHighWaterMark old_highwatermark;
508|                drsuapi_DsReplicaHighWaterMark new_highwatermark;
509|                drsuapi_DsReplicaCursorCtrEx *uptodateness_vector;
510|                drsuapi_DsReplicaOIDMapping_Ctr mapping_ctr;
511|                uint32 total_object_count;
512|                uint32 object_count;
513|                /* this +55 is sometimes +56, so I don't know where this comes from...
->    --metze */
514|                [value(ndr_size_drsuapi_DsGetNCChangesCtr1(r,ndr->flags)+55)] uint32
->    __ndr_size;
515|                drsuapi_DsReplicaObjectListItemEx *first_object;
516|                uint32 unknown4;
517|        } drsuapi_DsGetNCChangesCtr1;
518|
519|        /*
520|         * if the DRSUAPI_DS_LINKED_ATTRIBUTE_FLAG_ACTIVE flag
521|         * isn't there it means the value is deleted
522|         */
523|        typedef [public] bitmap {
524|                DRSUAPI_DS_LINKED_ATTRIBUTE_FLAG_ACTIVE = 0x00000001
525|        } drsuapi_DsLinkedAttributeFlags;
526|
527|        typedef [public] struct {
528|                drsuapi_DsReplicaObjectIdentifier *identifier;
529|                drsuapi_DsAttributeId attid;
530|                drsuapi_DsAttributeValue value;
531|                drsuapi_DsLinkedAttributeFlags flags;
532|                NTTIME_1sec originating_add_time;
533|                drsuapi_DsReplicaMetaData meta_data;
534|        } drsuapi_DsReplicaLinkedAttribute;
535|
536|        typedef [public,gensize] struct {
537|                GUID source_dsa_guid; /* the 'objectGUID' field of the CN=NTDS Settings
->    object */
538|                GUID source_dsa_invocation_id; /* the 'invocationId' field of the CN=NTDS
->    Settings object */
539|                drsuapi_DsReplicaObjectIdentifier *naming_context;
540|                drsuapi_DsReplicaHighWaterMark old_highwatermark;
541|                drsuapi_DsReplicaHighWaterMark new_highwatermark;
542|                drsuapi_DsReplicaCursor2CtrEx *uptodateness_vector;
543|                drsuapi_DsReplicaOIDMapping_Ctr mapping_ctr;
544|                uint32 total_object_count;
545|                uint32 object_count;
546|                /* this +55 is sometimes +56, so I don't know where this comes from...
->    --metze */
547|                [value(ndr_size_drsuapi_DsGetNCChangesCtr6(r,ndr->flags)+55)] uint32
->    __ndr_size;
548|                drsuapi_DsReplicaObjectListItemEx *first_object;
549|                uint32 unknown4;
```

```
 550|                uint32 unknown5;
 551|                uint32 unknown6;
 552|                [range(0,1048576)] uint32 linked_attributes_count;
 553|                [size_is(linked_attributes_count)] drsuapi_DsReplicaLinkedAttribute
->    *linked_attributes;
 554|                uint32 unknown7;
 555|        } drsuapi_DsGetNCChangesCtr6;
 556|
 557|        typedef struct {
 558|                uint32 decompressed_length;
 559|                uint32 compressed_length;
 560|                [subcontext(4),subcontext_size(compressed_length),
 561|                 compression(NDR_COMPRESSION_MSZIP,compressed_length,decompressed_length)
->    ]
 562|                 drsuapi_DsGetNCChangesCtr1 *ctr1;
 563|        } drsuapi_DsGetNCChangesMSZIPCtr1;
 564|
 565|        typedef struct {
 566|                uint32 decompressed_length;
 567|                uint32 compressed_length;
 568|                [subcontext(4),subcontext_size(compressed_length),
 569|                 compression(NDR_COMPRESSION_MSZIP,compressed_length,decompressed_length)
->    ]
 570|                 drsuapi_DsGetNCChangesCtr6 *ctr6;
 571|        } drsuapi_DsGetNCChangesMSZIPCtr6;
 572|
 573|        typedef struct {
 574|                uint32 decompressed_length;
 575|                uint32 compressed_length;
 576|                [subcontext(4),subcontext_size(compressed_length),
 577|                 compression(NDR_COMPRESSION_XPRESS,compressed_length,decompressed_length
->    ),
 578|                 flag(NDR_REMAINING)] DATA_BLOB *decompressed;
 579|        } drsuapi_DsGetNCChangesXPRESSCtr1;
 580|
 581|        typedef struct {
 582|                uint32 decompressed_length;
 583|                uint32 compressed_length;
 584|                [subcontext(4),subcontext_size(compressed_length),
 585|                 compression(NDR_COMPRESSION_XPRESS,compressed_length,decompressed_length
->    ),
 586|                 flag(NDR_REMAINING)] DATA_BLOB *decompressed;
 587|        } drsuapi_DsGetNCChangesXPRESSCtr6;
 588|
 589|        typedef [enum16bit] enum {
 590|                DRSUAPI_COMPRESSION_TYPE_MSZIP  = 2,
 591|                DRSUAPI_COMPRESSION_TYPE_XPRESS = 3
 592|        } drsuapi_DsGetNCChangesCompressionType;
 593|
 594|        typedef [nodiscriminant,flag(NDR_PAHEX)] union {
 595|                [case(1|(DRSUAPI_COMPRESSION_TYPE_MSZIP<<16))]
->    drsuapi_DsGetNCChangesMSZIPCtr1 mszip1;
 596|                [case(6|(DRSUAPI_COMPRESSION_TYPE_MSZIP<<16))]
->    drsuapi_DsGetNCChangesMSZIPCtr6 mszip6;
 597|                [case(1|(DRSUAPI_COMPRESSION_TYPE_XPRESS<<16))]
->    drsuapi_DsGetNCChangesXPRESSCtr1 xpress1;
 598|                [case(6|(DRSUAPI_COMPRESSION_TYPE_XPRESS<<16))]
->    drsuapi_DsGetNCChangesXPRESSCtr6 xpress6;
 599|        } drsuapi_DsGetNCChangesCompressedCtr;
 600|
 601|        typedef struct {
 602|                /*
 603|                 * this is a bit ugly, as the compression depends on the flags
 604|                 * in the DsBind(), but only w2k uses DsGetNCChangesReq5
 605|                 * and will get DsGetNCChangesCtr2 replies, and w2k only knowns
 606|                 * about MSZIP and level 1 replies
 607|                 */
 608|                [switch_is(1|(DRSUAPI_COMPRESSION_TYPE_MSZIP<<16))]
->    drsuapi_DsGetNCChangesCompressedCtr ctr;
 609|        } drsuapi_DsGetNCChangesCtr2;
 610|
 611|        typedef struct {
 612|                [range(0,6)] int32 level;
 613|                [range(2,3)] drsuapi_DsGetNCChangesCompressionType type;
 614|                [switch_is(level | (type<<16))] drsuapi_DsGetNCChangesCompressedCtr ctr;
 615|        } drsuapi_DsGetNCChangesCtr7;
 616|
 617|        typedef [switch_type(int32)] union {
 618|                [case(1)] drsuapi_DsGetNCChangesCtr1 ctr1;
 619|                [case(2)] drsuapi_DsGetNCChangesCtr2 ctr2;
 620|                [case(6)] drsuapi_DsGetNCChangesCtr6 ctr6;
 621|                [case(7)] drsuapi_DsGetNCChangesCtr7 ctr7;
 622|        } drsuapi_DsGetNCChangesCtr;
 623|
 624|        WERROR drsuapi_DsGetNCChanges(
 625|                [in] policy_handle *bind_handle,
 626|                [in,out,ref] int32 *level,
```

```
627|                [in,switch_is(*level)] drsuapi_DsGetNCChangesRequest req,
628|                [out,switch_is(*level)] drsuapi_DsGetNCChangesCtr ctr
629|                );
630|
631|        /*****************/
632|        /* Function 0x04 */
633|        typedef bitmap {
634|                DRSUAPI_DS_REPLICA_UPDATE_ASYNCHRONOUS_OPERATION        = 0x00000001,
635|                DRSUAPI_DS_REPLICA_UPDATE_WRITEABLE                     = 0x00000002,
636|                DRSUAPI_DS_REPLICA_UPDATE_ADD_REFERENCE                 = 0x00000004,
637|                DRSUAPI_DS_REPLICA_UPDATE_DELETE_REFERENCE              = 0x00000008,
638|                DRSUAPI_DS_REPLICA_UPDATE_0x00000010                    = 0x00000010
639|        } drsuapi_DsReplicaUpdateRefsOptions;
640|
641|        typedef struct {
642|                [ref] drsuapi_DsReplicaObjectIdentifier *naming_context;
643|                [ref,charset(DOS),string] uint8 *dest_dsa_dns_name;
644|                GUID dest_dsa_guid;
645|                drsuapi_DsReplicaUpdateRefsOptions options;
646|        } drsuapi_DsReplicaUpdateRefsRequest1;
647|
648|        typedef [switch_type(int32)] union {
649|                [case(1)] drsuapi_DsReplicaUpdateRefsRequest1 req1;
650|        } drsuapi_DsReplicaUpdateRefsRequest;
651|
652|        WERROR drsuapi_DsReplicaUpdateRefs(
653|                [in] policy_handle *bind_handle,
654|                [in] int32 level,
655|                [in,switch_is(level)] drsuapi_DsReplicaUpdateRefsRequest req
656|                );
657|
658|        /*****************/
659|        /* Function 0x05 */
660|        typedef bitmap {
661|                DRSUAPI_DS_REPLICA_ADD_ASYNCHRONOUS_OPERATION   = 0x00000001,
662|                DRSUAPI_DS_REPLICA_ADD_WRITEABLE                = 0x00000002
663|                /* TODO ... */
664|        } drsuapi_DsReplicaAddOptions;
665|
666|        WERROR DRSUAPI_REPLICA_ADD();
667|
668|        /*****************/
669|        /* Function 0x06 */
670|        typedef bitmap {
671|                DRSUAPI_DS_REPLICA_DELETE_ASYNCHRONOUS_OPERATION        = 0x00000001,
672|                DRSUAPI_DS_REPLICA_DELETE_WRITEABLE                     = 0x00000002
673|                /* TODO ... */
674|        } drsuapi_DsReplicaDeleteOptions;
675|
676|        WERROR DRSUAPI_REPLICA_DEL();
677|
678|        /*****************/
679|        /* Function 0x07 */
680|        typedef bitmap {
681|                DRSUAPI_DS_REPLICA_MODIFY_ASYNCHRONOUS_OPERATION        = 0x00000001,
682|                DRSUAPI_DS_REPLICA_MODIFY_WRITEABLE                     = 0x00000002
683|        } drsuapi_DsReplicaModifyOptions;
684|
685|        WERROR DRSUAPI_REPLICA_MODIFY();
686|
687|        /*****************/
688|        /* Function 0x08 */
689|        WERROR DRSUAPI_VERIFY_NAMES();
690|
691|        /*****************/
692|        /* Function 0x09 */
693|
694|        /* how are type 4 and 7 different from 2 and 3 ? */
695|        typedef [v1_enum] enum {
696|                DRSUAPI_DS_MEMBERSHIP_TYPE_UNIVERSAL_AND_DOMAIN_GROUPS = 1,
697|                DRSUAPI_DS_MEMBERSHIP_TYPE_DOMAIN_LOCAL_GROUPS  = 2,
698|                DRSUAPI_DS_MEMBERSHIP_TYPE_DOMAIN_GROUPS        = 3,
699|                DRSUAPI_DS_MEMBERSHIP_TYPE_DOMAIN_LOCAL_GROUPS2 = 4,
700|                DRSUAPI_DS_MEMBERSHIP_TYPE_UNIVERSAL_GROUPS     = 5,
701|                DRSUAPI_DS_MEMBERSHIP_TYPE_GROUPMEMBERS         = 6,
702|                DRSUAPI_DS_MEMBERSHIP_TYPE_DOMAIN_GROUPS2       = 7
703|        } drsuapi_DsMembershipType;
704|
705|        typedef struct {
706|                NTSTATUS status;
707|                [range(0,10000)] uint32 num_memberships;
708|                [range(0,10000)] uint32 num_sids;
709|                [size_is(num_memberships)] drsuapi_DsReplicaObjectIdentifier
->    **info_array;
710|                [size_is(num_memberships)] samr_GroupAttrs *group_attrs;
711|                [size_is(num_sids)] dom_sid28 **sids;
712|        } drsuapi_DsGetMembershipsCtr1;
```

```
713|
714|        typedef [switch_type(int32)] union {
715|                [case(1)] drsuapi_DsGetMembershipsCtr1 ctr1;
716|        } drsuapi_DsGetMembershipsCtr;
717|
718|        const int DRSUAPI_DS_MEMBERSHIP_FLAG_GROUP_ATTR = 0x1;
719|
720|        typedef struct {
721|                [range(1,10000)] uint32 count;
722|                [size_is(count)] drsuapi_DsReplicaObjectIdentifier **info_array;
723|                uint32 flags;
724|                drsuapi_DsMembershipType type;
725|                drsuapi_DsReplicaObjectIdentifier *domain;
726|        } drsuapi_DsGetMembershipsRequest1;
727|
728|        typedef [switch_type(int32)] union {
729|                [case(1)] drsuapi_DsGetMembershipsRequest1 req1;
730|        } drsuapi_DsGetMembershipsRequest;
731|
732|        WERROR drsuapi_DsGetMemberships(
733|                [in] policy_handle *bind_handle,
734|                [in,out] int32 level,
735|                [in] [switch_is(level)] drsuapi_DsGetMembershipsRequest req,
736|                [out] [switch_is(level)] drsuapi_DsGetMembershipsCtr ctr
737|                );
738|
739|        /*****************/
740|        /* Function 0x0a */
741|        WERROR DRSUAPI_INTER_DOMAIN_MOVE();
742|
743|        /*****************/
744|        /* Function 0x0b */
745|        typedef struct {
746|                uint32 unknown1;
747|                uint32 unknown2;
748|                [range(0,0x00A00000)] uint32 length;
749|                [size_is(length)] uint8 *data;
750|        } drsuapi_DsGetNT4ChangeLogRequest1;
751|
752|        typedef [switch_type(uint32)] union {
753|                [case(1)] drsuapi_DsGetNT4ChangeLogRequest1 req1;
754|        } drsuapi_DsGetNT4ChangeLogRequest;
755|
756|        typedef struct {
757|                [range(0,0x00A00000)] uint32 length1;
758|                [range(0,0x00A00000)] uint32 length2;
759|                hyper unknown1;
760|                NTTIME time2;
761|                hyper unknown3;
762|                NTTIME time4;
763|                hyper unknown5;
764|                NTTIME time6;
765|                NTSTATUS status;
766|                [size_is(length1)] uint8 *data1;
767|                [size_is(length2)] uint8 *data2;
768|        } drsuapi_DsGetNT4ChangeLogInfo1;
769|
770|        typedef [switch_type(uint32)] union {
771|                [case(1)] drsuapi_DsGetNT4ChangeLogInfo1 info1;
772|        } drsuapi_DsGetNT4ChangeLogInfo;
773|
774|        WERROR drsuapi_DsGetNT4ChangeLog(
775|                [in] policy_handle *bind_handle,
776|                [in,out] uint32 level,
777|                [in] [switch_is(level)] drsuapi_DsGetNT4ChangeLogRequest req,
778|                [out] [switch_is(level)] drsuapi_DsGetNT4ChangeLogInfo info
779|                );
780|
781|        /*****************/
782|        /* Function 0x0c */
783|        typedef [v1_enum] enum {
784|                DRSUAPI_DS_NAME_STATUS_OK                       = 0,
785|                DRSUAPI_DS_NAME_STATUS_RESOLVE_ERROR            = 1,
786|                DRSUAPI_DS_NAME_STATUS_NOT_FOUND                = 2,
787|                DRSUAPI_DS_NAME_STATUS_NOT_UNIQUE               = 3,
788|                DRSUAPI_DS_NAME_STATUS_NO_MAPPING               = 4,
789|                DRSUAPI_DS_NAME_STATUS_DOMAIN_ONLY              = 5,
790|                DRSUAPI_DS_NAME_STATUS_NO_SYNTACTICAL_MAPPING   = 6,
791|                DRSUAPI_DS_NAME_STATUS_TRUST_REFERRAL           = 7
792|        } drsuapi_DsNameStatus;
793|
794|        typedef [v1_enum] enum {
795|                DRSUAPI_DS_NAME_FLAG_NO_FLAGS                   = 0x0,
796|                DRSUAPI_DS_NAME_FLAG_SYNTACTICAL_ONLY           = 0x1,
797|                DRSUAPI_DS_NAME_FLAG_EVAL_AT_DC                 = 0x2,
798|                DRSUAPI_DS_NAME_FLAG_GCVERIFY                   = 0x4,
799|                DRSUAPI_DS_NAME_FLAG_TRUST_REFERRAL             = 0x8
```

```
800|        } drsuapi_DsNameFlags;
801|
802|        typedef [v1_enum] enum {
803|                DRSUAPI_DS_NAME_FORMAT_UKNOWN                   = 0,
804|                DRSUAPI_DS_NAME_FORMAT_FQDN_1779                = 1,
805|                DRSUAPI_DS_NAME_FORMAT_NT4_ACCOUNT              = 2,
806|                DRSUAPI_DS_NAME_FORMAT_DISPLAY                  = 3,
807|                DRSUAPI_DS_NAME_FORMAT_GUID                     = 6,
808|                DRSUAPI_DS_NAME_FORMAT_CANONICAL                = 7,
809|                DRSUAPI_DS_NAME_FORMAT_USER_PRINCIPAL           = 8,
810|                DRSUAPI_DS_NAME_FORMAT_CANONICAL_EX             = 9,
811|                DRSUAPI_DS_NAME_FORMAT_SERVICE_PRINCIPAL        = 10,
812|                DRSUAPI_DS_NAME_FORMAT_SID_OR_SID_HISTORY       = 11,
813|                DRSUAPI_DS_NAME_FORMAT_DNS_DOMAIN               = 12
814|        } drsuapi_DsNameFormat;
815|
816|        typedef struct {
817|                [string,charset(UTF16)] uint16 *str;
818|        } drsuapi_DsNameString;
819|
820|        typedef struct {
821|                uint32 codepage; /* 0x000004e4 - 1252 is german codepage*/
822|                uint32 language; /* 0x00000407 - german language ID*/
823|                drsuapi_DsNameFlags format_flags;
824|                drsuapi_DsNameFormat format_offered;
825|                drsuapi_DsNameFormat format_desired;
826|                [range(1,10000)] uint32 count;
827|                [size_is(count)] drsuapi_DsNameString *names;
828|        } drsuapi_DsNameRequest1;
829|
830|        typedef [switch_type(int32)] union {
831|                [case(1)] drsuapi_DsNameRequest1 req1;
832|        } drsuapi_DsNameRequest;
833|
834|        typedef struct {
835|                drsuapi_DsNameStatus status;
836|                [charset(UTF16),string] uint16 *dns_domain_name;
837|                [charset(UTF16),string] uint16 *result_name;
838|        } drsuapi_DsNameInfo1;
839|
840|        typedef struct {
841|                uint32 count;
842|                [size_is(count)] drsuapi_DsNameInfo1 *array;
843|        } drsuapi_DsNameCtr1;
844|
845|        typedef [switch_type(int32)] union {
846|                [case(1)] drsuapi_DsNameCtr1 *ctr1;
847|        } drsuapi_DsNameCtr;
848|
849|        WERROR drsuapi_DsCrackNames(
850|                [in] policy_handle *bind_handle,
851|                [in, out] int32 level,
852|                [in,switch_is(level)] drsuapi_DsNameRequest req,
853|                [out,switch_is(level)] drsuapi_DsNameCtr ctr
854|                );
855|
856|        /*****************/
857|        /* Function 0x0d */
858|        typedef [v1_enum] enum {
859|                DRSUAPI_DS_SPN_OPERATION_ADD    = 0,
860|                DRSUAPI_DS_SPN_OPERATION_REPLACE= 1,
861|                DRSUAPI_DS_SPN_OPERATION_DELETE = 2
862|        } drsuapi_DsSpnOperation;
863|
864|        typedef struct {
865|                drsuapi_DsSpnOperation operation;
866|                uint32 unknown1;
867|                [charset(UTF16),string] uint16 *object_dn;
868|                [range(0,10000)] uint32 count;
869|                [size_is(count)] drsuapi_DsNameString *spn_names;
870|        } drsuapi_DsWriteAccountSpnRequest1;
871|
872|        typedef [switch_type(int32)] union {
873|                [case(1)] drsuapi_DsWriteAccountSpnRequest1 req1;
874|        } drsuapi_DsWriteAccountSpnRequest;
875|
876|        typedef struct {
877|                WERROR status;
878|        } drsuapi_DsWriteAccountSpnResult1;
879|
880|        typedef [switch_type(int32)] union {
881|                [case(1)] drsuapi_DsWriteAccountSpnResult1 res1;
882|        } drsuapi_DsWriteAccountSpnResult;
883|
884|        WERROR drsuapi_DsWriteAccountSpn(
885|                [in] policy_handle *bind_handle,
886|                [in,out] int32 level,
```

```
887|                [in,switch_is(level)] drsuapi_DsWriteAccountSpnRequest req,
888|                [out,switch_is(level)] drsuapi_DsWriteAccountSpnResult res
889|                );
890|
891|        /*****************/
892|        /* Function 0x0e */
893|        typedef struct {
894|                [charset(UTF16),string] uint16 *server_dn;
895|                [charset(UTF16),string] uint16 *domain_dn;
896|                uint32 unknown; /* 0x000000001 */
897|        } drsuapi_DsRemoveDSServerRequest1;
898|
899|        typedef [switch_type(int32)] union {
900|                [case(1)] drsuapi_DsRemoveDSServerRequest1 req1;
901|        } drsuapi_DsRemoveDSServerRequest;
902|
903|        typedef struct {
904|                WERROR status;
905|        } drsuapi_DsRemoveDSServerResult1;
906|
907|        typedef [switch_type(int32)] union {
908|                [case(1)] drsuapi_DsRemoveDSServerResult1 res1;
909|        } drsuapi_DsRemoveDSServerResult;
910|
911|        WERROR drsuapi_DsRemoveDSServer(
912|                [in] policy_handle *bind_handle,
913|                [in,out] int32 level,
914|                [in,switch_is(level)] drsuapi_DsRemoveDSServerRequest req,
915|                [out,switch_is(level)] drsuapi_DsRemoveDSServerResult res
916|                );
917|
918|        /*****************/
919|        /* Function 0x0f */
920|        WERROR DRSUAPI_REMOVE_DS_DOMAIN();
921|
922|        /*****************/
923|        /* Function 0x10 */
924|        typedef struct {
925|                [charset(UTF16),string] uint16 *domain_name; /* netbios or dns */
926|                int32 level; /* specifies the switch level for the request */
927|        } drsuapi_DsGetDCInfoRequest1;
928|
929|        typedef [switch_type(int32)] union {
930|                [case(1)] drsuapi_DsGetDCInfoRequest1 req1;
931|        } drsuapi_DsGetDCInfoRequest;
932|
933|        typedef struct {
934|                [charset(UTF16),string] uint16 *netbios_name;
935|                [charset(UTF16),string] uint16 *dns_name;
936|                [charset(UTF16),string] uint16 *site_name;
937|                [charset(UTF16),string] uint16 *computer_dn;
938|                [charset(UTF16),string] uint16 *server_dn;
939|                uint32 is_pdc;
940|                uint32 is_enabled;
941|        } drsuapi_DsGetDCInfo1;
942|
943|        typedef struct {
944|                [range(0,10000)] uint32 count;
945|                [size_is(count)] drsuapi_DsGetDCInfo1 *array;
946|        } drsuapi_DsGetDCInfoCtr1;
947|
948|        typedef struct {
949|                [charset(UTF16),string] uint16 *netbios_name;
950|                [charset(UTF16),string] uint16 *dns_name;
951|                [charset(UTF16),string] uint16 *site_name;
952|                [charset(UTF16),string] uint16 *site_dn;
953|                [charset(UTF16),string] uint16 *computer_dn;
954|                [charset(UTF16),string] uint16 *server_dn;
955|                [charset(UTF16),string] uint16 *ntds_dn;
956|                uint32 is_pdc;
957|                uint32 is_enabled;
958|                uint32 is_gc;
959|                GUID site_guid;
960|                GUID computer_guid;
961|                GUID server_guid;
962|                GUID ntds_guid;
963|        } drsuapi_DsGetDCInfo2;
964|
965|        typedef struct {
966|                [range(0,10000)] uint32 count;
967|                [size_is(count)] drsuapi_DsGetDCInfo2 *array;
968|        } drsuapi_DsGetDCInfoCtr2;
969|
970|        /*
971|         * this represents an active connection to the
972|         * Directory System Agent (DSA)
973|         * this can be via LDAP or DRSUAPI
```

```
 974|         */
 975|        typedef struct {
 976|                [flag(NDR_BIG_ENDIAN)] ipv4address client_ip_address;
 977|                uint32 unknown2;
 978|                uint32 connection_time; /* in seconds */
 979|                uint32 unknown4;
 980|                uint32 unknown5;
 981|                uint32 unknown6;
 982|                /*
 983|                 * client_account can be the following:
 984|                 * "W2K3\Administrator"
 985|                 * "Administrator@W2K3"
 986|                 * "cn=Administrator,cn=Users,DC=w2k3,DC=vmnet1,DC=vm,DC=base"
 987|                 * ""
 988|                 * or NULL
 989|                 */
 990|                [charset(UTF16),string] uint16 *client_account;
 991|        } drsuapi_DsGetDCConnection01;
 992|
 993|        typedef struct {
 994|                [range(0,10000)] uint32 count;
 995|                [size_is(count)] drsuapi_DsGetDCConnection01 *array;
 996|        } drsuapi_DsGetDCConnectionCtr01;
 997|
 998|        typedef [v1_enum] enum {
 999|                DRSUAPI_DC_INFO_CTR_1  = 1,
1000|                DRSUAPI_DC_INFO_CTR_2  = 2,
1001|                DRSUAPI_DC_CONNECTION_CTR_01 = -1
1002|        } drsuapi_DsGetDCInfoCtrLevels;
1003|
1004|        typedef [switch_type(int32)] union {
1005|                [case(DRSUAPI_DC_INFO_CTR_1)]  drsuapi_DsGetDCInfoCtr1  ctr1;
1006|                [case(DRSUAPI_DC_INFO_CTR_2)]  drsuapi_DsGetDCInfoCtr2  ctr2;
1007|                [case(DRSUAPI_DC_CONNECTION_CTR_01)] drsuapi_DsGetDCConnectionCtr01
->    ctr01;
1008|        } drsuapi_DsGetDCInfoCtr;
1009|
1010|        WERROR drsuapi_DsGetDomainControllerInfo(
1011|                [in] policy_handle *bind_handle,
1012|                [in] int32 level,
1013|                [in,switch_is(level)] drsuapi_DsGetDCInfoRequest req,
1014|                [out] int32 level_out,
1015|                [out,switch_is(level_out)] drsuapi_DsGetDCInfoCtr ctr
1016|                );
1017|
1018|        /*****************/
1019|        /* Function 0x11 */
1020|        typedef [public,noprint] struct {
1021|                drsuapi_DsReplicaObjectListItem *next_object;
1022|                drsuapi_DsReplicaObject object;
1023|        } drsuapi_DsReplicaObjectListItem;
1024|
1025|        /*
1026|         * The DsAddEntry() call which creates a nTDSDSA object,
1027|         * also adds a servicePrincipalName in the following form
1028|         * to the computer account of the new domain controller
1029|         * referenced by the "serverReferenece" attribute.
1030|         *
1031|         * E3514235-4B06-11D1-AB04-00C04FC2DCD2/<new-ntdsdsa-object-guid-as-string>/<doma
->   in-dns-name>
1032|         *
1033|         * also note that the "serverReference" isn't added to the new object!
1034|         */
1035|        const char *DRSUAPI_NTDSDSA_KRB5_SERVICE_GUID = "E3514235-4B06-11D1-AB04-00C04FC2
->   DCD2";
1036|
1037|        /*
1038|         * please note the the current idl
1039|         * for DsAddEntry does only parse
1040|         * what I saw between 2 w2k3 boxes
1041|         * in my dssync experiments I got some other replies
1042|         * so all I want to say is that this is very incomplete yet...
1043|         * --metze
1044|         */
1045|        typedef struct {
1046|                drsuapi_DsReplicaObjectListItem first_object;
1047|        } drsuapi_DsAddEntryRequest2;
1048|
1049|        typedef [switch_type(int32)] union {
1050|                [case(2)] drsuapi_DsAddEntryRequest2 req2;
1051|        } drsuapi_DsAddEntryRequest;
1052|
1053|        typedef struct {
1054|                uint32 unknown1;
1055|                WERROR status;
1056|                uint32 unknown2;
1057|                uint16 unknown3;
```

```
1058|        } drsuapi_DsAddEntryErrorInfoX;
1059|
1060|        typedef struct {
1061|                [range(0,10485760)] uint32 size;
1062|                [size_is(size)] uint8 *data;
1063|        } drsuapi_DsAddEntryExtraErrorBuffer;
1064|
1065|        typedef struct {
1066|                drsuapi_DsAddEntryErrorInfoX error;
1067|                drsuapi_DsAttributeId attid;
1068|                uint32 unknown2;
1069|                drsuapi_DsAddEntryExtraErrorBuffer buffer;
1070|        } drsuapi_DsAddEntryExtraError1;
1071|
1072|        typedef /*[noprint]*/ struct {
1073|                drsuapi_DsAddEntryErrorListItem1 *next;
1074|                drsuapi_DsAddEntryExtraError1 error;
1075|        } drsuapi_DsAddEntryErrorListItem1;
1076|
1077|        typedef struct {
1078|                drsuapi_DsReplicaObjectIdentifier *id;
1079|                WERROR status;
1080|                drsuapi_DsAddEntryErrorListItem1 first;
1081|        } drsuapi_DsAddEntryErrorInfo1;
1082|
1083|        typedef [switch_type(uint32)] union {
1084|                [case(1)] drsuapi_DsAddEntryErrorInfo1 error1;
1085|/*              [case(2)] drsuapi_DsAddEntryErrorInfo2 error2;
1086|                [case(3)] drsuapi_DsAddEntryErrorInfo3 error3;
1087|*/              [case(4)] drsuapi_DsAddEntryErrorInfoX errorX;
1088|                [case(5)] drsuapi_DsAddEntryErrorInfoX errorX;
1089|                [case(6)] drsuapi_DsAddEntryErrorInfoX errorX;
1090|                [case(7)] drsuapi_DsAddEntryErrorInfoX errorX;
1091|        } drsuapi_DsAddEntryErrorInfo;
1092|
1093|        typedef struct {
1094|                WERROR status;
1095|                uint32 level;
1096|                [switch_is(level)] drsuapi_DsAddEntryErrorInfo *info;
1097|        } drsuapi_DsAddEntryError1;
1098|
1099|        typedef [switch_type(uint32)] union {
1100|                [case(1)] drsuapi_DsAddEntryError1 info1;
1101|        } drsuapi_DsAddEntryError;
1102|
1103|        typedef struct {
1104|                GUID guid;
1105|                dom_sid28 sid;
1106|        } drsuapi_DsReplicaObjectIdentifier2;
1107|
1108|        typedef struct {
1109|                drsuapi_DsReplicaObjectIdentifier *id;
1110|                uint32 unknown1;
1111|                drsuapi_DsAddEntryErrorInfoX error;
1112|                [range(0,10000)] uint32 count;
1113|                [size_is(count)] drsuapi_DsReplicaObjectIdentifier2 *objects;
1114|        } drsuapi_DsAddEntryCtr2;
1115|
1116|        typedef struct {
1117|                drsuapi_DsReplicaObjectIdentifier *id;
1118|                uint32 level;
1119|                [switch_is(level)] drsuapi_DsAddEntryError *error;
1120|                [range(0,10000)] uint32 count;
1121|                [size_is(count)] drsuapi_DsReplicaObjectIdentifier2 *objects;
1122|        } drsuapi_DsAddEntryCtr3;
1123|
1124|        typedef [switch_type(int32)] union {
1125|                [case(2)] drsuapi_DsAddEntryCtr2 ctr2;
1126|                [case(3)] drsuapi_DsAddEntryCtr3 ctr3;
1127|        } drsuapi_DsAddEntryCtr;
1128|
1129|        [public] WERROR drsuapi_DsAddEntry(
1130|                [in] policy_handle *bind_handle,
1131|                [in,out] int32 level,
1132|                [in,switch_is(level)] drsuapi_DsAddEntryRequest req,
1133|                [out,switch_is(level)] drsuapi_DsAddEntryCtr ctr
1134|                );
1135|
1136|        /*****************/
1137|        /* Function 0x12 */
1138|        WERROR DRSUAPI_EXECUTE_KCC();
1139|
1140|        /*****************/
1141|        /* Function 0x13 */
1142|        typedef [v1_enum] enum {
1143|                DRSUAPI_DS_REPLICA_GET_INFO                             = 1,
1144|                DRSUAPI_DS_REPLICA_GET_INFO2                            = 2
```

```
1145|        } drsuapi_DsReplicaGetInfoLevel;
1146|
1147|        typedef [v1_enum] enum {
1148|                DRSUAPI_DS_REPLICA_INFO_NEIGHBORS                       = 0,
1149|                DRSUAPI_DS_REPLICA_INFO_CURSORS                         = 1,
1150|                DRSUAPI_DS_REPLICA_INFO_OBJ_METADATA                    = 2,
1151|                DRSUAPI_DS_REPLICA_INFO_KCC_DSA_CONNECT_FAILURES        = 3,
1152|                DRSUAPI_DS_REPLICA_INFO_KCC_DSA_LINK_FAILURES           = 4,
1153|                DRSUAPI_DS_REPLICA_INFO_PENDING_OPS                     = 5,
1154|                DRSUAPI_DS_REPLICA_INFO_ATTRIBUTE_VALUE_METADATA        = 6,
1155|                DRSUAPI_DS_REPLICA_INFO_CURSORS2                        = 7,
1156|                DRSUAPI_DS_REPLICA_INFO_CURSORS3                        = 8,
1157|                DRSUAPI_DS_REPLICA_INFO_OBJ_METADATA2                   = 9,
1158|                DRSUAPI_DS_REPLICA_INFO_ATTRIBUTE_VALUE_METADATA2       = 10,
1159|                DRSUAPI_DS_REPLICA_INFO_NEIGHBORS02                     = -2,
1160|                DRSUAPI_DS_REPLICA_INFO_CONNECTIONS04                   = -4,
1161|                DRSUAPI_DS_REPLICA_INFO_CURSORS05                       = -5,
1162|                DRSUAPI_DS_REPLICA_INFO_06                              = -6
1163|        } drsuapi_DsReplicaInfoType;
1164|
1165|        typedef struct {
1166|                drsuapi_DsReplicaInfoType info_type;
1167|                [charset(UTF16),string] uint16 *object_dn;
1168|                GUID guid1;
1169|        } drsuapi_DsReplicaGetInfoRequest1;
1170|
1171|        typedef struct {
1172|                drsuapi_DsReplicaInfoType info_type;
1173|                [charset(UTF16),string] uint16 *object_dn;
1174|                GUID guid1;
1175|                uint32 unknown1;
1176|                [charset(UTF16),string] uint16 *string1;
1177|                [charset(UTF16),string] uint16 *string2;
1178|                uint32 unknown2;
1179|        } drsuapi_DsReplicaGetInfoRequest2;
1180|
1181|        typedef [switch_type(drsuapi_DsReplicaGetInfoLevel)] union {
1182|                [case(DRSUAPI_DS_REPLICA_GET_INFO)] drsuapi_DsReplicaGetInfoRequest1
->    req1;
1183|                [case(DRSUAPI_DS_REPLICA_GET_INFO2)] drsuapi_DsReplicaGetInfoRequest2
->    req2;
1184|        } drsuapi_DsReplicaGetInfoRequest;
1185|
1186|        typedef struct {
1187|                [charset(UTF16),string] uint16 *naming_context_dn;
1188|                [charset(UTF16),string] uint16 *source_dsa_obj_dn;
1189|                [charset(UTF16),string] uint16 *source_dsa_address;
1190|                [charset(UTF16),string] uint16 *transport_obj_dn;
1191|                drsuapi_DsReplicaNeighbourFlags replica_flags;
1192|                uint32 reserved;
1193|                GUID naming_context_obj_guid;
1194|                GUID source_dsa_obj_guid;
1195|                GUID source_dsa_invocation_id;
1196|                GUID transport_obj_guid;
1197|                hyper tmp_highest_usn;
1198|                hyper highest_usn;
1199|                NTTIME last_success;
1200|                NTTIME last_attempt;
1201|                WERROR result_last_attempt;
1202|                uint32 consecutive_sync_failures;
1203|        } drsuapi_DsReplicaNeighbour;
1204|
1205|        typedef struct {
1206|                uint32 count;
1207|                uint32 reserved;
1208|                [size_is(count)] drsuapi_DsReplicaNeighbour array[];
1209|        } drsuapi_DsReplicaNeighbourCtr;
1210|
1211|        typedef struct {
1212|                uint32 count;
1213|                uint32 reserved;
1214|                [size_is(count)] drsuapi_DsReplicaCursor array[];
1215|        } drsuapi_DsReplicaCursorCtr;
1216|
1217|        typedef struct {
1218|                [charset(UTF16),string] uint16 *attribute_name;
1219|                uint32 version;
1220|                NTTIME originating_change_time;
1221|                GUID originating_invocation_id;
1222|                hyper originating_usn;
1223|                hyper local_usn;
1224|        } drsuapi_DsReplicaObjMetaData;
1225|
1226|        typedef struct {
1227|                uint32 count;
1228|                uint32 reserved;
1229|                [size_is(count)] drsuapi_DsReplicaObjMetaData array[];
```

```
1230|        } drsuapi_DsReplicaObjMetaDataCtr;
1231|
1232|        typedef struct {
1233|                [charset(UTF16),string] uint16 *dsa_obj_dn;
1234|                GUID dsa_obj_guid;
1235|                NTTIME first_failure;
1236|                uint32 num_failures;
1237|                WERROR last_result;
1238|        } drsuapi_DsReplicaKccDsaFailure;
1239|
1240|        typedef struct {
1241|                uint32 count;
1242|                uint32 reserved;
1243|                [size_is(count)] drsuapi_DsReplicaKccDsaFailure array[];
1244|        } drsuapi_DsReplicaKccDsaFailuresCtr;
1245|
1246|        typedef enum {
1247|                DRSUAPI_DS_REPLICA_OP_TYPE_SYNC         = 0,
1248|                DRSUAPI_DS_REPLICA_OP_TYPE_ADD          = 1,
1249|                DRSUAPI_DS_REPLICA_OP_TYPE_DELETE       = 2,
1250|                DRSUAPI_DS_REPLICA_OP_TYPE_MODIFY       = 3,
1251|                DRSUAPI_DS_REPLICA_OP_TYPE_UPDATE_REFS  = 4
1252|        } drsuapi_DsReplicaOpType;
1253|
1254|        typedef [switch_type(drsuapi_DsReplicaOpType)] union {
1255|                [case(DRSUAPI_DS_REPLICA_OP_TYPE_SYNC)]
->    drsuapi_DsReplicaSyncOptions sync;
1256|                [case(DRSUAPI_DS_REPLICA_OP_TYPE_ADD)]
->    drsuapi_DsReplicaAddOptions add;
1257|                [case(DRSUAPI_DS_REPLICA_OP_TYPE_DELETE)]
->    drsuapi_DsReplicaDeleteOptions delete;
1258|                [case(DRSUAPI_DS_REPLICA_OP_TYPE_MODIFY)]
->    drsuapi_DsReplicaModifyOptions modify;
1259|                [case(DRSUAPI_DS_REPLICA_OP_TYPE_UPDATE_REFS)]
->    drsuapi_DsReplicaUpdateRefsOptions update_refs;
1260|                [default] uint32 unknown;
1261|        } drsuapi_DsRplicaOpOptions;
1262|
1263|        typedef struct {
1264|                NTTIME operation_start;
1265|                uint32 serial_num; /* unique till reboot */
1266|                uint32 priority;
1267|                drsuapi_DsReplicaOpType operation_type;
1268|                [switch_is(operation_type)] drsuapi_DsRplicaOpOptions options;
1269|                [charset(UTF16),string] uint16 *nc_dn;
1270|                [charset(UTF16),string] uint16 *remote_dsa_obj_dn;
1271|                [charset(UTF16),string] uint16 *remote_dsa_address;
1272|                GUID nc_obj_guid;
1273|                GUID remote_dsa_obj_guid;
1274|        } drsuapi_DsReplicaOp;
1275|
1276|        typedef struct {
1277|                NTTIME time;
1278|                uint32 count;
1279|                [size_is(count)] drsuapi_DsReplicaOp array[];
1280|        } drsuapi_DsReplicaOpCtr;
1281|
1282|        typedef struct {
1283|                [charset(UTF16),string] uint16 *attribute_name;
1284|                [charset(UTF16),string] uint16 *object_dn;
1285|                [value(ndr_size_DATA_BLOB(0,binary,0))] uint32 __ndr_size_binary;
1286|                DATA_BLOB *binary;
1287|                NTTIME deleted;
1288|                NTTIME created;
1289|                uint32 version;
1290|                NTTIME originating_change_time;
1291|                GUID originating_invocation_id;
1292|                hyper originating_usn;
1293|                hyper local_usn;
1294|        } drsuapi_DsReplicaAttrValMetaData;
1295|
1296|        typedef struct {
1297|                uint32 count;
1298|                int32 enumeration_context;
1299|                [size_is(count)] drsuapi_DsReplicaAttrValMetaData array[];
1300|        } drsuapi_DsReplicaAttrValMetaDataCtr;
1301|
1302|        typedef struct {
1303|                uint32 count;
1304|                int32 enumeration_context;
1305|                [size_is(count)] drsuapi_DsReplicaCursor2 array[];
1306|        } drsuapi_DsReplicaCursor2Ctr;
1307|
1308|        typedef struct {
1309|                GUID source_dsa_invocation_id;
1310|                hyper highest_usn;
1311|                NTTIME last_sync_success;
```

```
1312|                [charset(UTF16),string] uint16 *source_dsa_obj_dn;
1313|        } drsuapi_DsReplicaCursor3;
1314|
1315|        typedef struct {
1316|                uint32 count;
1317|                int32 enumeration_context;
1318|                [size_is(count)] drsuapi_DsReplicaCursor3 array[];
1319|        } drsuapi_DsReplicaCursor3Ctr;
1320|
1321|        typedef struct {
1322|                [charset(UTF16),string] uint16 *attribute_name;
1323|                uint32 version;
1324|                NTTIME originating_change_time;
1325|                GUID originating_invocation_id;
1326|                hyper originating_usn;
1327|                hyper local_usn;
1328|                [charset(UTF16),string] uint16 *originating_dsa_dn;
1329|        } drsuapi_DsReplicaObjMetaData2;
1330|
1331|        typedef struct {
1332|                uint32 count;
1333|                int32 enumeration_context;
1334|                [size_is(count)] drsuapi_DsReplicaObjMetaData2 array[];
1335|        } drsuapi_DsReplicaObjMetaData2Ctr;
1336|
1337|        typedef struct {
1338|                [charset(UTF16),string] uint16 *attribute_name;
1339|                [charset(UTF16),string] uint16 *object_dn;
1340|                [value(ndr_size_DATA_BLOB(0,binary,0))] uint32 __ndr_size_binary;
1341|                DATA_BLOB *binary;
1342|                NTTIME deleted;
1343|                NTTIME created;
1344|                uint32 version;
1345|                NTTIME originating_change_time;
1346|                GUID originating_invocation_id;
1347|                hyper originating_usn;
1348|                hyper local_usn;
1349|                [charset(UTF16),string] uint16 *originating_dsa_dn;
1350|        } drsuapi_DsReplicaAttrValMetaData2;
1351|
1352|        typedef struct {
1353|                uint32 count;
1354|                int32 enumeration_context;
1355|                [size_is(count)] drsuapi_DsReplicaAttrValMetaData2 array[];
1356|        } drsuapi_DsReplicaAttrValMetaData2Ctr;
1357|
1358|        typedef struct {
1359|                hyper u1; /* session number? */
1360|                uint32 u2;
1361|                uint32 u3;
1362|                GUID bind_guid;
1363|                NTTIME_1sec bind_time;
1364|                [flag(NDR_BIG_ENDIAN)] ipv4address client_ip_address;
1365|                uint32 u5; /* this is the same value the client used as u1 in the
->    DsBindInfoX struct */
1366|        } drsuapi_DsReplicaConnection04;
1367|
1368|        typedef struct {
1369|                [range(0,10000)] uint32 count;
1370|                uint32 reserved;
1371|                [size_is(count)] drsuapi_DsReplicaConnection04 array[];
1372|        } drsuapi_DsReplicaConnection04Ctr;
1373|
1374|        typedef struct {
1375|                [charset(UTF16),string] uint16 *str1;
1376|                uint32 u1;
1377|                uint32 u2;
1378|                uint32 u3;
1379|                uint32 u4;
1380|                uint32 u5;
1381|                hyper u6;
1382|                uint32 u7;
1383|        } drsuapi_DsReplica06;
1384|
1385|        typedef struct {
1386|                [range(0,256)] uint32 count;
1387|                uint32 reserved;
1388|                [size_is(count)] drsuapi_DsReplica06 array[];
1389|        } drsuapi_DsReplica06Ctr;
1390|
1391|        typedef [switch_type(drsuapi_DsReplicaInfoType)] union {
1392|                [case(DRSUAPI_DS_REPLICA_INFO_NEIGHBORS)] drsuapi_DsReplicaNeighbourCtr
->    *neighbours;
1393|                [case(DRSUAPI_DS_REPLICA_INFO_CURSORS)] drsuapi_DsReplicaCursorCtr
->    *cursors;
1394|                [case(DRSUAPI_DS_REPLICA_INFO_OBJ_METADATA)]
->    drsuapi_DsReplicaObjMetaDataCtr *objmetadata;
```

```
1395|                [case(DRSUAPI_DS_REPLICA_INFO_KCC_DSA_CONNECT_FAILURES)]
->    drsuapi_DsReplicaKccDsaFailuresCtr *connectfailures;
1396|                [case(DRSUAPI_DS_REPLICA_INFO_KCC_DSA_LINK_FAILURES)]
->    drsuapi_DsReplicaKccDsaFailuresCtr *linkfailures;
1397|                [case(DRSUAPI_DS_REPLICA_INFO_PENDING_OPS)] drsuapi_DsReplicaOpCtr
->    *pendingops;
1398|                [case(DRSUAPI_DS_REPLICA_INFO_ATTRIBUTE_VALUE_METADATA)]
->    drsuapi_DsReplicaAttrValMetaDataCtr *attrvalmetadata;
1399|                [case(DRSUAPI_DS_REPLICA_INFO_CURSORS2)] drsuapi_DsReplicaCursor2Ctr
->    *cursors2;
1400|                [case(DRSUAPI_DS_REPLICA_INFO_CURSORS3)] drsuapi_DsReplicaCursor3Ctr
->    *cursors3;
1401|                [case(DRSUAPI_DS_REPLICA_INFO_OBJ_METADATA2)]
->    drsuapi_DsReplicaObjMetaData2Ctr *objmetadata2;
1402|                [case(DRSUAPI_DS_REPLICA_INFO_ATTRIBUTE_VALUE_METADATA2)]
->    drsuapi_DsReplicaAttrValMetaData2Ctr *attrvalmetadata2;
1403|                [case(DRSUAPI_DS_REPLICA_INFO_NEIGHBORS02)] drsuapi_DsReplicaNeighbourCtr
->    *neighbours02;
1404|                [case(DRSUAPI_DS_REPLICA_INFO_CONNECTIONS04)]
->    drsuapi_DsReplicaConnection04Ctr *connections04;
1405|                [case(DRSUAPI_DS_REPLICA_INFO_CURSORS05)] drsuapi_DsReplicaCursorCtrEx
->    *cursors05;
1406|                [case(DRSUAPI_DS_REPLICA_INFO_06)] drsuapi_DsReplica06Ctr *i06;
1407|        } drsuapi_DsReplicaInfo;
1408|
1409|        WERROR drsuapi_DsReplicaGetInfo(
1410|                [in] policy_handle *bind_handle,
1411|                [in] drsuapi_DsReplicaGetInfoLevel level,
1412|                [in,switch_is(level)] drsuapi_DsReplicaGetInfoRequest req,
1413|                [out] drsuapi_DsReplicaInfoType info_type,
1414|                [out,switch_is(info_type)] drsuapi_DsReplicaInfo info
1415|                );
1416|
1417|        /*****************/
1418|        /* Function 0x14 */
1419|        WERROR DRSUAPI_ADD_SID_HISTORY();
1420|
1421|        /*****************/
1422|        /* Function 0x15 */
1423|
1424|        typedef struct {
1425|                [range(0,10000)] uint32 num_entries;
1426|                [size_is(num_entries)] drsuapi_DsGetMembershipsCtr1 **ctrl_array;
1427|        } drsuapi_DsGetMemberships2Ctr1;
1428|
1429|        typedef [switch_type(int32)] union {
1430|                [case(1)] drsuapi_DsGetMembershipsCtr1 ctr1;
1431|        } drsuapi_DsGetMemberships2Ctr;
1432|
1433|        typedef struct {
1434|                [range(1,10000)] uint32 num_req;
1435|                [size_is(num_req)] drsuapi_DsGetMembershipsRequest1 **req_array;
1436|        } drsuapi_DsGetMemberships2Request1;
1437|
1438|        typedef [switch_type(int32)] union {
1439|                [case(1)] drsuapi_DsGetMemberships2Request1 req1;
1440|        } drsuapi_DsGetMemberships2Request;
1441|
1442|        WERROR drsuapi_DsGetMemberships2(
1443|                [in] policy_handle *bind_handle,
1444|                [in,out] int32 level,
1445|                [in] [switch_is(level)] drsuapi_DsGetMemberships2Request req,
1446|                [out] [switch_is(level)] drsuapi_DsGetMemberships2Ctr ctr
1447|                );
1448|
1449|
1450|        /*****************/
1451|        /* Function 0x16 */
1452|        WERROR DRSUAPI_REPLICA_VERIFY_OBJECTS();
1453|
1454|        /*****************/
1455|        /* Function 0x17 */
1456|        WERROR DRSUAPI_GET_OBJECT_EXISTENCE();
1457|
1458|        /*****************/
1459|        /* Function 0x18 */
1460|        WERROR DRSUAPI_QUERY_SITES_BY_COST();
1461|}
```

A3. source/dsdb/samdb/samdb.h

source/dsdb/samdb/samdb.h:

```
   1|/*
   2|   Unix SMB/CIFS implementation.
   3|
   4|   interface functions for the sam database
   5|
   6|   Copyright (C) Andrew Tridgell 2004
   7|
   8|   This program is free software; you can redistribute it and/or modify
   9|   it under the terms of the GNU General Public License as published by
  10|   the Free Software Foundation; either version 2 of the License, or
  11|   (at your option) any later version.
  12|
  13|   This program is distributed in the hope that it will be useful,
  14|   but WITHOUT ANY WARRANTY; without even the implied warranty of
  15|   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  16|   GNU General Public License for more details.
  17|
  18|   You should have received a copy of the GNU General Public License
  19|   along with this program; if not, write to the Free Software
  20|   Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
  21|*/
  22|
  23|#ifndef __SAMDB_H__
  24|#define __SAMDB_H__
  25|
  26|struct auth_session_info;
  27|struct dsdb_control_current_partition;
  28|struct dsdb_extended_replicated_object;
  29|struct dsdb_extended_replicated_objects;
  30|
  31|#include "librpc/gen_ndr/security.h"
  32|#include "lib/ldb/include/ldb.h"
  33|#include "librpc/gen_ndr/samr.h"
  34|#include "librpc/gen_ndr/drsuapi.h"
  35|#include "librpc/gen_ndr/drsblobs.h"
  36|#include "dsdb/schema/schema.h"
  37|#include "dsdb/samdb/samdb_proto.h"
  38|
  39|#define DSDB_CONTROL_CURRENT_PARTITION_OID "1.3.6.1.4.1.7165.4.3.2"
  40|struct dsdb_control_current_partition {
  41|        /*
  42|         * this is the version of the dsdb_control_current_partition
  43|         * version 0: initial implementation
  44|         */
  45|#define DSDB_CONTROL_CURRENT_PARTITION_VERSION 0
  46|        uint32_t version;
  47|
  48|        struct ldb_dn *dn;
  49|
  50|        const char *backend;
  51|
  52|        struct ldb_module *module;
  53|};
  54|
  55|#define DSDB_EXTENDED_REPLICATED_OBJECTS_OID "1.3.6.1.4.1.7165.4.4.1"
  56|struct dsdb_extended_replicated_object {
  57|        struct ldb_message *msg;
  58|        struct ldb_val guid_value;
  59|        const char *when_changed;
  60|        struct replPropertyMetaDataBlob *meta_data;
  61|};
  62|
  63|struct dsdb_extended_replicated_objects {
  64|        /*
  65|         * this is the version of the dsdb_extended_replicated_objects
  66|         * version 0: initial implementation
  67|         */
  68|#define DSDB_EXTENDED_REPLICATED_OBJECTS_VERSION 0
  69|        uint32_t version;
  70|
  71|        struct ldb_dn *partition_dn;
  72|
  73|        const struct repsFromTo1 *source_dsa;
  74|        const struct drsuapi_DsReplicaCursor2CtrEx *uptodateness_vector;
  75|
  76|        uint32_t num_objects;
  77|        struct dsdb_extended_replicated_object *objects;
  78|};
  79|
```

```
80|struct dsdb_schema_fsmo {
81|        bool we_are_master;
82|        struct ldb_dn *master_dn;
83|};
84|
85|struct dsdb_naming_fsmo {
86|        bool we_are_master;
87|        struct ldb_dn *master_dn;
88|};
89|
90|struct dsdb_pdc_fsmo {
91|        bool we_are_master;
92|        struct ldb_dn *master_dn;
93|};
94|
95|#endif /* __SAMDB_H__ */
```

A4. source/dsdb/schema/schema.h

source/dsdb/schema/schema.h:

```
/*
   Unix SMB/CIFS mplementation.
   DSDB schema header

   Copyright (C) Stefan Metzmacher <metze@samba.org> 2006

   This program is free software; you can redistribute it and/or modify
   it under the terms of the GNU General Public License as published by
   the Free Software Foundation; either version 2 of the License, or
   (at your option) any later version.

   This program is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   GNU General Public License for more details.

   You should have received a copy of the GNU General Public License
   along with this program; if not, write to the Free Software
   Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

*/

#ifndef _DSDB_SCHEMA_H
#define _DSDB_SCHEMA_H

struct dsdb_attribute;
struct dsdb_class;
struct dsdb_schema;

struct dsdb_syntax {
        const char *name;
        const char *ldap_oid;
        uint32_t oMSyntax;
        struct ldb_val oMObjectClass;
        const char *attributeSyntax_oid;

        WERROR (*drsuapi_to_ldb)(const struct dsdb_schema *schema,
                                 const struct dsdb_attribute *attr,
                                 const struct drsuapi_DsReplicaAttribute *in,
                                 TALLOC_CTX *mem_ctx,
                                 struct ldb_message_element *out);
        WERROR (*ldb_to_drsuapi)(const struct dsdb_schema *schema,
                                 const struct dsdb_attribute *attr,
                                 const struct ldb_message_element *in,
                                 TALLOC_CTX *mem_ctx,
                                 struct drsuapi_DsReplicaAttribute *out);
};

struct dsdb_attribute {
        struct dsdb_attribute *prev, *next;

        const char *cn;
        const char *lDAPDisplayName;
        const char *attributeID_oid;
        uint32_t attributeID_id;
        struct GUID schemaIDGUID;
        uint32_t mAPIID;

        struct GUID attributeSecurityGUID;

        uint32_t searchFlags;
        uint32_t systemFlags;
        BOOL isMemberOfPartialAttributeSet;
        uint32_t linkID;

        const char *attributeSyntax_oid;
        uint32_t attributeSyntax_id;
        uint32_t oMSyntax;
        struct ldb_val oMObjectClass;

        BOOL isSingleValued;
        uint32_t rangeLower;
        uint32_t rangeUpper;
        BOOL extendedCharsAllowed;

        uint32_t schemaFlagsEx;
        struct ldb_val msDs_Schema_Extensions;

        BOOL showInAdvancedViewOnly;
```

```
 80|        const char *adminDisplayName;
 81|        const char *adminDescription;
 82|        const char *classDisplayName;
 83|        BOOL isEphemeral;
 84|        BOOL isDefunct;
 85|        BOOL systemOnly;
 86|
 87|        /* internal stuff */
 88|        const struct dsdb_syntax *syntax;
 89|};
 90|
 91|struct dsdb_class {
 92|        struct dsdb_class *prev, *next;
 93|
 94|        const char *cn;
 95|        const char *lDAPDisplayName;
 96|        const char *governsID_oid;
 97|        uint32_t governsID_id;
 98|        struct GUID schemaIDGUID;
 99|
100|        uint32_t objectClassCategory;
101|        const char *rDNAttID;
102|        const char *defaultObjectCategory;
103|
104|        const char *subClassOf;
105|
106|        const char **systemAuxiliaryClass;
107|        const char **systemPossSuperiors;
108|        const char **systemMustContain;
109|        const char **systemMayContain;
110|
111|        const char **auxiliaryClass;
112|        const char **possSuperiors;
113|        const char **mustContain;
114|        const char **mayContain;
115|
116|        const char *defaultSecurityDescriptor;
117|
118|        uint32_t schemaFlagsEx;
119|        struct ldb_val msDs_Schema_Extensions;
120|
121|        BOOL showInAdvancedViewOnly;
122|        const char *adminDisplayName;
123|        const char *adminDescription;
124|        const char *classDisplayName;
125|        BOOL defaultHidingValue;
126|        BOOL isDefunct;
127|        BOOL systemOnly;
128|};
129|
130|struct dsdb_schema_oid_prefix {
131|        uint32_t id;
132|        const char *oid;
133|        size_t oid_len;
134|};
135|
136|struct dsdb_schema {
137|        uint32_t num_prefixes;
138|        struct dsdb_schema_oid_prefix *prefixes;
139|
140|        /*
141|         * the last element of the prefix mapping table isn't a oid,
142|         * it starts with 0xFF and has 21 bytes and is maybe a schema
143|         * version number
144|         *
145|         * this is the content of the schemaInfo attribute of the
146|         * Schema-Partition head object.
147|         */
148|        const char *schema_info;
149|
150|        struct dsdb_attribute *attributes;
151|        struct dsdb_class *classes;
152|};
153|
154|#endif /* _DSDB_SCHEMA_H */
```

A5. source/libnet/libnet_become_dc.h

source/libnet/libnet_become_dc.h:

```
/*
   Unix SMB/CIFS implementation.

   Copyright (C) Stefan Metzmacher <metze@samba.org> 2006

   This program is free software; you can redistribute it and/or modify
   it under the terms of the GNU General Public License as published by
   the Free Software Foundation; either version 2 of the License, or
   (at your option) any later version.

   This program is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   GNU General Public License for more details.

   You should have received a copy of the GNU General Public License
   along with this program; if not, write to the Free Software
   Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
*/

#ifndef _LIBNET_BECOME_DC_H
#define _LIBNET_BECOME_DC_H

#include "librpc/gen_ndr/drsuapi.h"

struct libnet_BecomeDC_Domain {
        /* input */
        const char *dns_name;
        const char *netbios_name;
        const struct dom_sid *sid;

        /* constructed */
        struct GUID guid;
        const char *dn_str;
        uint32_t behavior_version;
        uint32_t w2k3_update_revision;
};

struct libnet_BecomeDC_Forest {
        /* constructed */
        const char *dns_name;
        const char *root_dn_str;
        const char *config_dn_str;
        uint32_t crossref_behavior_version;
        const char *schema_dn_str;
        uint32_t schema_object_version;
};

struct libnet_BecomeDC_SourceDSA {
        /* input */
        const char *address;

        /* constructed */
        const char *dns_name;
        const char *netbios_name;
        const char *site_name;
        const char *server_dn_str;
        const char *ntds_dn_str;
};

struct libnet_BecomeDC_CheckOptions {
        const struct libnet_BecomeDC_Domain *domain;
        const struct libnet_BecomeDC_Forest *forest;
        const struct libnet_BecomeDC_SourceDSA *source_dsa;
};

struct libnet_BecomeDC_DestDSA {
        /* input */
        const char *netbios_name;

        /* constructed */
        const char *dns_name;
        const char *site_name;
        struct GUID site_guid;
        const char *computer_dn_str;
        const char *server_dn_str;
        const char *ntds_dn_str;
        struct GUID ntds_guid;
        struct GUID invocation_id;
```

```
 80|        uint32_t user_account_control;
 81|};
 82|
 83|struct libnet_BecomeDC_PrepareDB {
 84|        const struct libnet_BecomeDC_Domain *domain;
 85|        const struct libnet_BecomeDC_Forest *forest;
 86|        const struct libnet_BecomeDC_SourceDSA *source_dsa;
 87|        const struct libnet_BecomeDC_DestDSA *dest_dsa;
 88|};
 89|
 90|struct libnet_BecomeDC_StoreChunk;
 91|
 92|struct libnet_BecomeDC_Partition {
 93|        struct drsuapi_DsReplicaObjectIdentifier nc;
 94|        struct GUID destination_dsa_guid;
 95|        struct GUID source_dsa_guid;
 96|        struct GUID source_dsa_invocation_id;
 97|        struct drsuapi_DsReplicaHighWaterMark highwatermark;
 98|        uint32_t replica_flags;
 99|
100|        NTSTATUS (*store_chunk)(void *private_data,
101|                               const struct libnet_BecomeDC_StoreChunk *info);
102|};
103|
104|struct libnet_BecomeDC_StoreChunk {
105|        const struct libnet_BecomeDC_Domain *domain;
106|        const struct libnet_BecomeDC_Forest *forest;
107|        const struct libnet_BecomeDC_SourceDSA *source_dsa;
108|        const struct libnet_BecomeDC_DestDSA *dest_dsa;
109|        const struct libnet_BecomeDC_Partition *partition;
110|        uint32_t ctr_level;
111|        const struct drsuapi_DsGetNCChangesCtr1 *ctr1;
112|        const struct drsuapi_DsGetNCChangesCtr6 *ctr6;
113|        const DATA_BLOB *gensec_skey;
114|};
115|
116|struct libnet_BecomeDC_Callbacks {
117|        void *private_data;
118|        NTSTATUS (*check_options)(void *private_data,
119|                                  const struct libnet_BecomeDC_CheckOptions *info);
120|        NTSTATUS (*prepare_db)(void *private_data,
121|                              const struct libnet_BecomeDC_PrepareDB *info);
122|        NTSTATUS (*schema_chunk)(void *private_data,
123|                               const struct libnet_BecomeDC_StoreChunk *info);
124|        NTSTATUS (*config_chunk)(void *private_data,
125|                               const struct libnet_BecomeDC_StoreChunk *info);
126|        NTSTATUS (*domain_chunk)(void *private_data,
127|                               const struct libnet_BecomeDC_StoreChunk *info);
128|};
129|
130|struct libnet_BecomeDC {
131|        struct {
132|                const char *domain_dns_name;
133|                const char *domain_netbios_name;
134|                const struct dom_sid *domain_sid;
135|                const char *source_dsa_address;
136|                const char *dest_dsa_netbios_name;
137|
138|                struct libnet_BecomeDC_Callbacks callbacks;
139|        } in;
140|
141|        struct {
142|                const char *error_string;
143|        } out;
144|};
145|
146|#endif /* _LIBNET_BECOME_DC_H */
```

A6. source/libnet/libnet_unbecome_dc.h

source/libnet/libnet_unbecome_dc.h:

```
   1|/*
   2|   Unix SMB/CIFS implementation.
   3|
   4|   Copyright (C) Stefan Metzmacher <metze@samba.org> 2006
   5|
   6|   This program is free software; you can redistribute it and/or modify
   7|   it under the terms of the GNU General Public License as published by
   8|   the Free Software Foundation; either version 2 of the License, or
   9|   (at your option) any later version.
  10|
  11|   This program is distributed in the hope that it will be useful,
  12|   but WITHOUT ANY WARRANTY; without even the implied warranty of
  13|   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  14|   GNU General Public License for more details.
  15|
  16|   You should have received a copy of the GNU General Public License
  17|   along with this program; if not, write to the Free Software
  18|   Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
  19|*/
  20|
  21|struct libnet_UnbecomeDC {
  22|        struct {
  23|                const char *domain_dns_name;
  24|                const char *domain_netbios_name;
  25|                const char *source_dsa_address;
  26|                const char *dest_dsa_netbios_name;
  27|        } in;
  28|
  29|        struct {
  30|                const char *error_string;
  31|        } out;
  32|};
```

A7. source/torture/libnet/libnet_BecomeDC.c

source/torture/libnet/libnet_BecomeDC.c:

```
   1|/*
   2|   Unix SMB/CIFS implementation.
   3|
   4|   libnet_BecomeDC() tests
   5|
   6|   Copyright (C) Stefan Metzmacher <metze@samba.org> 2006
   7|
   8|   This program is free software; you can redistribute it and/or modify
   9|   it under the terms of the GNU General Public License as published by
  10|   the Free Software Foundation; either version 2 of the License, or
  11|   (at your option) any later version.
  12|
  13|   This program is distributed in the hope that it will be useful,
  14|   but WITHOUT ANY WARRANTY; without even the implied warranty of
  15|   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  16|   GNU General Public License for more details.
  17|
  18|   You should have received a copy of the GNU General Public License
  19|   along with this program; if not, write to the Free Software
  20|   Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
  21|*/
  22|
  23|#include "includes.h"
  24|#include "lib/cmdline/popt_common.h"
  25|#include "torture/torture.h"
  26|#include "torture/rpc/rpc.h"
  27|#include "libnet/libnet.h"
  28|#include "lib/events/events.h"
  29|#include "dsdb/samdb/samdb.h"
  30|#include "lib/util/dlinklist.h"
  31|#include "lib/ldb/include/ldb.h"
  32|#include "lib/ldb/include/ldb_errors.h"
  33|#include "librpc/ndr/libndr.h"
  34|#include "librpc/gen_ndr/ndr_drsuapi.h"
  35|#include "librpc/gen_ndr/ndr_drsblobs.h"
  36|#include "librpc/gen_ndr/ndr_misc.h"
  37|#include "system/time.h"
  38|#include "auth/auth.h"
  39|#include "lib/db_wrap.h"
  40|#include "lib/appweb/ejs/ejs.h"
  41|#include "lib/appweb/ejs/ejsInternal.h"
  42|#include "scripting/ejs/smbcalls.h"
  43|
  44|static EjsId eid;
  45|static int ejs_error;
  46|
  47|static void test_ejs_exception(const char *reason)
  48|{
  49|        Ejs *ep = ejsPtr(eid);
  50|        ejsSetErrorMsg(eid, "%s", reason);
  51|        fprintf(stderr, "%s", ep->error);
  52|        ejs_error = 127;
  53|}
  54|
  55|static int test_run_ejs(char *script)
  56|{
  57|        EjsHandle handle = 0;
  58|        MprVar result;
  59|        char *emsg;
  60|        TALLOC_CTX *mem_ctx = talloc_new(NULL);
  61|        struct MprVar *return_var;
  62|
  63|        mprSetCtx(mem_ctx);
  64|
  65|        if (ejsOpen(NULL, NULL, NULL) != 0) {
  66|                d_printf("ejsOpen(): unable to initialise EJS subsystem\n");
  67|                ejs_error = 127;
  68|                goto failed;
  69|        }
  70|
  71|        smb_setup_ejs_functions(test_ejs_exception);
  72|
  73|        if ((eid = ejsOpenEngine(handle, 0)) == (EjsId)-1) {
  74|                d_printf("smbscript: ejsOpenEngine(): unable to initialise an EJS
->    engine\n");
  75|                ejs_error = 127;
  76|                goto failed;
  77|        }
  78|
```

```
  79|        mprSetVar(ejsGetGlobalObject(eid), "ARGV", mprList("ARGV", NULL));
  80|
  81|        /* run the script */
  82|        if (ejsEvalScript(eid, script, &result, &emsg) == -1) {
  83|                d_printf("smbscript: ejsEvalScript(): %s\n", emsg);
  84|                if (ejs_error == 0) ejs_error = 127;
  85|                goto failed;
  86|        }
  87|
  88|        return_var = ejsGetReturnValue(eid);
  89|        ejs_error = mprVarToNumber(return_var);
  90|
  91|failed:
  92|        ejsClose();
  93|        talloc_free(mem_ctx);
  94|        return ejs_error;
  95|}
  96|
  97|struct test_become_dc_state {
  98|        struct libnet_context *ctx;
  99|        const char *netbios_name;
 100|        struct test_join *tj;
 101|        struct cli_credentials *machine_account;
 102|        struct dsdb_schema *self_made_schema;
 103|        const struct dsdb_schema *schema;
 104|
 105|        struct ldb_context *ldb;
 106|
 107|        struct {
 108|                uint32_t object_count;
 109|                struct drsuapi_DsReplicaObjectListItemEx *first_object;
 110|                struct drsuapi_DsReplicaObjectListItemEx *last_object;
 111|        } schema_part;
 112|
 113|        struct {
 114|                const char *samdb_ldb;
 115|                const char *domaindn_ldb;
 116|                const char *configdn_ldb;
 117|                const char *schemadn_ldb;
 118|                const char *secrets_ldb;
 119|                const char *secrets_keytab;
 120|        } path;
 121|};
 122|
 123|static NTSTATUS test_become_dc_check_options(void *private_data,
 124|                                             const struct libnet_BecomeDC_CheckOptions
->    *o)
 125|{
 126|        struct test_become_dc_state *s = talloc_get_type(private_data, struct
->    test_become_dc_state);
 127|
 128|        DEBUG(0,("Become DC [%s] of Domain[%s]/[%s]\n",
 129|                s->netbios_name,
 130|                o->domain->netbios_name, o->domain->dns_name));
 131|
 132|        DEBUG(0,("Promotion Partner is Server[%s] from Site[%s]\n",
 133|                o->source_dsa->dns_name, o->source_dsa->site_name));
 134|
 135|        DEBUG(0,("Options:crossRef behavior_version[%u]\n"
 136|                        "\tschema object_version[%u]\n"
 137|                        "\tdomain behavior_version[%u]\n"
 138|                        "\tdomain w2k3_update_revision[%u]\n",
 139|                o->forest->crossref_behavior_version,
 140|                o->forest->schema_object_version,
 141|                o->domain->behavior_version,
 142|                o->domain->w2k3_update_revision));
 143|
 144|        return NT_STATUS_OK;
 145|}
 146|
 147|static NTSTATUS test_become_dc_prepare_db(void *private_data,
 148|                                          const struct libnet_BecomeDC_PrepareDB *p)
 149|{
 150|        struct test_become_dc_state *s = talloc_get_type(private_data, struct
->    test_become_dc_state);
 151|        char *ejs;
 152|        int ret;
 153|        bool ok;
 154|
 155|        DEBUG(0,("New Server[%s] in Site[%s]\n",
 156|                p->dest_dsa->dns_name, p->dest_dsa->site_name));
 157|
 158|        DEBUG(0,("DSA Instance [%s]\n"
 159|                "\tobjectGUID[%s]\n"
 160|                "\tinvocationId[%s]\n",
 161|                p->dest_dsa->ntds_dn_str,
 162|                GUID_string(s, &p->dest_dsa->ntds_guid),
```

```
163|                GUID_string(s, &p->dest_dsa->invocation_id)));
164|
165|        DEBUG(0,("Pathes under PRIVATEDIR[%s]\n"
166|                 "SAMDB[%s] SECRETS[%s] KEYTAB[%s]\n",
167|                lp_private_dir(),
168|                s->path.samdb_ldb,
169|                s->path.secrets_ldb,
170|                s->path.secrets_keytab));
171|
172|        DEBUG(0,("Schema Partition[%s => %s]\n",
173|                p->forest->schema_dn_str, s->path.schemadn_ldb));
174|
175|        DEBUG(0,("Config Partition[%s => %s]\n",
176|                p->forest->config_dn_str, s->path.configdn_ldb));
177|
178|        DEBUG(0,("Domain Partition[%s => %s]\n",
179|                p->domain->dn_str, s->path.domaindn_ldb));
180|
181|        ejs = talloc_asprintf(s,
182|                "libinclude(\"base.js\");\n"
183|                "libinclude(\"provision.js\");\n"
184|                "\n"
185|                "function message() { print(vsprintf(arguments)); }\n"
186|                "\n"
187|                "var subobj = provision_guess();\n"
188|                "subobj.ROOTDN       = \"%s\";\n"
189|                "subobj.DOMAINDN     = \"%s\";\n"
190|                "subobj.DOMAINDN_LDB = \"%s\";\n"
191|                "subobj.CONFIGDN     = \"%s\";\n"
192|                "subobj.CONFIGDN_LDB = \"%s\";\n"
193|                "subobj.SCHEMADN     = \"%s\";\n"
194|                "subobj.SCHEMADN_LDB = \"%s\";\n"
195|                "subobj.HOSTNAME     = \"%s\";\n"
196|                "subobj.DNSNAME      = \"%s\";\n"
197|                "subobj.DEFAULTSITE  = \"%s\";\n"
198|                "\n"
199|                "modules_list        = new Array(\"rootdse\",\n"
200|                "                                \"kludge_acl\",\n"
201|                "                                \"paged_results\",\n"
202|                "                                \"server_sort\",\n"
203|                "                                \"extended_dn\",\n"
204|                "                                \"asq\",\n"
205|                "                                \"samldb\",\n"
206|                "                                \"operational\",\n"
207|                "                                \"objectclass\",\n"
208|                "                                \"rdn_name\",\n"
209|                "                                \"show_deleted\",\n"
210|                "                                \"partition\");\n"
211|                "subobj.MODULES_LIST = join(\",\", modules_list);\n"
212|                "subobj.DOMAINDN_MOD = \"pdc_fsmo,password_hash,repl_meta_data\";\n"
213|                "subobj.CONFIGDN_MOD = \"naming_fsmo,repl_meta_data\";\n"
214|                "subobj.SCHEMADN_MOD = \"schema_fsmo,repl_meta_data\";\n"
215|                "\n"
216|                "subobj.KRBTGTPASS   = \"_NOT_USED_\";\n"
217|                "subobj.MACHINEPASS  = \"%s\";\n"
218|                "subobj.ADMINPASS    = \"_NOT_USED_\";\n"
219|                "\n"
220|                "var paths = provision_default_paths(subobj);\n"
221|                "paths.samdb = \"%s\";\n"
222|                "paths.secrets = \"%s\";\n"
223|                "paths.keytab = \"%s\";\n"
224|                "\n"
225|                "var system_session = system_session();\n"
226|                "\n"
227|                "var ok = provision_become_dc(subobj, message, paths,
->    system_session);\n"
228|                "assert(ok);\n"
229|                "\n"
230|                "return 0;\n",
231|                p->forest->root_dn_str,         /* subobj.ROOTDN */
232|                p->domain->dn_str,              /* subobj.DOMAINDN */
233|                s->path.domaindn_ldb,           /* subobj.DOMAINDN_LDB */
234|                p->forest->config_dn_str,       /* subobj.CONFIGDN */
235|                s->path.configdn_ldb,           /* subobj.CONFIGDN_LDB */
236|                p->forest->schema_dn_str,       /* subobj.SCHEMADN */
237|                s->path.schemadn_ldb,           /* subobj.SCHEMADN_LDB */
238|                p->dest_dsa->netbios_name,      /* subobj.HOSTNAME */
239|                p->dest_dsa->dns_name,          /* subobj.DNSNAME */
240|                p->dest_dsa->site_name,         /* subobj.DEFAULTSITE */
241|                cli_credentials_get_password(s->machine_account),/* subobj.MACHINEPASS
->    */
242|                s->path.samdb_ldb,              /* paths.samdb */
243|                s->path.secrets_ldb,            /* paths.secrets */
244|                s->path.secrets_keytab);        /* paths.keytab */
245|        NT_STATUS_HAVE_NO_MEMORY(ejs);
246|
247|        ret = test_run_ejs(ejs);
```

```
248|        if (ret != 0) {
249|                DEBUG(0,("Failed to run ejs script: %d:\n%s",
250|                         ret, ejs));
251|                talloc_free(ejs);
252|                return NT_STATUS_FOOBAR;
253|        }
254|        talloc_free(ejs);
255|
256|        talloc_free(s->ldb);
257|
258|        DEBUG(0,("Open the SAM LDB with system credentials: %s\n", s->path.samdb_ldb));
259|
260|        s->ldb = ldb_wrap_connect(s, s->path.samdb_ldb,
261|                                   system_session(s),
262|                                   NULL, 0, NULL);
263|        if (!s->ldb) {
264|                DEBUG(0,("Failed to open '%s'\n",
265|                         s->path.samdb_ldb));
266|                return NT_STATUS_INTERNAL_DB_ERROR;
267|        }
268|
269|        ok = samdb_set_ntds_invocation_id(s->ldb, &p->dest_dsa->invocation_id);
270|        if (!ok) {
271|                DEBUG(0,("Failed to set cached ntds invocationId\n"));
272|                return NT_STATUS_FOOBAR;
273|        }
274|        ok = samdb_set_ntds_objectGUID(s->ldb, &p->dest_dsa->ntds_guid);
275|        if (!ok) {
276|                DEBUG(0,("Failed to set cached ntds objectGUID\n"));
277|                return NT_STATUS_FOOBAR;
278|        }
279|
280|        return NT_STATUS_OK;
281|}
282|
283|static NTSTATUS test_apply_schema(struct test_become_dc_state *s,
284|                                   const struct libnet_BecomeDC_StoreChunk *c)
285|{
286|        WERROR status;
287|        const struct drsuapi_DsReplicaOIDMapping_Ctr *mapping_ctr;
288|        uint32_t total_object_count;
289|        uint32_t object_count;
290|        struct drsuapi_DsReplicaObjectListItemEx *first_object;
291|        struct drsuapi_DsReplicaObjectListItemEx *cur;
292|        uint32_t linked_attributes_count;
293|        struct drsuapi_DsReplicaLinkedAttribute *linked_attributes;
294|        const struct drsuapi_DsReplicaCursor2CtrEx *uptodateness_vector;
295|        struct dsdb_extended_replicated_objects *objs;
296|        struct repsFromTo1 *s_dsa;
297|        char *tmp_dns_name;
298|        struct ldb_message *msg;
299|        struct ldb_val prefixMap_val;
300|        struct ldb_message_element *prefixMap_el;
301|        struct ldb_val schemaInfo_val;
302|        uint32_t i;
303|        int ret;
304|        bool ok;
305|
306|        DEBUG(0,("Analyze and apply schema objects\n"));
307|
308|        s_dsa                   = talloc_zero(s, struct repsFromTo1);
309|        NT_STATUS_HAVE_NO_MEMORY(s_dsa);
310|        s_dsa->other_info       = talloc(s_dsa, struct repsFromTo1OtherInfo);
311|        NT_STATUS_HAVE_NO_MEMORY(s_dsa->other_info);
312|
313|        switch (c->ctr_level) {
314|        case 1:
315|                mapping_ctr                     = &c->ctr1->mapping_ctr;
316|                total_object_count              = c->ctr1->total_object_count;
317|                object_count                    = s->schema_part.object_count;
318|                first_object                    = s->schema_part.first_object;
319|                linked_attributes_count         = 0;
320|                linked_attributes               = NULL;
321|                s_dsa->highwatermark            = c->ctr1->new_highwatermark;
322|                s_dsa->source_dsa_obj_guid      = c->ctr1->source_dsa_guid;
323|                s_dsa->source_dsa_invocation_id = c->ctr1->source_dsa_invocation_id;
324|                uptodateness_vector             = NULL; /* TODO: map it */
325|                break;
326|        case 6:
327|                mapping_ctr                     = &c->ctr6->mapping_ctr;
328|                total_object_count              = c->ctr6->total_object_count;
329|                object_count                    = s->schema_part.object_count;
330|                first_object                    = s->schema_part.first_object;
331|                linked_attributes_count         = 0; /* TODO: ! */
332|                linked_attributes               = NULL; /* TODO: ! */;
333|                s_dsa->highwatermark            = c->ctr6->new_highwatermark;
334|                s_dsa->source_dsa_obj_guid      = c->ctr6->source_dsa_guid;
```

```
 335|                s_dsa->source_dsa_invocation_id = c->ctr6->source_dsa_invocation_id;
 336|                uptodateness_vector            = c->ctr6->uptodateness_vector;
 337|                break;
 338|        default:
 339|                return NT_STATUS_INVALID_PARAMETER;
 340|        }
 341|
 342|        s_dsa->replica_flags           = DRSUAPI_DS_REPLICA_NEIGHBOUR_WRITEABLE
 343|                                        | DRSUAPI_DS_REPLICA_NEIGHBOUR_SYNC_ON_STARTUP
 344|                                        | DRSUAPI_DS_REPLICA_NEIGHBOUR_DO_SCHEDULED_SYNCS
->   ;
 345|        memset(s_dsa->schedule, 0x11, sizeof(s_dsa->schedule));
 346|
 347|        tmp_dns_name    = GUID_string(s_dsa->other_info, &s_dsa->source_dsa_obj_guid);
 348|        NT_STATUS_HAVE_NO_MEMORY(tmp_dns_name);
 349|        tmp_dns_name    = talloc_asprintf_append(tmp_dns_name, "._msdcs.%s",
->    c->forest->dns_name);
 350|        NT_STATUS_HAVE_NO_MEMORY(tmp_dns_name);
 351|        s_dsa->other_info->dns_name = tmp_dns_name;
 352|
 353|        for (cur = first_object; cur; cur = cur->next_object) {
 354|                bool is_attr = false;
 355|                bool is_class = false;
 356|
 357|                for (i=0; i < cur->object.attribute_ctr.num_attributes; i++) {
 358|                        struct drsuapi_DsReplicaAttribute *a;
 359|                        uint32_t j;
 360|                        const char *oid = NULL;
 361|
 362|                        a = &cur->object.attribute_ctr.attributes[i];
 363|                        status = dsdb_map_int2oid(s->self_made_schema, a->attid, s,
->    &oid);
 364|                        if (!W_ERROR_IS_OK(status)) {
 365|                                return werror_to_ntstatus(status);
 366|                        }
 367|
 368|                        switch (a->attid) {
 369|                        case DRSUAPI_ATTRIBUTE_objectClass:
 370|                                for (j=0; j < a->value_ctr.num_values; j++) {
 371|                                        uint32_t val = 0xFFFFFFFF;
 372|
 373|                                        if (a->value_ctr.values[i].blob
 374|                                            && a->value_ctr.values[i].blob->length == 4)
->    {
 375|                                                val = IVAL(a->value_ctr.values[i].blob->d
->   ata,0);
 376|                                        }
 377|
 378|                                        if (val == DRSUAPI_OBJECTCLASS_attributeSchema)
->    {
 379|                                                is_attr = true;
 380|                                        }
 381|                                        if (val == DRSUAPI_OBJECTCLASS_classSchema) {
 382|                                                is_class = true;
 383|                                        }
 384|                                }
 385|
 386|                                break;
 387|                        default:
 388|                                break;
 389|                        }
 390|                }
 391|
 392|                if (is_attr) {
 393|                        struct dsdb_attribute *sa;
 394|
 395|                        sa = talloc_zero(s->self_made_schema, struct dsdb_attribute);
 396|                        NT_STATUS_HAVE_NO_MEMORY(sa);
 397|
 398|                        status = dsdb_attribute_from_drsuapi(s->self_made_schema,
->    &cur->object, s, sa);
 399|                        if (!W_ERROR_IS_OK(status)) {
 400|                                return werror_to_ntstatus(status);
 401|                        }
 402|
 403|                        DLIST_ADD_END(s->self_made_schema->attributes, sa, struct
->    dsdb_attribute *);
 404|                }
 405|
 406|                if (is_class) {
 407|                        struct dsdb_class *sc;
 408|
 409|                        sc = talloc_zero(s->self_made_schema, struct dsdb_class);
 410|                        NT_STATUS_HAVE_NO_MEMORY(sc);
 411|
 412|                        status = dsdb_class_from_drsuapi(s->self_made_schema,
->    &cur->object, s, sc);
```

```
413|                       if (!W_ERROR_IS_OK(status)) {
414|                               return werror_to_ntstatus(status);
415|                       }
416|
417|                       DLIST_ADD_END(s->self_made_schema->classes, sc, struct dsdb_class
->    *);
418|               }
419|       }
420|
421|       /* attach the schema to the ldb */
422|       ret = dsdb_set_schema(s->ldb, s->self_made_schema);
423|       if (ret != LDB_SUCCESS) {
424|               return NT_STATUS_FOOBAR;
425|       }
426|       /* we don't want to access the self made schema anymore */
427|       s->self_made_schema = NULL;
428|       s->schema = dsdb_get_schema(s->ldb);
429|
430|       status = dsdb_extended_replicated_objects_commit(s->ldb,
431|                                                       c->partition->nc.dn,
432|                                                       mapping_ctr,
433|                                                       object_count,
434|                                                       first_object,
435|                                                       linked_attributes_count,
436|                                                       linked_attributes,
437|                                                       s_dsa,
438|                                                       uptodateness_vector,
439|                                                       c->gensec_skey,
440|                                                       s, &objs);
441|       if (!W_ERROR_IS_OK(status)) {
442|               DEBUG(0,("Failed to commit objects: %s\n", win_errstr(status)));
443|               return werror_to_ntstatus(status);
444|       }
445|
446|       if (lp_parm_bool(-1, "become dc", "dump objects", False)) {
447|               for (i=0; i < objs->num_objects; i++) {
448|                       struct ldb_ldif ldif;
449|                       fprintf(stdout, "#\n");
450|                       ldif.changetype = LDB_CHANGETYPE_NONE;
451|                       ldif.msg = objs->objects[i].msg;
452|                       ldb_ldif_write_file(s->ldb, stdout, &ldif);
453|                       NDR_PRINT_DEBUG(replPropertyMetaDataBlob,
->    objs->objects[i].meta_data);
454|               }
455|       }
456|
457|       msg = ldb_msg_new(objs);
458|       NT_STATUS_HAVE_NO_MEMORY(msg);
459|       msg->dn = objs->partition_dn;
460|
461|       status = dsdb_get_oid_mappings_ldb(s->schema, msg, &prefixMap_val,
->    &schemaInfo_val);
462|       if (!W_ERROR_IS_OK(status)) {
463|               DEBUG(0,("Failed dsdb_get_oid_mappings_ldb(%s)\n", win_errstr(status)));
464|               return werror_to_ntstatus(status);
465|       }
466|
467|       /* we only add prefixMap here, because schemaInfo is a replicated attribute and
->    already applied */
468|       ret = ldb_msg_add_value(msg, "prefixMap", &prefixMap_val, &prefixMap_el);
469|       if (ret != LDB_SUCCESS) {
470|               return NT_STATUS_FOOBAR;
471|       }
472|       prefixMap_el->flags = LDB_FLAG_MOD_REPLACE;
473|
474|       ret = ldb_modify(s->ldb, msg);
475|       if (ret != LDB_SUCCESS) {
476|               DEBUG(0,("Failed to add prefixMap and schemaInfo %s\n",
->    ldb_strerror(ret)));
477|               return NT_STATUS_FOOBAR;
478|       }
479|
480|       talloc_free(s_dsa);
481|       talloc_free(objs);
482|
483|       /* reopen the ldb */
484|       talloc_free(s->ldb); /* this also free's the s->schema, because dsdb_set_schema()
->    steals it */
485|       s->schema = NULL;
486|
487|       DEBUG(0,("Reopen the SAM LDB with system credentials and a already stored schema:
->    %s\n", s->path.samdb_ldb));
488|       s->ldb = ldb_wrap_connect(s, s->path.samdb_ldb,
489|                                  system_session(s),
490|                                  NULL, 0, NULL);
491|       if (!s->ldb) {
492|               DEBUG(0,("Failed to open '%s'\n",
```

```
493|                       s->path.samdb_ldb));
494|                return NT_STATUS_INTERNAL_DB_ERROR;
495|        }
496|
497|        ok = samdb_set_ntds_invocation_id(s->ldb, &c->dest_dsa->invocation_id);
498|        if (!ok) {
499|                DEBUG(0,("Failed to set cached ntds invocationId\n"));
500|                return NT_STATUS_FOOBAR;
501|        }
502|        ok = samdb_set_ntds_objectGUID(s->ldb, &c->dest_dsa->ntds_guid);
503|        if (!ok) {
504|                DEBUG(0,("Failed to set cached ntds objectGUID\n"));
505|                return NT_STATUS_FOOBAR;
506|        }
507|
508|        s->schema = dsdb_get_schema(s->ldb);
509|        if (!s->schema) {
510|                DEBUG(0,("Failed to get loaded dsdb_schema\n"));
511|                return NT_STATUS_FOOBAR;
512|        }
513|
514|        return NT_STATUS_OK;
515|}
516|
517|static NTSTATUS test_become_dc_schema_chunk(void *private_data,
518|                                            const struct libnet_BecomeDC_StoreChunk *c)
519|{
520|        struct test_become_dc_state *s = talloc_get_type(private_data, struct
->    test_become_dc_state);
521|        WERROR status;
522|        const struct drsuapi_DsReplicaOIDMapping_Ctr *mapping_ctr;
523|        uint32_t total_object_count;
524|        uint32_t object_count;
525|        struct drsuapi_DsReplicaObjectListItemEx *first_object;
526|        struct drsuapi_DsReplicaObjectListItemEx *cur;
527|
528|        switch (c->ctr_level) {
529|        case 1:
530|                mapping_ctr             = &c->ctr1->mapping_ctr;
531|                total_object_count      = c->ctr1->total_object_count;
532|                object_count            = c->ctr1->object_count;
533|                first_object            = c->ctr1->first_object;
534|                break;
535|        case 6:
536|                mapping_ctr             = &c->ctr6->mapping_ctr;
537|                total_object_count      = c->ctr6->total_object_count;
538|                object_count            = c->ctr6->object_count;
539|                first_object            = c->ctr6->first_object;
540|                break;
541|        default:
542|                return NT_STATUS_INVALID_PARAMETER;
543|        }
544|
545|        if (total_object_count) {
546|                DEBUG(0,("Schema-DN[%s] objects[%u/%u]\n",
547|                         c->partition->nc.dn, object_count, total_object_count));
548|        } else {
549|                DEBUG(0,("Schema-DN[%s] objects[%u]\n",
550|                c->partition->nc.dn, object_count));
551|        }
552|
553|        if (!s->schema) {
554|                s->self_made_schema = talloc_zero(s, struct dsdb_schema);
555|                NT_STATUS_HAVE_NO_MEMORY(s->self_made_schema);
556|
557|                status = dsdb_load_oid_mappings_drsuapi(s->self_made_schema,
->    mapping_ctr);
558|                if (!W_ERROR_IS_OK(status)) {
559|                        return werror_to_ntstatus(status);
560|                }
561|
562|                s->schema = s->self_made_schema;
563|        } else {
564|                status = dsdb_verify_oid_mappings_drsuapi(s->schema, mapping_ctr);
565|                if (!W_ERROR_IS_OK(status)) {
566|                        return werror_to_ntstatus(status);
567|                }
568|        }
569|
570|        if (!s->schema_part.first_object) {
571|                s->schema_part.object_count = object_count;
572|                s->schema_part.first_object = talloc_steal(s, first_object);
573|        } else {
574|                s->schema_part.object_count             += object_count;
575|                s->schema_part.last_object->next_object = talloc_steal(s->schema_part.las
->   t_object,
576|                                                                       first_object);
```

```
 577|        }
 578|        for (cur = first_object; cur->next_object; cur = cur->next_object) {}
 579|        s->schema_part.last_object = cur;
 580|
 581|        if (c->partition->highwatermark.tmp_highest_usn ==
->    c->partition->highwatermark.highest_usn) {
 582|                return test_apply_schema(s, c);
 583|        }
 584|
 585|        return NT_STATUS_OK;
 586|}
 587|
 588|static NTSTATUS test_become_dc_store_chunk(void *private_data,
 589|                                           const struct libnet_BecomeDC_StoreChunk *c)
 590|{
 591|        struct test_become_dc_state *s = talloc_get_type(private_data, struct
->    test_become_dc_state);
 592|        WERROR status;
 593|        const struct drsuapi_DsReplicaOIDMapping_Ctr *mapping_ctr;
 594|        uint32_t total_object_count;
 595|        uint32_t object_count;
 596|        struct drsuapi_DsReplicaObjectListItemEx *first_object;
 597|        uint32_t linked_attributes_count;
 598|        struct drsuapi_DsReplicaLinkedAttribute *linked_attributes;
 599|        const struct drsuapi_DsReplicaCursor2CtrEx *uptodateness_vector;
 600|        struct dsdb_extended_replicated_objects *objs;
 601|        struct repsFromTo1 *s_dsa;
 602|        char *tmp_dns_name;
 603|        uint32_t i;
 604|
 605|        s_dsa                   = talloc_zero(s, struct repsFromTo1);
 606|        NT_STATUS_HAVE_NO_MEMORY(s_dsa);
 607|        s_dsa->other_info       = talloc(s_dsa, struct repsFromTo1OtherInfo);
 608|        NT_STATUS_HAVE_NO_MEMORY(s_dsa->other_info);
 609|
 610|        switch (c->ctr_level) {
 611|        case 1:
 612|                mapping_ctr                     = &c->ctr1->mapping_ctr;
 613|                total_object_count              = c->ctr1->total_object_count;
 614|                object_count                    = c->ctr1->object_count;
 615|                first_object                    = c->ctr1->first_object;
 616|                linked_attributes_count         = 0;
 617|                linked_attributes               = NULL;
 618|                s_dsa->highwatermark            = c->ctr1->new_highwatermark;
 619|                s_dsa->source_dsa_obj_guid      = c->ctr1->source_dsa_guid;
 620|                s_dsa->source_dsa_invocation_id = c->ctr1->source_dsa_invocation_id;
 621|                uptodateness_vector             = NULL; /* TODO: map it */
 622|                break;
 623|        case 6:
 624|                mapping_ctr                     = &c->ctr6->mapping_ctr;
 625|                total_object_count              = c->ctr6->total_object_count;
 626|                object_count                    = c->ctr6->object_count;
 627|                first_object                    = c->ctr6->first_object;
 628|                linked_attributes_count         = c->ctr6->linked_attributes_count;
 629|                linked_attributes               = c->ctr6->linked_attributes;
 630|                s_dsa->highwatermark            = c->ctr6->new_highwatermark;
 631|                s_dsa->source_dsa_obj_guid      = c->ctr6->source_dsa_guid;
 632|                s_dsa->source_dsa_invocation_id = c->ctr6->source_dsa_invocation_id;
 633|                uptodateness_vector             = c->ctr6->uptodateness_vector;
 634|                break;
 635|        default:
 636|                return NT_STATUS_INVALID_PARAMETER;
 637|        }
 638|
 639|        s_dsa->replica_flags           = DRSUAPI_DS_REPLICA_NEIGHBOUR_WRITEABLE
 640|                                        | DRSUAPI_DS_REPLICA_NEIGHBOUR_SYNC_ON_STARTUP
 641|                                        | DRSUAPI_DS_REPLICA_NEIGHBOUR_DO_SCHEDULED_SYNCS
->   ;
 642|        memset(s_dsa->schedule, 0x11, sizeof(s_dsa->schedule));
 643|
 644|        tmp_dns_name    = GUID_string(s_dsa->other_info, &s_dsa->source_dsa_obj_guid);
 645|        NT_STATUS_HAVE_NO_MEMORY(tmp_dns_name);
 646|        tmp_dns_name    = talloc_asprintf_append(tmp_dns_name, "._msdcs.%s",
->    c->forest->dns_name);
 647|        NT_STATUS_HAVE_NO_MEMORY(tmp_dns_name);
 648|        s_dsa->other_info->dns_name = tmp_dns_name;
 649|
 650|        if (total_object_count) {
 651|                DEBUG(0,("Partition[%s] objects[%u/%u]\n",
 652|                         c->partition->nc.dn, object_count, total_object_count));
 653|        } else {
 654|                DEBUG(0,("Partition[%s] objects[%u]\n",
 655|                         c->partition->nc.dn, object_count));
 656|        }
 657|
 658|        status = dsdb_extended_replicated_objects_commit(s->ldb,
 659|                                                           c->partition->nc.dn,
```

```
660|                                                        mapping_ctr,
661|                                                        object_count,
662|                                                        first_object,
663|                                                        linked_attributes_count,
664|                                                        linked_attributes,
665|                                                        s_dsa,
666|                                                        uptodateness_vector,
667|                                                        c->gensec_skey,
668|                                                        s, &objs);
669|        if (!W_ERROR_IS_OK(status)) {
670|                DEBUG(0,("Failed to commit objects: %s\n", win_errstr(status)));
671|                return werror_to_ntstatus(status);
672|        }
673|
674|        if (lp_parm_bool(-1, "become dc", "dump objects", False)) {
675|                for (i=0; i < objs->num_objects; i++) {
676|                        struct ldb_ldif ldif;
677|                        fprintf(stdout, "#\n");
678|                        ldif.changetype = LDB_CHANGETYPE_NONE;
679|                        ldif.msg = objs->objects[i].msg;
680|                        ldb_ldif_write_file(s->ldb, stdout, &ldif);
681|                        NDR_PRINT_DEBUG(replPropertyMetaDataBlob,
->    objs->objects[i].meta_data);
682|                }
683|        }
684|        talloc_free(s_dsa);
685|        talloc_free(objs);
686|
687|        for (i=0; i < linked_attributes_count; i++) {
688|                const struct dsdb_attribute *sa;
689|
690|                if (!linked_attributes[i].identifier) {
691|                        return NT_STATUS_FOOBAR;
692|                }
693|
694|                if (!linked_attributes[i].value.blob) {
695|                        return NT_STATUS_FOOBAR;
696|                }
697|
698|                sa = dsdb_attribute_by_attributeID_id(s->schema,
699|                                                      linked_attributes[i].attid);
700|                if (!sa) {
701|                        return NT_STATUS_FOOBAR;
702|                }
703|
704|                if (lp_parm_bool(-1, "become dc", "dump objects", False)) {
705|                        DEBUG(0,("# %s\n", sa->lDAPDisplayName));
706|                        NDR_PRINT_DEBUG(drsuapi_DsReplicaLinkedAttribute,
->    &linked_attributes[i]);
707|                        dump_data(0,
708|                                linked_attributes[i].value.blob->data,
709|                                linked_attributes[i].value.blob->length);
710|                }
711|        }
712|
713|        return NT_STATUS_OK;
714|}
715|
716|BOOL torture_net_become_dc(struct torture_context *torture)
717|{
718|        BOOL ret = True;
719|        NTSTATUS status;
720|        struct libnet_BecomeDC b;
721|        struct libnet_UnbecomeDC u;
722|        struct test_become_dc_state *s;
723|        struct ldb_message *msg;
724|        int ldb_ret;
725|        uint32_t i;
726|
727|        s = talloc_zero(torture, struct test_become_dc_state);
728|        if (!s) return False;
729|
730|        s->netbios_name = lp_parm_string(-1, "become dc", "smbtorture dc");
731|        if (!s->netbios_name || !s->netbios_name[0]) {
732|                s->netbios_name = "smbtorturedc";
733|        }
734|
735|        s->path.samdb_ldb       = talloc_asprintf(s, "%s_samdb.ldb", s->netbios_name);
736|        if (!s->path.samdb_ldb) return false;
737|        s->path.domaindn_ldb    = talloc_asprintf(s, "%s_domain.ldb", s->netbios_name);
738|        if (!s->path.domaindn_ldb) return false;
739|        s->path.configdn_ldb    = talloc_asprintf(s, "%s_config.ldb", s->netbios_name);
740|        if (!s->path.configdn_ldb) return false;
741|        s->path.schemadn_ldb    = talloc_asprintf(s, "%s_schema.ldb", s->netbios_name);
742|        if (!s->path.schemadn_ldb) return false;
743|        s->path.secrets_ldb     = talloc_asprintf(s, "%s_secrets.ldb", s->netbios_name);
744|        if (!s->path.secrets_ldb) return false;
```

```
 745|        s->path.secrets_keytab  = talloc_asprintf(s, "%s_secrets.keytab",
->    s->netbios_name);
 746|        if (!s->path.secrets_keytab) return false;
 747|
 748|        /* Join domain as a member server. */
 749|        s->tj = torture_join_domain(s->netbios_name,
 750|                                   ACB_WSTRUST,
 751|                                   &s->machine_account);
 752|        if (!s->tj) {
 753|                DEBUG(0, ("%s failed to join domain as workstation\n",
 754|                          s->netbios_name));
 755|                return False;
 756|        }
 757|
 758|        s->ctx = libnet_context_init(event_context_init(s));
 759|        s->ctx->cred = cmdline_credentials;
 760|
 761|        s->ldb = ldb_init(s);
 762|
 763|        ZERO_STRUCT(b);
 764|        b.in.domain_dns_name            = torture_join_dom_dns_name(s->tj);
 765|        b.in.domain_netbios_name        = torture_join_dom_netbios_name(s->tj);
 766|        b.in.domain_sid                 = torture_join_sid(s->tj);
 767|        b.in.source_dsa_address         = lp_parm_string(-1, "torture", "host");
 768|        b.in.dest_dsa_netbios_name      = s->netbios_name;
 769|
 770|        b.in.callbacks.private_data     = s;
 771|        b.in.callbacks.check_options    = test_become_dc_check_options;
 772|        b.in.callbacks.prepare_db       = test_become_dc_prepare_db;
 773|        b.in.callbacks.schema_chunk     = test_become_dc_schema_chunk;
 774|        b.in.callbacks.config_chunk     = test_become_dc_store_chunk;
 775|        b.in.callbacks.domain_chunk     = test_become_dc_store_chunk;
 776|
 777|        status = libnet_BecomeDC(s->ctx, s, &b);
 778|        if (!NT_STATUS_IS_OK(status)) {
 779|                printf("libnet_BecomeDC() failed - %s\n", nt_errstr(status));
 780|                ret = False;
 781|                goto cleanup;
 782|        }
 783|
 784|        msg = ldb_msg_new(s);
 785|        if (!msg) {
 786|                printf("ldb_msg_new() failed\n");
 787|                ret = False;
 788|                goto cleanup;
 789|        }
 790|        msg->dn = ldb_dn_new(msg, s->ldb, "cn=ROOTDSE");
 791|        if (!msg->dn) {
 792|                printf("ldb_msg_new(cn=ROOTDSE) failed\n");
 793|                ret = False;
 794|                goto cleanup;
 795|        }
 796|
 797|        ldb_ret = ldb_msg_add_string(msg, "isSynchronized", "TRUE");
 798|        if (ldb_ret != LDB_SUCCESS) {
 799|                printf("ldb_msg_add_string(msg, isSynchronized, TRUE) failed: %d\n",
->    ldb_ret);
 800|                ret = False;
 801|                goto cleanup;
 802|        }
 803|
 804|        for (i=0; i < msg->num_elements; i++) {
 805|                msg->elements[i].flags = LDB_FLAG_MOD_REPLACE;
 806|        }
 807|
 808|        printf("mark ROOTDSE with isSynchronized=TRUE\n");
 809|        ldb_ret = ldb_modify(s->ldb, msg);
 810|        if (ldb_ret != LDB_SUCCESS) {
 811|                printf("ldb_modify() failed: %d\n", ldb_ret);
 812|                ret = False;
 813|                goto cleanup;
 814|        }
 815|
 816|        /* reopen the ldb */
 817|        talloc_free(s->ldb); /* this also free's the s->schema, because dsdb_set_schema()
->    steals it */
 818|        s->schema = NULL;
 819|
 820|        DEBUG(0,("Reopen the SAM LDB with system credentials and all replicated data:
->    %s\n", s->path.samdb_ldb));
 821|        s->ldb = ldb_wrap_connect(s, s->path.samdb_ldb,
 822|                                  system_session(s),
 823|                                  NULL, 0, NULL);
 824|        if (!s->ldb) {
 825|                DEBUG(0,("Failed to open '%s'\n",
 826|                         s->path.samdb_ldb));
 827|                ret = False;
```

```
828|                goto cleanup;
829|        }
830|
831|        s->schema = dsdb_get_schema(s->ldb);
832|        if (!s->schema) {
833|                DEBUG(0,("Failed to get loaded dsdb_schema\n"));
834|                ret = False;
835|                goto cleanup;
836|        }
837|
838|        if (lp_parm_bool(-1, "become dc", "do not unjoin", false)) {
839|                talloc_free(s);
840|                return ret;
841|        }
842|
843|cleanup:
844|        ZERO_STRUCT(u);
845|        u.in.domain_dns_name            = torture_join_dom_dns_name(s->tj);
846|        u.in.domain_netbios_name        = torture_join_dom_netbios_name(s->tj);
847|        u.in.source_dsa_address         = lp_parm_string(-1, "torture", "host");
848|        u.in.dest_dsa_netbios_name      = s->netbios_name;
849|
850|        status = libnet_UnbecomeDC(s->ctx, s, &u);
851|        if (!NT_STATUS_IS_OK(status)) {
852|                printf("libnet_UnbecomeDC() failed - %s\n", nt_errstr(status));
853|                ret = False;
854|        }
855|
856|        /* Leave domain. */
857|        torture_leave_domain(s->tj);
858|
859|        talloc_free(s);
860|        return ret;
861|}
```

A8. source/libnet/libnet_become_dc.c

source/libnet/libnet_become_dc.c:

```
   1|/*
   2|   Unix SMB/CIFS implementation.
   3|
   4|   Copyright (C) Stefan Metzmacher <metze@samba.org> 2006
   5|
   6|   This program is free software; you can redistribute it and/or modify
   7|   it under the terms of the GNU General Public License as published by
   8|   the Free Software Foundation; either version 2 of the License, or
   9|   (at your option) any later version.
  10|
  11|   This program is distributed in the hope that it will be useful,
  12|   but WITHOUT ANY WARRANTY; without even the implied warranty of
  13|   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  14|   GNU General Public License for more details.
  15|
  16|   You should have received a copy of the GNU General Public License
  17|   along with this program; if not, write to the Free Software
  18|   Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
  19|*/
  20|
  21|#include "includes.h"
  22|#include "libnet/libnet.h"
  23|#include "libcli/composite/composite.h"
  24|#include "libcli/cldap/cldap.h"
  25|#include "lib/ldb/include/ldb.h"
  26|#include "lib/ldb/include/ldb_errors.h"
  27|#include "lib/db_wrap.h"
  28|#include "dsdb/samdb/samdb.h"
  29|#include "dsdb/common/flags.h"
  30|#include "librpc/gen_ndr/ndr_drsuapi_c.h"
  31|#include "libcli/security/security.h"
  32|#include "librpc/gen_ndr/ndr_misc.h"
  33|#include "librpc/gen_ndr/ndr_security.h"
  34|#include "librpc/gen_ndr/ndr_drsuapi.h"
  35|#include "auth/gensec/gensec.h"
  36|
  37|/****************************************************************************
  38| * Windows 2003 (w2k3) does the following steps when changing the server role
  39| * from domain member to domain controller
  40| *
  41| * We mostly do the same.
  42| ****************************************************************************/
  43|
  44|/*
  45| * lookup DC:
  46| * - using nbt name<1C> request and a samlogon mailslot request
  47| * or
  48| * - using a DNS SRV _ldap._tcp.dc._msdcs. request and a CLDAP netlogon request
  49| *
  50| * see: becomeDC_recv_cldap() and becomeDC_send_cldap()
  51| */
  52|
  53|/*
  54| * Open 1st LDAP connection to the DC using admin credentials
  55| *
  56| * see: becomeDC_connect_ldap1() and becomeDC_ldap_connect()
  57| */
  58|
  59|/*
  60| * LDAP search 1st LDAP connection:
  61| *
  62| * see: becomeDC_ldap1_rootdse()
  63| *
  64| * Request:
  65| *      basedn: ""
  66| *      scope:  base
  67| *      filter: (objectClass=*)
  68| *      attrs:  *
  69| * Result:
  70| *      ""
  71| *              currentTime:            20061202155100.0Z
  72| *              subschemaSubentry:      CN=Aggregate,CN=Schema,CN=Configuration,<domain_p
->   artition>
  73| *              dsServiceName:          CN=<netbios_name>,CN=Servers,CN=<site_name>,CN=Si
->   tes,CN=Configuration,<domain_partition>
  74| *              namingContexts:         <domain_partition>
  75| *                                      CN=Configuration,<domain_partition>
  76| *                                      CN=Schema,CN=Configuration,<domain_partition>
  77| *              defaultNamingContext:   <domain_partition>
```

```
  78| *              schemaNamingContext:    CN=Schema,CN=Configuration,<domain_partition>
  79| *              configurationNamingContext:CN=Configuration,<domain_partition>
  80| *              rootDomainNamingContext:<domain_partition>
  81| *              supportedControl:       ...
  82| *              supportedLDAPVersion:   3
  83| *                                      2
  84| *              supportedLDAPPolicies:  ...
  85| *              highestCommitedUSN:     ...
  86| *              supportedSASLMechanisms:GSSAPI
  87| *                                      GSS-SPNEGO
  88| *                                      EXTERNAL
  89| *                                      DIGEST-MD5
  90| *              dnsHostName:            <dns_host_name>
  91| *              ldapServiceName:        <domain_dns_name>:<netbios_name>$@<REALM>
  92| *              serverName:             CN=Servers,CN=<site_name>,CN=Sites,CN=Configurati
->   on,<domain_partition>
  93| *              supportedCapabilities:  ...
  94| *              isSyncronized:          TRUE
  95| *              isGlobalCatalogReady:   TRUE
  96| *              domainFunctionality:    0
  97| *              forestFunctionality:    0
  98| *              domainControllerFunctionality: 2
  99| */
 100|
 101|/*
 102| * LDAP search 1st LDAP connection:
 103| *
 104| * see: becomeDC_ldap1_crossref_behavior_version()
 105| *
 106| * Request:
 107| *      basedn: CN=Configuration,<domain_partition>
 108| *      scope:  one
 109| *      filter: (cn=Partitions)
 110| *      attrs:  msDS-Behavior-Version
 111| * Result:
 112| *      CN=Partitions,CN=Configuration,<domain_partition>
 113| *              msDS-Behavior-Version:  0
 114| */
 115|
 116|/*
 117| * LDAP search 1st LDAP connection:
 118| *
 119| * NOTE: this seems to be a bug! as the messageID of the LDAP message is corrupted!
 120| *
 121| * not implemented here
 122| *
 123| * Request:
 124| *      basedn: CN=Schema,CN=Configuration,<domain_partition>
 125| *      scope:  one
 126| *      filter: (cn=Partitions)
 127| *      attrs:  msDS-Behavior-Version
 128| * Result:
 129| *      <none>
 130| *
 131| */
 132|
 133|/*
 134| * LDAP search 1st LDAP connection:
 135| *
 136| * see: becomeDC_ldap1_domain_behavior_version()
 137| *
 138| * Request:
 139| *      basedn: <domain_partition>
 140| *      scope:  base
 141| *      filter: (objectClass=*)
 142| *      attrs:  msDS-Behavior-Version
 143| * Result:
 144| *      <domain_partition>
 145| *              msDS-Behavior-Version:  0
 146| */
 147|
 148|/*
 149| * LDAP search 1st LDAP connection:
 150| *
 151| * see: becomeDC_ldap1_schema_object_version()
 152| *
 153| * Request:
 154| *      basedn: CN=Schema,CN=Configuration,<domain_partition>
 155| *      scope:  base
 156| *      filter: (objectClass=*)
 157| *      attrs:  objectVersion
 158| * Result:
 159| *      CN=Schema,CN=Configuration,<domain_partition>
 160| *              objectVersion:  30
 161| */
 162|
 163|/*
```

```
164| * LDAP search 1st LDAP connection:
165| *
166| * not implemented, because the information is already there
167| *
168| * Request:
169| *      basedn: ""
170| *      scope:  base
171| *      filter: (objectClass=*)
172| *      attrs:  defaultNamingContext
173| *              dnsHostName
174| * Result:
175| *      ""
176| *              defaultNamingContext:   <domain_partition>
177| *              dnsHostName:            <dns_host_name>
178| */
179|
180|/*
181| * LDAP search 1st LDAP connection:
182| *
183| * see: becomeDC_ldap1_infrastructure_fsmo()
184| *
185| * Request:
186| *      basedn: <WKGUID=2fbac1870ade11d297c400c04fd8d5cd,domain_partition>
187| *      scope:  base
188| *      filter: (objectClass=*)
189| *      attrs:  1.1
190| * Result:
191| *      CN=Infrastructure,<domain_partition>
192| */
193|
194|/*
195| * LDAP search 1st LDAP connection:
196| *
197| * see: becomeDC_ldap1_w2k3_update_revision()
198| *
199| * Request:
200| *      basedn: CN=Windows2003Update,CN=DomainUpdates,CN=System,<domain_partition>
201| *      scope:  base
202| *      filter: (objectClass=*)
203| *      attrs:  revision
204| * Result:
205| *      CN=Windows2003Update,CN=DomainUpdates,CN=System,<domain_partition>
206| *              revision:       8
207| */
208|
209|/*
210| * LDAP search 1st LDAP connection:
211| *
212| * see: becomeDC_ldap1_infrastructure_fsmo()
213| *
214| * Request:
215| *      basedn: CN=Infrastructure,<domain_partition>
216| *      scope:  base
217| *      filter: (objectClass=*)
218| *      attrs:  fSMORoleOwner
219| * Result:
220| *      CN=Infrastructure,<domain_partition>
221| *              fSMORoleOwner:  CN=NTDS Settings,<infrastructure_fsmo_server_object>
222| */
223|
224|/*
225| * LDAP search 1st LDAP connection:
226| *
227| * see: becomeDC_ldap1_infrastructure_fsmo()
228| *
229| * Request:
230| *      basedn: <infrastructure_fsmo_server_object>
231| *      scope:  base
232| *      filter: (objectClass=*)
233| *      attrs:  dnsHostName
234| * Result:
235| *      <infrastructure_fsmo_server_object>
236| *              dnsHostName:    <dns_host_name>
237| */
238|
239|/*
240| * LDAP search 1st LDAP connection:
241| *
242| * see: becomeDC_ldap1_infrastructure_fsmo()
243| *
244| * Request:
245| *      basedn: CN=NTDS Settings,<infrastructure_fsmo_server_object>
246| *      scope:  base
247| *      filter: (objectClass=*)
248| *      attrs:  objectGUID
249| * Result:
250| *      CN=NTDS Settings,<infrastructure_fsmo_server_object>
```

```
 251| *              objectGUID:     <object_guid>
 252| */
 253|
 254|/*
 255| * LDAP search 1st LDAP connection:
 256| *
 257| * see: becomeDC_ldap1_rid_manager_fsmo()
 258| *
 259| * Request:
 260| *      basedn: <domain_partition>
 261| *      scope:  base
 262| *      filter: (objectClass=*)
 263| *      attrs:  rIDManagerReference
 264| * Result:
 265| *      <domain_partition>
 266| *              rIDManagerReference:    CN=RID Manager$,CN=System,<domain_partition>
 267| */
 268|
 269|/*
 270| * LDAP search 1st LDAP connection:
 271| *
 272| * see: becomeDC_ldap1_rid_manager_fsmo()
 273| *
 274| * Request:
 275| *      basedn: CN=RID Manager$,CN=System,<domain_partition>
 276| *      scope:  base
 277| *      filter: (objectClass=*)
 278| *      attrs:  fSMORoleOwner
 279| * Result:
 280| *      CN=Infrastructure,<domain_partition>
 281| *              fSMORoleOwner:  CN=NTDS Settings,<rid_manager_fsmo_server_object>
 282| */
 283|
 284|/*
 285| * LDAP search 1st LDAP connection:
 286| *
 287| * see: becomeDC_ldap1_rid_manager_fsmo()
 288| *
 289| * Request:
 290| *      basedn: <rid_manager_fsmo_server_object>
 291| *      scope:  base
 292| *      filter: (objectClass=*)
 293| *      attrs:  dnsHostName
 294| * Result:
 295| *      <rid_manager_fsmo_server_object>
 296| *              dnsHostName:    <dns_host_name>
 297| */
 298|
 299|/*
 300| * LDAP search 1st LDAP connection:
 301| *
 302| * see: becomeDC_ldap1_rid_manager_fsmo()
 303| *
 304| * Request:
 305| *      basedn: CN=NTDS Settings,<rid_manager_fsmo_server_object>
 306| *      scope:  base
 307| *      filter: (objectClass=*)
 308| *      attrs:  msDs-ReplicationEpoch
 309| * Result:
 310| *      CN=NTDS Settings,<rid_manager_fsmo_server_object>
 311| */
 312|
 313|/*
 314| * LDAP search 1st LDAP connection:
 315| *
 316| * see: becomeDC_ldap1_site_object()
 317| *
 318| * Request:
 319| *      basedn: CN=<new_dc_site_name>,CN=Sites,CN=Configuration,<domain_partition>
 320| *      scope:  base
 321| *      filter: (objectClass=*)
 322| *      attrs:
 323| * Result:
 324| *      CN=<new_dc_site_name>,CN=Sites,CN=Configuration,<domain_partition>
 325| *              objectClass:    top
 326| *                              site
 327| *              cn:             <new_dc_site_name>
 328| *              distinguishedName:CN=<new_dc_site_name>,CN=Sites,CN=Configuration,<domain
->   _partition>
 329| *              instanceType:   4
 330| *              whenCreated:    ...
 331| *              whenChanged:    ...
 332| *              uSNCreated:     ...
 333| *              uSNChanged:     ...
 334| *              showInAdvancedViewOnly: TRUE
 335| *              name:           <new_dc_site_name>
 336| *              objectGUID:     <object_guid>
```

```
 337| *              systemFlags:    1107296256 <0x42000000>
 338| *              objectCategory: CN=Site,C=Schema,CN=Configuration,<domain_partition>
 339| */
 340|
 341|/*************************************************************
 342| * Add this stage we call the check_options() callback function
 343| * of the caller, to see if he wants us to continue
 344| *
 345| * see: becomeDC_check_options()
 346| *************************************************************/
 347|
 348|/*
 349| * LDAP search 1st LDAP connection:
 350| *
 351| * see: becomeDC_ldap1_computer_object()
 352| *
 353| * Request:
 354| *      basedn: <domain_partition>
 355| *      scope:  sub
 356| *      filter: (&(|(objectClass=user)(objectClass=computer))(sAMAccountName=<new_dc_acco
->   unt_name>))
 357| *      attrs:  distinguishedName
 358| *              userAccountControl
 359| * Result:
 360| *      CN=<new_dc_netbios_name>,CN=Computers,<domain_partition>
 361| *              distinguishedName:      CN=<new_dc_netbios_name>,CN=Computers,<domain_par
->   tition>
 362| *              userAccoountControl:    4096 <0x1000>
 363| */
 364|
 365|/*
 366| * LDAP search 1st LDAP connection:
 367| *
 368| * see: becomeDC_ldap1_server_object_1()
 369| *
 370| * Request:
 371| *      basedn: CN=<new_dc_netbios_name>,CN=Servers,CN=<new_dc_site_name>,CN=Sites,CN=Con
->   figuration,<domain_partition>
 372| *      scope:  base
 373| *      filter: (objectClass=*)
 374| *      attrs:
 375| * Result:
 376| *      <noSuchObject>
 377| *      <matchedDN:CN=Servers,CN=<new_dc_site_name>,CN=Sites,CN=Configuration,<domain_par
->   tition>>
 378| */
 379|
 380|/*
 381| * LDAP search 1st LDAP connection:
 382| *
 383| * see: becomeDC_ldap1_server_object_2()
 384| *
 385| * Request:
 386| *      basedn: CN=<new_dc_netbios_name>,CN=Computers,<domain_partition>
 387| *      scope:  base
 388| *      filter: (objectClass=*)
 389| *      attrs:  serverReferenceBL
 390| *      typesOnly: TRUE!!!
 391| * Result:
 392| *      CN=<new_dc_netbios_name>,CN=Computers,<domain_partition>
 393| */
 394|
 395|/*
 396| * LDAP add 1st LDAP connection:
 397| *
 398| * see: becomeDC_ldap1_server_object_add()
 399| *
 400| * Request:
 401| *      CN=<new_dc_netbios_name>,CN=Computers,<domain_partition>
 402| *      objectClass:    server
 403| *      systemFlags:    50000000 <0x2FAF080>
 404| *      serverReference:CN=<new_dc_netbios_name>,CN=Computers,<domain_partition>
 405| * Result:
 406| *      <success>
 407| */
 408|
 409|/*
 410| * LDAP search 1st LDAP connection:
 411| *
 412| * not implemented, maybe we can add that later
 413| *
 414| * Request:
 415| *      basedn: CN=NTDS Settings,CN=<new_dc_netbios_name>,CN=Servers,CN=<new_dc_site_name
->   >,CN=Sites,CN=Configuration,<domain_partition>
 416| *      scope:  base
 417| *      filter: (objectClass=*)
 418| *      attrs:
```

```
 419| * Result:
 420| *      <noSuchObject>
 421| *      <matchedDN:CN=<new_dc_netbios_name>,CN=Servers,CN=<new_dc_site_name>,CN=Sites,CN=
->   Configuration,<domain_partition>>
 422| */
 423|
 424|/*
 425| * LDAP search 1st LDAP connection:
 426| *
 427| * not implemented because it gives no new information
 428| *
 429| * Request:
 430| *      basedn: CN=Partitions,CN=Configuration,<domain_partition>
 431| *      scope:  sub
 432| *      filter: (nCName=<domain_partition>)
 433| *      attrs:  nCName
 434| *              dnsRoot
 435| *      controls: LDAP_SERVER_EXTENDED_DN_OID:critical=false
 436| * Result:
 437| *      <GUID=<hex_guid>>;CN=<domain_netbios_name>,CN=Partitions,<domain_partition>>
 438| *              nCName:         <GUID=<hex_guid>>;<SID=<hex_sid>>;<domain_partition>>
 439| *              dnsRoot:        <domain_dns_name>
 440| */
 441|
 442|/*
 443| * LDAP modify 1st LDAP connection:
 444| *
 445| * see: becomeDC_ldap1_server_object_modify()
 446| *
 447| * Request (add):
 448| *      CN=<new_dc_netbios_name>,CN=Servers,CN=<new_dc_site_name>,CN=Sites,CN=Configurati
->   on,<domain_partition>>
 449| *      serverReference:CN=<new_dc_netbios_name>,CN=Computers,<domain_partition>
 450| * Result:
 451| *      <attributeOrValueExist>
 452| */
 453|
 454|/*
 455| * LDAP modify 1st LDAP connection:
 456| *
 457| * see: becomeDC_ldap1_server_object_modify()
 458| *
 459| * Request (replace):
 460| *      CN=<new_dc_netbios_name>,CN=Servers,CN=<new_dc_site_name>,CN=Sites,CN=Configurati
->   on,<domain_partition>>
 461| *      serverReference:CN=<new_dc_netbios_name>,CN=Computers,<domain_partition>
 462| * Result:
 463| *      <success>
 464| */
 465|
 466|/*
 467| * Open 1st DRSUAPI connection to the DC using admin credentials
 468| * DsBind with DRSUAPI_DS_BIND_GUID_W2K3 ("6afab99c-6e26-464a-975f-f58f105218bc")
 469| * (w2k3 does 2 DsBind() calls here..., where is first is unused and contains garbage at
->    the end)
 470| *
 471| * see: becomeDC_drsuapi_connect_send(), becomeDC_drsuapi1_connect_recv(),
 472| *      becomeDC_drsuapi_bind_send(), becomeDC_drsuapi_bind_recv() and
->    becomeDC_drsuapi1_bind_recv()
 473| */
 474|
 475|/*
 476| * DsAddEntry to create the CN=NTDS Settings,CN=<machine_name>,CN=Servers,CN=Default-Firs
->   t-Site-Name, ...
 477| * on the 1st DRSUAPI connection
 478| *
 479| * see: becomeDC_drsuapi1_add_entry_send() and becomeDC_drsuapi1_add_entry_recv()
 480| */
 481|
 482|/*************************************************************
 483| * Add this stage we call the prepare_db() callback function
 484| * of the caller, to see if he wants us to continue
 485| *
 486| * see: becomeDC_prepare_db()
 487| *************************************************************/
 488|
 489|/*
 490| * Open 2nd and 3rd DRSUAPI connection to the DC using admin credentials
 491| * - a DsBind with DRSUAPI_DS_BIND_GUID_W2K3 ("6afab99c-6e26-464a-975f-f58f105218bc")
 492| *   on the 2nd connection
 493| *
 494| * see: becomeDC_drsuapi_connect_send(), becomeDC_drsuapi2_connect_recv(),
 495| *      becomeDC_drsuapi_bind_send(), becomeDC_drsuapi_bind_recv(),
->    becomeDC_drsuapi2_bind_recv()
 496| *      and becomeDC_drsuapi3_connect_recv()
 497| */
 498|
```

```
 499|/*
 500| * replicate CN=Schema,CN=Configuration,...
 501| * on the 3rd DRSUAPI connection and the bind_handle from the 2nd connection
 502| *
 503| * see: becomeDC_drsuapi_pull_partition_send(), becomeDC_drsuapi_pull_partition_recv(),
 504| *      becomeDC_drsuapi3_pull_schema_send() and becomeDC_drsuapi3_pull_schema_recv()
 505| *
 506| ************************************************************
 507| * Add this stage we call the schema_chunk() callback function
 508| * for each replication message
 509| ************************************************************/
 510|
 511|/*
 512| * replicate CN=Configuration,...
 513| * on the 3rd DRSUAPI connection and the bind_handle from the 2nd connection
 514| *
 515| * see: becomeDC_drsuapi_pull_partition_send(), becomeDC_drsuapi_pull_partition_recv(),
 516| *      becomeDC_drsuapi3_pull_config_send() and becomeDC_drsuapi3_pull_config_recv()
 517| *
 518| ************************************************************
 519| * Add this stage we call the config_chunk() callback function
 520| * for each replication message
 521| ************************************************************/
 522|
 523|/*
 524| * LDAP unbind on the 1st LDAP connection
 525| *
 526| * not implemented, because it's not needed...
 527| */
 528|
 529|/*
 530| * Open 2nd LDAP connection to the DC using admin credentials
 531| *
 532| * see: becomeDC_connect_ldap2() and becomeDC_ldap_connect()
 533| */
 534|
 535|/*
 536| * LDAP search 2nd LDAP connection:
 537| *
 538| * not implemented because it gives no new information
 539| * same as becomeDC_ldap1_computer_object()
 540| *
 541| * Request:
 542| *      basedn: <domain_partition>
 543| *      scope:  sub
 544| *      filter: (&(|(objectClass=user)(objectClass=computer))(sAMAccountName=<new_dc_acco
->   unt_name>))
 545| *      attrs:  distinguishedName
 546| *              userAccountControl
 547| * Result:
 548| *      CN=<new_dc_netbios_name>,CN=Computers,<domain_partition>
 549| *              distinguishedName:      CN=<new_dc_netbios_name>,CN=Computers,<domain_par
->   tition>
 550| *              userAccoountControl:    4096 <0x00001000>
 551| */
 552|
 553|/*
 554| * LDAP search 2nd LDAP connection:
 555| *
 556| * not implemented because it gives no new information
 557| * same as becomeDC_ldap1_computer_object()
 558| *
 559| * Request:
 560| *      basedn: CN=<new_dc_netbios_name>,CN=Computers,<domain_partition>
 561| *      scope:  base
 562| *      filter: (objectClass=*)
 563| *      attrs:  userAccountControl
 564| * Result:
 565| *      CN=<new_dc_netbios_name>,CN=Computers,<domain_partition>
 566| *              userAccoountControl:    4096 <0x00001000>
 567| */
 568|
 569|/*
 570| * LDAP modify 2nd LDAP connection:
 571| *
 572| * see: becomeDC_ldap2_modify_computer()
 573| *
 574| * Request (replace):
 575| *      CN=<new_dc_netbios_name>,CN=Computers,<domain_partition>
 576| *      userAccoountControl:    532480 <0x82000>
 577| * Result:
 578| *      <success>
 579| */
 580|
 581|/*
 582| * LDAP search 2nd LDAP connection:
 583| *
```

```
 584| * see: becomeDC_ldap2_move_computer()
 585| *
 586| * Request:
 587| *      basedn: <WKGUID=2fbac1870ade11d297c400c04fd8d5cd,<domain_partition>>
 588| *      scope:  base
 589| *      filter: (objectClass=*)
 590| *      attrs:  1.1
 591| * Result:
 592| *      CN=Domain Controllers,<domain_partition>
 593| */
 594|
 595|/*
 596| * LDAP search 2nd LDAP connection:
 597| *
 598| * not implemented because it gives no new information
 599| *
 600| * Request:
 601| *      basedn: CN=Domain Controllers,<domain_partition>
 602| *      scope:  base
 603| *      filter: (objectClass=*)
 604| *      attrs:  distinguishedName
 605| * Result:
 606| *      CN=Domain Controller,<domain_partition>
 607| *              distinguishedName:      CN=Domain Controllers,<domain_partition>
 608| */
 609|
 610|/*
 611| * LDAP modifyRDN 2nd LDAP connection:
 612| *
 613| * see: becomeDC_ldap2_move_computer()
 614| *
 615| * Request:
 616| *      entry:          CN=<new_dc_netbios_name>,CN=Computers,<domain_partition>
 617| *      newrdn:         CN=<new_dc_netbios_name>
 618| *      deleteoldrdn:   TRUE
 619| *      newparent:      CN=Domain Controllers,<domain_partition>
 620| * Result:
 621| *      <success>
 622| */
 623|
 624|/*
 625| * LDAP unbind on the 2nd LDAP connection
 626| *
 627| * not implemented, because it's not needed...
 628| */
 629|
 630|/*
 631| * replicate Domain Partition
 632| * on the 3rd DRSUAPI connection and the bind_handle from the 2nd connection
 633| *
 634| * see: becomeDC_drsuapi_pull_partition_send(), becomeDC_drsuapi_pull_partition_recv(),
 635| *      becomeDC_drsuapi3_pull_domain_send() and becomeDC_drsuapi3_pull_domain_recv()
 636| *
 637| ************************************************************
 638| * Add this stage we call the domain_chunk() callback function
 639| * for each replication message
 640| ************************************************************/
 641|
 642|/* call DsReplicaUpdateRefs() for all partitions like this:
 643| *     req1: struct drsuapi_DsReplicaUpdateRefsRequest1
 644| *
 645| *                 naming_context: struct drsuapi_DsReplicaObjectIdentifier
 646| *                     __ndr_size              : 0x000000ae (174)
 647| *                     __ndr_size_sid          : 0x00000000 (0)
 648| *                     guid                    : 00000000-0000-0000-0000-000000000000
 649| *                     sid                     : S-0-0
 650| *                     dn                      : 'CN=Schema,CN=Configuration,DC=w2k3,DC=
->   vmnet1,DC=vm,DC=base'
 651| *
 652| *                 dest_dsa_dns_name       : '4a0df188-a0b8-47ea-bbe5-e614723f16dd._msdc
->   s.w2k3.vmnet1.vm.base'
 653| *           dest_dsa_guid           : 4a0df188-a0b8-47ea-bbe5-e614723f16dd
 654| *           options                 : 0x0000001c (28)
 655| *                 0: DRSUAPI_DS_REPLICA_UPDATE_ASYNCHRONOUS_OPERATION
 656| *                 0: DRSUAPI_DS_REPLICA_UPDATE_WRITEABLE
 657| *                 1: DRSUAPI_DS_REPLICA_UPDATE_ADD_REFERENCE
 658| *                 1: DRSUAPI_DS_REPLICA_UPDATE_DELETE_REFERENCE
 659| *                 1: DRSUAPI_DS_REPLICA_UPDATE_0x00000010
 660| *
 661| * 4a0df188-a0b8-47ea-bbe5-e614723f16dd is the objectGUID the DsAddEntry() returned for
->    the
 662| * CN=NTDS Settings,CN=<machine_name>,CN=Servers,CN=Default-First-Site-Name, ...
 663| * on the 2nd!!! DRSUAPI connection
 664| *
 665| * see: becomeDC_drsuapi_update_refs_send(), becomeDC_drsuapi2_update_refs_schema_recv(),
->
 666| *      becomeDC_drsuapi2_update_refs_config_recv() and becomeDC_drsuapi2_update_refs_dom
```

```
->   ain_recv()
 667| */
 668|
 669|/*
 670| * Windows does opens the 4th and 5th DRSUAPI connection...
 671| * and does a DsBind() with the objectGUID from DsAddEntry() as bind_guid
 672| * on the 4th connection
 673| *
 674| * and then 2 full replications of the domain partition on the 5th connection
 675| * with the bind_handle from the 4th connection
 676| *
 677| * not implemented because it gives no new information
 678| */
 679|
 680|struct libnet_BecomeDC_state {
 681|        struct composite_context *creq;
 682|
 683|        struct libnet_context *libnet;
 684|
 685|        struct dom_sid zero_sid;
 686|
 687|        struct {
 688|                struct cldap_socket *sock;
 689|                struct cldap_netlogon io;
 690|                struct nbt_cldap_netlogon_5 netlogon5;
 691|        } cldap;
 692|
 693|        struct becomeDC_ldap {
 694|                struct ldb_context *ldb;
 695|                const struct ldb_message *rootdse;
 696|        } ldap1, ldap2;
 697|
 698|        struct becomeDC_drsuapi {
 699|                struct libnet_BecomeDC_state *s;
 700|                struct dcerpc_binding *binding;
 701|                struct dcerpc_pipe *pipe;
 702|                DATA_BLOB gensec_skey;
 703|                struct drsuapi_DsBind bind_r;
 704|                struct GUID bind_guid;
 705|                struct drsuapi_DsBindInfoCtr bind_info_ctr;
 706|                struct drsuapi_DsBindInfo28 local_info28;
 707|                struct drsuapi_DsBindInfo28 remote_info28;
 708|                struct policy_handle bind_handle;
 709|        } drsuapi1, drsuapi2, drsuapi3;
 710|
 711|        struct libnet_BecomeDC_Domain domain;
 712|        struct libnet_BecomeDC_Forest forest;
 713|        struct libnet_BecomeDC_SourceDSA source_dsa;
 714|        struct libnet_BecomeDC_DestDSA dest_dsa;
 715|
 716|        struct libnet_BecomeDC_Partition schema_part, config_part, domain_part;
 717|
 718|        struct becomeDC_fsmo {
 719|                const char *dns_name;
 720|                const char *server_dn_str;
 721|                const char *ntds_dn_str;
 722|                struct GUID ntds_guid;
 723|        } infrastructure_fsmo;
 724|
 725|        struct becomeDC_fsmo rid_manager_fsmo;
 726|
 727|        struct libnet_BecomeDC_CheckOptions _co;
 728|        struct libnet_BecomeDC_PrepareDB _pp;
 729|        struct libnet_BecomeDC_StoreChunk _sc;
 730|        struct libnet_BecomeDC_Callbacks callbacks;
 731|};
 732|
 733|static void becomeDC_recv_cldap(struct cldap_request *req);
 734|
 735|static void becomeDC_send_cldap(struct libnet_BecomeDC_state *s)
 736|{
 737|        struct composite_context *c = s->creq;
 738|        struct cldap_request *req;
 739|
 740|        s->cldap.io.in.dest_address     = s->source_dsa.address;
 741|        s->cldap.io.in.realm           = s->domain.dns_name;
 742|        s->cldap.io.in.host            = s->dest_dsa.netbios_name;
 743|        s->cldap.io.in.user            = NULL;
 744|        s->cldap.io.in.domain_guid     = NULL;
 745|        s->cldap.io.in.domain_sid      = NULL;
 746|        s->cldap.io.in.acct_control    = -1;
 747|        s->cldap.io.in.version         = 6;
 748|
 749|        s->cldap.sock = cldap_socket_init(s, s->libnet->event_ctx);
 750|        if (composite_nomem(s->cldap.sock, c)) return;
 751|
 752|        req = cldap_netlogon_send(s->cldap.sock, &s->cldap.io);
```

```
753|        if (composite_nomem(req, c)) return;
754|        req->async.fn           = becomeDC_recv_cldap;
755|        req->async.private      = s;
756|}
757|
758|static void becomeDC_connect_ldap1(struct libnet_BecomeDC_state *s);
759|
760|static void becomeDC_recv_cldap(struct cldap_request *req)
761|{
762|        struct libnet_BecomeDC_state *s = talloc_get_type(req->async.private,
763|                                             struct libnet_BecomeDC_state);
764|        struct composite_context *c = s->creq;
765|
766|        c->status = cldap_netlogon_recv(req, s, &s->cldap.io);
767|        if (!composite_is_ok(c)) return;
768|
769|        s->cldap.netlogon5 = s->cldap.io.out.netlogon.logon5;
770|
771|        s->domain.dns_name             = s->cldap.netlogon5.dns_domain;
772|        s->domain.netbios_name         = s->cldap.netlogon5.domain;
773|        s->domain.guid                 = s->cldap.netlogon5.domain_uuid;
774|
775|        s->forest.dns_name             = s->cldap.netlogon5.forest;
776|
777|        s->source_dsa.dns_name         = s->cldap.netlogon5.pdc_dns_name;
778|        s->source_dsa.netbios_name     = s->cldap.netlogon5.pdc_name;
779|        s->source_dsa.site_name        = s->cldap.netlogon5.server_site;
780|
781|        s->dest_dsa.site_name          = s->cldap.netlogon5.client_site;
782|
783|        becomeDC_connect_ldap1(s);
784|}
785|
786|static NTSTATUS becomeDC_ldap_connect(struct libnet_BecomeDC_state *s, struct
->    becomeDC_ldap *ldap)
787|{
788|        char *url;
789|
790|        url = talloc_asprintf(s, "ldap://%s/", s->source_dsa.dns_name);
791|        NT_STATUS_HAVE_NO_MEMORY(url);
792|
793|        ldap->ldb = ldb_wrap_connect(s, url,
794|                                      NULL,
795|                                      s->libnet->cred,
796|                                      0, NULL);
797|        talloc_free(url);
798|        if (ldap->ldb == NULL) {
799|                return NT_STATUS_UNEXPECTED_NETWORK_ERROR;
800|        }
801|
802|        return NT_STATUS_OK;
803|}
804|
805|static NTSTATUS becomeDC_ldap1_rootdse(struct libnet_BecomeDC_state *s)
806|{
807|        int ret;
808|        struct ldb_result *r;
809|        struct ldb_dn *basedn;
810|        static const char *attrs[] = {
811|                "*",
812|                NULL
813|        };
814|
815|        basedn = ldb_dn_new(s, s->ldap1.ldb, NULL);
816|        NT_STATUS_HAVE_NO_MEMORY(basedn);
817|
818|        ret = ldb_search(s->ldap1.ldb, basedn, LDB_SCOPE_BASE,
819|                         "(objectClass=*)", attrs, &r);
820|        talloc_free(basedn);
821|        if (ret != LDB_SUCCESS) {
822|                return NT_STATUS_LDAP(ret);
823|        } else if (r->count != 1) {
824|                talloc_free(r);
825|                return NT_STATUS_INVALID_NETWORK_RESPONSE;
826|        }
827|        talloc_steal(s, r);
828|
829|        s->ldap1.rootdse = r->msgs[0];
830|
831|        s->domain.dn_str        = ldb_msg_find_attr_as_string(s->ldap1.rootdse,
->    "defaultNamingContext", NULL);
832|        if (!s->domain.dn_str) return NT_STATUS_INVALID_NETWORK_RESPONSE;
833|
834|        s->forest.root_dn_str   = ldb_msg_find_attr_as_string(s->ldap1.rootdse,
->    "rootDomainNamingContext", NULL);
835|        if (!s->forest.root_dn_str) return NT_STATUS_INVALID_NETWORK_RESPONSE;
836|        s->forest.config_dn_str = ldb_msg_find_attr_as_string(s->ldap1.rootdse,
```

```
->    "configurationNamingContext", NULL);
 837|        if (!s->forest.config_dn_str) return NT_STATUS_INVALID_NETWORK_RESPONSE;
 838|        s->forest.schema_dn_str = ldb_msg_find_attr_as_string(s->ldap1.rootdse,
->    "schemaNamingContext", NULL);
 839|        if (!s->forest.schema_dn_str) return NT_STATUS_INVALID_NETWORK_RESPONSE;
 840|
 841|        s->source_dsa.server_dn_str     = ldb_msg_find_attr_as_string(s->ldap1.rootdse,
->    "serverName", NULL);
 842|        if (!s->source_dsa.server_dn_str) return NT_STATUS_INVALID_NETWORK_RESPONSE;
 843|        s->source_dsa.ntds_dn_str       = ldb_msg_find_attr_as_string(s->ldap1.rootdse,
->    "dsServiceName", NULL);
 844|        if (!s->source_dsa.ntds_dn_str) return NT_STATUS_INVALID_NETWORK_RESPONSE;
 845|
 846|        return NT_STATUS_OK;
 847|}
 848|
 849|static NTSTATUS becomeDC_ldap1_crossref_behavior_version(struct libnet_BecomeDC_state
->    *s)
 850|{
 851|        int ret;
 852|        struct ldb_result *r;
 853|        struct ldb_dn *basedn;
 854|        static const char *attrs[] = {
 855|                "msDs-Behavior-Version",
 856|                NULL
 857|        };
 858|
 859|        basedn = ldb_dn_new(s, s->ldap1.ldb, s->forest.config_dn_str);
 860|        NT_STATUS_HAVE_NO_MEMORY(basedn);
 861|
 862|        ret = ldb_search(s->ldap1.ldb, basedn, LDB_SCOPE_ONELEVEL,
 863|                           "(cn=Partitions)", attrs, &r);
 864|        talloc_free(basedn);
 865|        if (ret != LDB_SUCCESS) {
 866|                return NT_STATUS_LDAP(ret);
 867|        } else if (r->count != 1) {
 868|                talloc_free(r);
 869|                return NT_STATUS_INVALID_NETWORK_RESPONSE;
 870|        }
 871|
 872|        s->forest.crossref_behavior_version = ldb_msg_find_attr_as_uint(r->msgs[0],
->    "msDs-Behavior-Version", 0);
 873|
 874|        talloc_free(r);
 875|        return NT_STATUS_OK;
 876|}
 877|
 878|static NTSTATUS becomeDC_ldap1_domain_behavior_version(struct libnet_BecomeDC_state *s)
 879|{
 880|        int ret;
 881|        struct ldb_result *r;
 882|        struct ldb_dn *basedn;
 883|        static const char *attrs[] = {
 884|                "msDs-Behavior-Version",
 885|                NULL
 886|        };
 887|
 888|        basedn = ldb_dn_new(s, s->ldap1.ldb, s->domain.dn_str);
 889|        NT_STATUS_HAVE_NO_MEMORY(basedn);
 890|
 891|        ret = ldb_search(s->ldap1.ldb, basedn, LDB_SCOPE_BASE,
 892|                           "(objectClass=*)", attrs, &r);
 893|        talloc_free(basedn);
 894|        if (ret != LDB_SUCCESS) {
 895|                return NT_STATUS_LDAP(ret);
 896|        } else if (r->count != 1) {
 897|                talloc_free(r);
 898|                return NT_STATUS_INVALID_NETWORK_RESPONSE;
 899|        }
 900|
 901|        s->domain.behavior_version = ldb_msg_find_attr_as_uint(r->msgs[0],
->    "msDs-Behavior-Version", 0);
 902|
 903|        talloc_free(r);
 904|        return NT_STATUS_OK;
 905|}
 906|
 907|static NTSTATUS becomeDC_ldap1_schema_object_version(struct libnet_BecomeDC_state *s)
 908|{
 909|        int ret;
 910|        struct ldb_result *r;
 911|        struct ldb_dn *basedn;
 912|        static const char *attrs[] = {
 913|                "objectVersion",
 914|                NULL
 915|        };
 916|
```

```
 917|        basedn = ldb_dn_new(s, s->ldap1.ldb, s->forest.schema_dn_str);
 918|        NT_STATUS_HAVE_NO_MEMORY(basedn);
 919|
 920|        ret = ldb_search(s->ldap1.ldb, basedn, LDB_SCOPE_BASE,
 921|                         "(objectClass=*)", attrs, &r);
 922|        talloc_free(basedn);
 923|        if (ret != LDB_SUCCESS) {
 924|                return NT_STATUS_LDAP(ret);
 925|        } else if (r->count != 1) {
 926|                talloc_free(r);
 927|                return NT_STATUS_INVALID_NETWORK_RESPONSE;
 928|        }
 929|
 930|        s->forest.schema_object_version = ldb_msg_find_attr_as_uint(r->msgs[0],
->    "objectVersion", 0);
 931|
 932|        talloc_free(r);
 933|        return NT_STATUS_OK;
 934|}
 935|
 936|static NTSTATUS becomeDC_ldap1_w2k3_update_revision(struct libnet_BecomeDC_state *s)
 937|{
 938|        int ret;
 939|        struct ldb_result *r;
 940|        struct ldb_dn *basedn;
 941|        static const char *attrs[] = {
 942|                "revision",
 943|                NULL
 944|        };
 945|
 946|        basedn = ldb_dn_new_fmt(s, s->ldap1.ldb, "CN=Windows2003Update,CN=DomainUpdates,C
->   N=System,%s",
 947|                                 s->domain.dn_str);
 948|        NT_STATUS_HAVE_NO_MEMORY(basedn);
 949|
 950|        ret = ldb_search(s->ldap1.ldb, basedn, LDB_SCOPE_BASE,
 951|                         "(objectClass=*)", attrs, &r);
 952|        talloc_free(basedn);
 953|        if (ret == LDB_ERR_NO_SUCH_OBJECT) {
 954|                /* w2k doesn't have this object */
 955|                s->domain.w2k3_update_revision = 0;
 956|                return NT_STATUS_OK;
 957|        } else if (ret != LDB_SUCCESS) {
 958|                return NT_STATUS_LDAP(ret);
 959|        } else if (r->count != 1) {
 960|                talloc_free(r);
 961|                return NT_STATUS_INVALID_NETWORK_RESPONSE;
 962|        }
 963|
 964|        s->domain.w2k3_update_revision = ldb_msg_find_attr_as_uint(r->msgs[0],
->    "revision", 0);
 965|
 966|        talloc_free(r);
 967|        return NT_STATUS_OK;
 968|}
 969|
 970|static NTSTATUS becomeDC_ldap1_infrastructure_fsmo(struct libnet_BecomeDC_state *s)
 971|{
 972|        int ret;
 973|        struct ldb_result *r;
 974|        struct ldb_dn *basedn;
 975|        struct ldb_dn *ntds_dn;
 976|        struct ldb_dn *server_dn;
 977|        static const char *_1_1_attrs[] = {
 978|                "1.1",
 979|                NULL
 980|        };
 981|        static const char *fsmo_attrs[] = {
 982|                "fSMORoleOwner",
 983|                NULL
 984|        };
 985|        static const char *dns_attrs[] = {
 986|                "dnsHostName",
 987|                NULL
 988|        };
 989|        static const char *guid_attrs[] = {
 990|                "objectGUID",
 991|                NULL
 992|        };
 993|
 994|        basedn = ldb_dn_new_fmt(s, s->ldap1.ldb, "<WKGUID=2fbac1870ade11d297c400c04fd8d5c
->   d,%s>",
 995|                                 s->domain.dn_str);
 996|        NT_STATUS_HAVE_NO_MEMORY(basedn);
 997|
 998|        ret = ldb_search(s->ldap1.ldb, basedn, LDB_SCOPE_BASE,
 999|                         "(objectClass=*)", _1_1_attrs, &r);
```

```
1000|        talloc_free(basedn);
1001|        if (ret != LDB_SUCCESS) {
1002|                return NT_STATUS_LDAP(ret);
1003|        } else if (r->count != 1) {
1004|                talloc_free(r);
1005|                return NT_STATUS_INVALID_NETWORK_RESPONSE;
1006|        }
1007|
1008|        basedn = talloc_steal(s, r->msgs[0]->dn);
1009|        talloc_free(r);
1010|
1011|        ret = ldb_search(s->ldap1.ldb, basedn, LDB_SCOPE_BASE,
1012|                        "(objectClass=*)", fsmo_attrs, &r);
1013|        talloc_free(basedn);
1014|        if (ret != LDB_SUCCESS) {
1015|                return NT_STATUS_LDAP(ret);
1016|        } else if (r->count != 1) {
1017|                talloc_free(r);
1018|                return NT_STATUS_INVALID_NETWORK_RESPONSE;
1019|        }
1020|
1021|        s->infrastructure_fsmo.ntds_dn_str      = samdb_result_string(r->msgs[0],
->    "fSMORoleOwner", NULL);
1022|        if (!s->infrastructure_fsmo.ntds_dn_str) return NT_STATUS_INVALID_NETWORK_RESPONS
->   E;
1023|        talloc_steal(s, s->infrastructure_fsmo.ntds_dn_str);
1024|
1025|        talloc_free(r);
1026|
1027|        ntds_dn = ldb_dn_new(s, s->ldap1.ldb, s->infrastructure_fsmo.ntds_dn_str);
1028|        NT_STATUS_HAVE_NO_MEMORY(ntds_dn);
1029|
1030|        server_dn = ldb_dn_get_parent(s, ntds_dn);
1031|        NT_STATUS_HAVE_NO_MEMORY(server_dn);
1032|
1033|        s->infrastructure_fsmo.server_dn_str = ldb_dn_alloc_linearized(s, server_dn);
1034|        NT_STATUS_HAVE_NO_MEMORY(s->infrastructure_fsmo.server_dn_str);
1035|
1036|        ret = ldb_search(s->ldap1.ldb, server_dn, LDB_SCOPE_BASE,
1037|                        "(objectClass=*)", dns_attrs, &r);
1038|        if (ret != LDB_SUCCESS) {
1039|                return NT_STATUS_LDAP(ret);
1040|        } else if (r->count != 1) {
1041|                talloc_free(r);
1042|                return NT_STATUS_INVALID_NETWORK_RESPONSE;
1043|        }
1044|
1045|        s->infrastructure_fsmo.dns_name = samdb_result_string(r->msgs[0], "dnsHostName",
->    NULL);
1046|        if (!s->infrastructure_fsmo.dns_name) return NT_STATUS_INVALID_NETWORK_RESPONSE;
1047|        talloc_steal(s, s->infrastructure_fsmo.dns_name);
1048|
1049|        talloc_free(r);
1050|
1051|        ret = ldb_search(s->ldap1.ldb, ntds_dn, LDB_SCOPE_BASE,
1052|                        "(objectClass=*)", guid_attrs, &r);
1053|        if (ret != LDB_SUCCESS) {
1054|                return NT_STATUS_LDAP(ret);
1055|        } else if (r->count != 1) {
1056|                talloc_free(r);
1057|                return NT_STATUS_INVALID_NETWORK_RESPONSE;
1058|        }
1059|
1060|        s->infrastructure_fsmo.ntds_guid = samdb_result_guid(r->msgs[0], "objectGUID");
1061|
1062|        talloc_free(r);
1063|
1064|        return NT_STATUS_OK;
1065|}
1066|
1067|static NTSTATUS becomeDC_ldap1_rid_manager_fsmo(struct libnet_BecomeDC_state *s)
1068|{
1069|        int ret;
1070|        struct ldb_result *r;
1071|        struct ldb_dn *basedn;
1072|        const char *reference_dn_str;
1073|        struct ldb_dn *ntds_dn;
1074|        struct ldb_dn *server_dn;
1075|        static const char *rid_attrs[] = {
1076|                "rIDManagerReference",
1077|                NULL
1078|        };
1079|        static const char *fsmo_attrs[] = {
1080|                "fSMORoleOwner",
1081|                NULL
1082|        };
1083|        static const char *dns_attrs[] = {
```

```
1084|                "dnsHostName",
1085|                NULL
1086|        };
1087|        static const char *guid_attrs[] = {
1088|                "objectGUID",
1089|                NULL
1090|        };
1091|
1092|        basedn = ldb_dn_new(s, s->ldap1.ldb, s->domain.dn_str);
1093|        NT_STATUS_HAVE_NO_MEMORY(basedn);
1094|
1095|        ret = ldb_search(s->ldap1.ldb, basedn, LDB_SCOPE_BASE,
1096|                           "(objectClass=*)", rid_attrs, &r);
1097|        talloc_free(basedn);
1098|        if (ret != LDB_SUCCESS) {
1099|                return NT_STATUS_LDAP(ret);
1100|        } else if (r->count != 1) {
1101|                talloc_free(r);
1102|                return NT_STATUS_INVALID_NETWORK_RESPONSE;
1103|        }
1104|
1105|        reference_dn_str        = samdb_result_string(r->msgs[0], "rIDManagerReference",
->    NULL);
1106|        if (!reference_dn_str) return NT_STATUS_INVALID_NETWORK_RESPONSE;
1107|
1108|        basedn = ldb_dn_new(s, s->ldap1.ldb, reference_dn_str);
1109|        NT_STATUS_HAVE_NO_MEMORY(basedn);
1110|
1111|        talloc_free(r);
1112|
1113|        ret = ldb_search(s->ldap1.ldb, basedn, LDB_SCOPE_BASE,
1114|                           "(objectClass=*)", fsmo_attrs, &r);
1115|        talloc_free(basedn);
1116|        if (ret != LDB_SUCCESS) {
1117|                return NT_STATUS_LDAP(ret);
1118|        } else if (r->count != 1) {
1119|                talloc_free(r);
1120|                return NT_STATUS_INVALID_NETWORK_RESPONSE;
1121|        }
1122|
1123|        s->rid_manager_fsmo.ntds_dn_str = samdb_result_string(r->msgs[0],
->    "fSMORoleOwner", NULL);
1124|        if (!s->rid_manager_fsmo.ntds_dn_str) return NT_STATUS_INVALID_NETWORK_RESPONSE;
1125|        talloc_steal(s, s->rid_manager_fsmo.ntds_dn_str);
1126|
1127|        talloc_free(r);
1128|
1129|        ntds_dn = ldb_dn_new(s, s->ldap1.ldb, s->rid_manager_fsmo.ntds_dn_str);
1130|        NT_STATUS_HAVE_NO_MEMORY(ntds_dn);
1131|
1132|        server_dn = ldb_dn_get_parent(s, ntds_dn);
1133|        NT_STATUS_HAVE_NO_MEMORY(server_dn);
1134|
1135|        s->rid_manager_fsmo.server_dn_str = ldb_dn_alloc_linearized(s, server_dn);
1136|        NT_STATUS_HAVE_NO_MEMORY(s->rid_manager_fsmo.server_dn_str);
1137|
1138|        ret = ldb_search(s->ldap1.ldb, server_dn, LDB_SCOPE_BASE,
1139|                           "(objectClass=*)", dns_attrs, &r);
1140|        if (ret != LDB_SUCCESS) {
1141|                return NT_STATUS_LDAP(ret);
1142|        } else if (r->count != 1) {
1143|                talloc_free(r);
1144|                return NT_STATUS_INVALID_NETWORK_RESPONSE;
1145|        }
1146|
1147|        s->rid_manager_fsmo.dns_name    = samdb_result_string(r->msgs[0], "dnsHostName",
->    NULL);
1148|        if (!s->rid_manager_fsmo.dns_name) return NT_STATUS_INVALID_NETWORK_RESPONSE;
1149|        talloc_steal(s, s->rid_manager_fsmo.dns_name);
1150|
1151|        talloc_free(r);
1152|
1153|        ret = ldb_search(s->ldap1.ldb, ntds_dn, LDB_SCOPE_BASE,
1154|                           "(objectClass=*)", guid_attrs, &r);
1155|        if (ret != LDB_SUCCESS) {
1156|                return NT_STATUS_LDAP(ret);
1157|        } else if (r->count != 1) {
1158|                talloc_free(r);
1159|                return NT_STATUS_INVALID_NETWORK_RESPONSE;
1160|        }
1161|
1162|        s->rid_manager_fsmo.ntds_guid = samdb_result_guid(r->msgs[0], "objectGUID");
1163|
1164|        talloc_free(r);
1165|
1166|        return NT_STATUS_OK;
1167|}
```

```
1168|
1169|static NTSTATUS becomeDC_ldap1_site_object(struct libnet_BecomeDC_state *s)
1170|{
1171|        int ret;
1172|        struct ldb_result *r;
1173|        struct ldb_dn *basedn;
1174|
1175|        basedn = ldb_dn_new_fmt(s, s->ldap1.ldb, "CN=%s,CN=Sites,%s",
1176|                                 s->dest_dsa.site_name,
1177|                                 s->forest.config_dn_str);
1178|        NT_STATUS_HAVE_NO_MEMORY(basedn);
1179|
1180|        ret = ldb_search(s->ldap1.ldb, basedn, LDB_SCOPE_BASE,
1181|                          "(objectClass=*)", NULL, &r);
1182|        talloc_free(basedn);
1183|        if (ret != LDB_SUCCESS) {
1184|                return NT_STATUS_LDAP(ret);
1185|        } else if (r->count != 1) {
1186|                talloc_free(r);
1187|                return NT_STATUS_INVALID_NETWORK_RESPONSE;
1188|        }
1189|
1190|        s->dest_dsa.site_guid = samdb_result_guid(r->msgs[0], "objectGUID");
1191|
1192|        talloc_free(r);
1193|        return NT_STATUS_OK;
1194|}
1195|
1196|static NTSTATUS becomeDC_check_options(struct libnet_BecomeDC_state *s)
1197|{
1198|        if (!s->callbacks.check_options) return NT_STATUS_OK;
1199|
1200|        s->_co.domain           = &s->domain;
1201|        s->_co.forest           = &s->forest;
1202|        s->_co.source_dsa       = &s->source_dsa;
1203|
1204|        return s->callbacks.check_options(s->callbacks.private_data, &s->_co);
1205|}
1206|
1207|static NTSTATUS becomeDC_ldap1_computer_object(struct libnet_BecomeDC_state *s)
1208|{
1209|        int ret;
1210|        struct ldb_result *r;
1211|        struct ldb_dn *basedn;
1212|        char *filter;
1213|        static const char *attrs[] = {
1214|                "distinguishedName",
1215|                "userAccountControl",
1216|                NULL
1217|        };
1218|
1219|        basedn = ldb_dn_new(s, s->ldap1.ldb, s->domain.dn_str);
1220|        NT_STATUS_HAVE_NO_MEMORY(basedn);
1221|
1222|        filter = talloc_asprintf(basedn, "(&(|(objectClass=user)(objectClass=computer))(s
->   AMAccountName=%s$))",
1223|                                  s->dest_dsa.netbios_name);
1224|        NT_STATUS_HAVE_NO_MEMORY(filter);
1225|
1226|        ret = ldb_search(s->ldap1.ldb, basedn, LDB_SCOPE_SUBTREE,
1227|                          filter, attrs, &r);
1228|        talloc_free(basedn);
1229|        if (ret != LDB_SUCCESS) {
1230|                return NT_STATUS_LDAP(ret);
1231|        } else if (r->count != 1) {
1232|                talloc_free(r);
1233|                return NT_STATUS_INVALID_NETWORK_RESPONSE;
1234|        }
1235|
1236|        s->dest_dsa.computer_dn_str     = samdb_result_string(r->msgs[0],
->    "distinguishedName", NULL);
1237|        if (!s->dest_dsa.computer_dn_str) return NT_STATUS_INVALID_NETWORK_RESPONSE;
1238|        talloc_steal(s, s->dest_dsa.computer_dn_str);
1239|
1240|        s->dest_dsa.user_account_control = samdb_result_uint(r->msgs[0],
->    "userAccountControl", 0);
1241|
1242|        talloc_free(r);
1243|        return NT_STATUS_OK;
1244|}
1245|
1246|static NTSTATUS becomeDC_ldap1_server_object_1(struct libnet_BecomeDC_state *s)
1247|{
1248|        int ret;
1249|        struct ldb_result *r;
1250|        struct ldb_dn *basedn;
1251|        const char *server_reference_dn_str;
```

```
1252|        struct ldb_dn *server_reference_dn;
1253|        struct ldb_dn *computer_dn;
1254|
1255|        basedn = ldb_dn_new_fmt(s, s->ldap1.ldb, "CN=%s,CN=Servers,CN=%s,CN=Sites,%s",
1256|                                s->dest_dsa.netbios_name,
1257|                                s->dest_dsa.site_name,
1258|                                s->forest.config_dn_str);
1259|        NT_STATUS_HAVE_NO_MEMORY(basedn);
1260|
1261|        ret = ldb_search(s->ldap1.ldb, basedn, LDB_SCOPE_BASE,
1262|                           "(objectClass=*)", NULL, &r);
1263|        talloc_free(basedn);
1264|        if (ret == LDB_ERR_NO_SUCH_OBJECT) {
1265|                /* if the object doesn't exist, we'll create it later */
1266|                return NT_STATUS_OK;
1267|        } else if (ret != LDB_SUCCESS) {
1268|                return NT_STATUS_LDAP(ret);
1269|        } else if (r->count != 1) {
1270|                talloc_free(r);
1271|                return NT_STATUS_INVALID_NETWORK_RESPONSE;
1272|        }
1273|
1274|        server_reference_dn_str = samdb_result_string(r->msgs[0], "serverReference",
->    NULL);
1275|        if (server_reference_dn_str) {
1276|                server_reference_dn     = ldb_dn_new(r, s->ldap1.ldb,
->    server_reference_dn_str);
1277|                NT_STATUS_HAVE_NO_MEMORY(server_reference_dn);
1278|
1279|                computer_dn             = ldb_dn_new(r, s->ldap1.ldb,
->    s->dest_dsa.computer_dn_str);
1280|                NT_STATUS_HAVE_NO_MEMORY(computer_dn);
1281|
1282|                /*
1283|                 * if the server object belongs to another DC in another domain in the
->    forest,
1284|                 * we should not touch this object!
1285|                 */
1286|                if (ldb_dn_compare(computer_dn, server_reference_dn) != 0) {
1287|                        talloc_free(r);
1288|                        return NT_STATUS_OBJECT_NAME_COLLISION;
1289|                }
1290|        }
1291|
1292|        /* if the server object is already for the dest_dsa, then we don't need to create
->    it */
1293|        s->dest_dsa.server_dn_str       = samdb_result_string(r->msgs[0],
->    "distinguishedName", NULL);
1294|        if (!s->dest_dsa.server_dn_str) return NT_STATUS_INVALID_NETWORK_RESPONSE;
1295|        talloc_steal(s, s->dest_dsa.server_dn_str);
1296|
1297|        talloc_free(r);
1298|        return NT_STATUS_OK;
1299|}
1300|
1301|static NTSTATUS becomeDC_ldap1_server_object_2(struct libnet_BecomeDC_state *s)
1302|{
1303|        int ret;
1304|        struct ldb_result *r;
1305|        struct ldb_dn *basedn;
1306|        const char *server_reference_bl_dn_str;
1307|        static const char *attrs[] = {
1308|                "serverReferenceBL",
1309|                NULL
1310|        };
1311|
1312|        /* if the server_dn_str has a valid value, we skip this lookup */
1313|        if (s->dest_dsa.server_dn_str) return NT_STATUS_OK;
1314|
1315|        basedn = ldb_dn_new(s, s->ldap1.ldb, s->dest_dsa.computer_dn_str);
1316|        NT_STATUS_HAVE_NO_MEMORY(basedn);
1317|
1318|        ret = ldb_search(s->ldap1.ldb, basedn, LDB_SCOPE_BASE,
1319|                          "(objectClass=*)", attrs, &r);
1320|        talloc_free(basedn);
1321|        if (ret != LDB_SUCCESS) {
1322|                return NT_STATUS_LDAP(ret);
1323|        } else if (r->count != 1) {
1324|                talloc_free(r);
1325|                return NT_STATUS_INVALID_NETWORK_RESPONSE;
1326|        }
1327|
1328|        server_reference_bl_dn_str = samdb_result_string(r->msgs[0], "serverReferenceBL",
->    NULL);
1329|        if (!server_reference_bl_dn_str) {
1330|                /* if no back link is present, we're done for this function */
1331|                talloc_free(r);
```

```
1332|                return NT_STATUS_OK;
1333|        }
1334|
1335|        /* if the server object is already for the dest_dsa, then we don't need to create
->    it */
1336|        s->dest_dsa.server_dn_str       = samdb_result_string(r->msgs[0],
->    "serverReferenceBL", NULL);
1337|        if (s->dest_dsa.server_dn_str) {
1338|                /* if a back link is present, we know that the server object is present
->    */
1339|                talloc_steal(s, s->dest_dsa.server_dn_str);
1340|        }
1341|
1342|        talloc_free(r);
1343|        return NT_STATUS_OK;
1344|}
1345|
1346|static NTSTATUS becomeDC_ldap1_server_object_add(struct libnet_BecomeDC_state *s)
1347|{
1348|        int ret;
1349|        struct ldb_message *msg;
1350|        char *server_dn_str;
1351|
1352|        /* if the server_dn_str has a valid value, we skip this lookup */
1353|        if (s->dest_dsa.server_dn_str) return NT_STATUS_OK;
1354|
1355|        msg = ldb_msg_new(s);
1356|        NT_STATUS_HAVE_NO_MEMORY(msg);
1357|
1358|        msg->dn = ldb_dn_new_fmt(msg, s->ldap1.ldb, "CN=%s,CN=Servers,CN=%s,CN=Sites,%s",
->
1359|                                 s->dest_dsa.netbios_name,
1360|                                 s->dest_dsa.site_name,
1361|                                 s->forest.config_dn_str);
1362|        NT_STATUS_HAVE_NO_MEMORY(msg->dn);
1363|
1364|        ret = ldb_msg_add_string(msg, "objectClass", "server");
1365|        if (ret != 0) {
1366|                talloc_free(msg);
1367|                return NT_STATUS_NO_MEMORY;
1368|        }
1369|        ret = ldb_msg_add_string(msg, "systemFlags", "50000000");
1370|        if (ret != 0) {
1371|                talloc_free(msg);
1372|                return NT_STATUS_NO_MEMORY;
1373|        }
1374|        ret = ldb_msg_add_string(msg, "serverReference", s->dest_dsa.computer_dn_str);
1375|        if (ret != 0) {
1376|                talloc_free(msg);
1377|                return NT_STATUS_NO_MEMORY;
1378|        }
1379|
1380|        server_dn_str = ldb_dn_alloc_linearized(s, msg->dn);
1381|        NT_STATUS_HAVE_NO_MEMORY(server_dn_str);
1382|
1383|        ret = ldb_add(s->ldap1.ldb, msg);
1384|        talloc_free(msg);
1385|        if (ret != LDB_SUCCESS) {
1386|                talloc_free(server_dn_str);
1387|                return NT_STATUS_LDAP(ret);
1388|        }
1389|
1390|        s->dest_dsa.server_dn_str = server_dn_str;
1391|
1392|        return NT_STATUS_OK;
1393|}
1394|
1395|static NTSTATUS becomeDC_ldap1_server_object_modify(struct libnet_BecomeDC_state *s)
1396|{
1397|        int ret;
1398|        struct ldb_message *msg;
1399|        uint32_t i;
1400|
1401|        /* make a 'modify' msg, and only for serverReference */
1402|        msg = ldb_msg_new(s);
1403|        NT_STATUS_HAVE_NO_MEMORY(msg);
1404|        msg->dn = ldb_dn_new(msg, s->ldap1.ldb, s->dest_dsa.server_dn_str);
1405|        NT_STATUS_HAVE_NO_MEMORY(msg->dn);
1406|
1407|        ret = ldb_msg_add_string(msg, "serverReference", s->dest_dsa.computer_dn_str);
1408|        if (ret != 0) {
1409|                talloc_free(msg);
1410|                return NT_STATUS_NO_MEMORY;
1411|        }
1412|
1413|        /* mark all the message elements (should be just one)
1414|           as LDB_FLAG_MOD_ADD */
```

```
1415|        for (i=0;i<msg->num_elements;i++) {
1416|                msg->elements[i].flags = LDB_FLAG_MOD_ADD;
1417|        }
1418|
1419|        ret = ldb_modify(s->ldap1.ldb, msg);
1420|        if (ret == LDB_SUCCESS) {
1421|                talloc_free(msg);
1422|                return NT_STATUS_OK;
1423|        } else if (ret == LDB_ERR_ATTRIBUTE_OR_VALUE_EXISTS) {
1424|                /* retry with LDB_FLAG_MOD_REPLACE */
1425|        } else {
1426|                talloc_free(msg);
1427|                return NT_STATUS_LDAP(ret);
1428|        }
1429|
1430|        /* mark all the message elements (should be just one)
1431|           as LDB_FLAG_MOD_REPLACE */
1432|        for (i=0;i<msg->num_elements;i++) {
1433|                msg->elements[i].flags = LDB_FLAG_MOD_REPLACE;
1434|        }
1435|
1436|        ret = ldb_modify(s->ldap1.ldb, msg);
1437|        talloc_free(msg);
1438|        if (ret != LDB_SUCCESS) {
1439|                return NT_STATUS_LDAP(ret);
1440|        }
1441|
1442|        return NT_STATUS_OK;
1443|}
1444|
1445|static void becomeDC_drsuapi_connect_send(struct libnet_BecomeDC_state *s,
1446|                                          struct becomeDC_drsuapi *drsuapi,
1447|                                          void (*recv_fn)(struct composite_context
->    *req));
1448|static void becomeDC_drsuapi1_connect_recv(struct composite_context *req);
1449|static void becomeDC_connect_ldap2(struct libnet_BecomeDC_state *s);
1450|
1451|static void becomeDC_connect_ldap1(struct libnet_BecomeDC_state *s)
1452|{
1453|        struct composite_context *c = s->creq;
1454|
1455|        c->status = becomeDC_ldap_connect(s, &s->ldap1);
1456|        if (!composite_is_ok(c)) return;
1457|
1458|        c->status = becomeDC_ldap1_rootdse(s);
1459|        if (!composite_is_ok(c)) return;
1460|
1461|        c->status = becomeDC_ldap1_crossref_behavior_version(s);
1462|        if (!composite_is_ok(c)) return;
1463|
1464|        c->status = becomeDC_ldap1_domain_behavior_version(s);
1465|        if (!composite_is_ok(c)) return;
1466|
1467|        c->status = becomeDC_ldap1_schema_object_version(s);
1468|        if (!composite_is_ok(c)) return;
1469|
1470|        c->status = becomeDC_ldap1_w2k3_update_revision(s);
1471|        if (!composite_is_ok(c)) return;
1472|
1473|        c->status = becomeDC_ldap1_infrastructure_fsmo(s);
1474|        if (!composite_is_ok(c)) return;
1475|
1476|        c->status = becomeDC_ldap1_rid_manager_fsmo(s);
1477|        if (!composite_is_ok(c)) return;
1478|
1479|        c->status = becomeDC_ldap1_site_object(s);
1480|        if (!composite_is_ok(c)) return;
1481|
1482|        c->status = becomeDC_check_options(s);
1483|        if (!composite_is_ok(c)) return;
1484|
1485|        c->status = becomeDC_ldap1_computer_object(s);
1486|        if (!composite_is_ok(c)) return;
1487|
1488|        c->status = becomeDC_ldap1_server_object_1(s);
1489|        if (!composite_is_ok(c)) return;
1490|
1491|        c->status = becomeDC_ldap1_server_object_2(s);
1492|        if (!composite_is_ok(c)) return;
1493|
1494|        c->status = becomeDC_ldap1_server_object_add(s);
1495|        if (!composite_is_ok(c)) return;
1496|
1497|        c->status = becomeDC_ldap1_server_object_modify(s);
1498|        if (!composite_is_ok(c)) return;
1499|
1500|        becomeDC_drsuapi_connect_send(s, &s->drsuapi1, becomeDC_drsuapi1_connect_recv);
```

```
1501|}
1502|
1503|static void becomeDC_drsuapi_connect_send(struct libnet_BecomeDC_state *s,
1504|                                          struct becomeDC_drsuapi *drsuapi,
1505|                                          void (*recv_fn)(struct composite_context
->    *req))
1506|{
1507|        struct composite_context *c = s->creq;
1508|        struct composite_context *creq;
1509|        char *binding_str;
1510|
1511|        drsuapi->s = s;
1512|
1513|        if (!drsuapi->binding) {
1514|                if (lp_parm_bool(-1, "become_dc", "print", False)) {
1515|                        binding_str = talloc_asprintf(s, "ncacn_ip_tcp:%s[krb5,print,seal
->   ]", s->source_dsa.dns_name);
1516|                        if (composite_nomem(binding_str, c)) return;
1517|                } else {
1518|                        binding_str = talloc_asprintf(s, "ncacn_ip_tcp:%s[krb5,seal]",
->    s->source_dsa.dns_name);
1519|                        if (composite_nomem(binding_str, c)) return;
1520|                }
1521|                c->status = dcerpc_parse_binding(s, binding_str, &drsuapi->binding);
1522|                talloc_free(binding_str);
1523|                if (!composite_is_ok(c)) return;
1524|        }
1525|
1526|        creq = dcerpc_pipe_connect_b_send(s, drsuapi->binding, &dcerpc_table_drsuapi,
1527|                                          s->libnet->cred, s->libnet->event_ctx);
1528|        composite_continue(c, creq, recv_fn, s);
1529|}
1530|
1531|static void becomeDC_drsuapi_bind_send(struct libnet_BecomeDC_state *s,
1532|                                       struct becomeDC_drsuapi *drsuapi,
1533|                                       void (*recv_fn)(struct rpc_request *req));
1534|static void becomeDC_drsuapi1_bind_recv(struct rpc_request *req);
1535|
1536|static void becomeDC_drsuapi1_connect_recv(struct composite_context *req)
1537|{
1538|        struct libnet_BecomeDC_state *s = talloc_get_type(req->async.private_data,
1539|                                          struct libnet_BecomeDC_state);
1540|        struct composite_context *c = s->creq;
1541|
1542|        c->status = dcerpc_pipe_connect_b_recv(req, s, &s->drsuapi1.pipe);
1543|        if (!composite_is_ok(c)) return;
1544|
1545|        c->status = gensec_session_key(s->drsuapi1.pipe->conn->security_state.generic_sta
->   te,
1546|                                       &s->drsuapi1.gensec_skey);
1547|        if (!composite_is_ok(c)) return;
1548|
1549|        becomeDC_drsuapi_bind_send(s, &s->drsuapi1, becomeDC_drsuapi1_bind_recv);
1550|}
1551|
1552|static void becomeDC_drsuapi_bind_send(struct libnet_BecomeDC_state *s,
1553|                                       struct becomeDC_drsuapi *drsuapi,
1554|                                       void (*recv_fn)(struct rpc_request *req))
1555|{
1556|        struct composite_context *c = s->creq;
1557|        struct rpc_request *req;
1558|        struct drsuapi_DsBindInfo28 *bind_info28;
1559|
1560|        GUID_from_string(DRSUAPI_DS_BIND_GUID_W2K3, &drsuapi->bind_guid);
1561|
1562|        bind_info28                             = &drsuapi->local_info28;
1563|        bind_info28->supported_extensions       |= DRSUAPI_SUPPORTED_EXTENSION_BASE;
1564|        bind_info28->supported_extensions       |= DRSUAPI_SUPPORTED_EXTENSION_ASYNC_REPL
->   ICATION;
1565|        bind_info28->supported_extensions       |= DRSUAPI_SUPPORTED_EXTENSION_REMOVEAPI;
->
1566|        bind_info28->supported_extensions       |= DRSUAPI_SUPPORTED_EXTENSION_MOVEREQ_V2
->   ;
1567|        bind_info28->supported_extensions       |= DRSUAPI_SUPPORTED_EXTENSION_GETCHG_COM
->   PRESS;
1568|        bind_info28->supported_extensions       |= DRSUAPI_SUPPORTED_EXTENSION_DCINFO_V1;
->
1569|        bind_info28->supported_extensions       |= DRSUAPI_SUPPORTED_EXTENSION_RESTORE_US
->   N_OPTIMIZATION;
1570|        bind_info28->supported_extensions       |= DRSUAPI_SUPPORTED_EXTENSION_KCC_EXECUT
->   E;
1571|        bind_info28->supported_extensions       |= DRSUAPI_SUPPORTED_EXTENSION_ADDENTRY_V
->   2;
1572|        if (s->domain.behavior_version == 2) {
1573|                /* TODO: find out how this is really triggered! */
1574|                bind_info28->supported_extensions       |=
->    DRSUAPI_SUPPORTED_EXTENSION_LINKED_VALUE_REPLICATION;
```

```
1575|        }
1576|        bind_info28->supported_extensions       |= DRSUAPI_SUPPORTED_EXTENSION_DCINFO_V2;
->
1577|        bind_info28->supported_extensions       |= DRSUAPI_SUPPORTED_EXTENSION_INSTANCE_T
->   YPE_NOT_REQ_ON_MOD;
1578|        bind_info28->supported_extensions       |= DRSUAPI_SUPPORTED_EXTENSION_CRYPTO_BIN
->   D;
1579|        bind_info28->supported_extensions       |= DRSUAPI_SUPPORTED_EXTENSION_GET_REPL_I
->   NFO;
1580|        bind_info28->supported_extensions       |= DRSUAPI_SUPPORTED_EXTENSION_STRONG_ENC
->   RYPTION;
1581|        bind_info28->supported_extensions       |= DRSUAPI_SUPPORTED_EXTENSION_DCINFO_V01
->   ;
1582|        bind_info28->supported_extensions       |= DRSUAPI_SUPPORTED_EXTENSION_TRANSITIVE
->   _MEMBERSHIP;
1583|        bind_info28->supported_extensions       |= DRSUAPI_SUPPORTED_EXTENSION_ADD_SID_HI
->   STORY;
1584|        bind_info28->supported_extensions       |= DRSUAPI_SUPPORTED_EXTENSION_POST_BETA3
->   ;
1585|        bind_info28->supported_extensions       |= DRSUAPI_SUPPORTED_EXTENSION_00100000;
1586|        bind_info28->supported_extensions       |= DRSUAPI_SUPPORTED_EXTENSION_GET_MEMBER
->   SHIPS2;
1587|        bind_info28->supported_extensions       |= DRSUAPI_SUPPORTED_EXTENSION_GETCHGREQ_
->   V6;
1588|        bind_info28->supported_extensions       |= DRSUAPI_SUPPORTED_EXTENSION_NONDOMAIN_
->   NCS;
1589|        bind_info28->supported_extensions       |= DRSUAPI_SUPPORTED_EXTENSION_GETCHGREQ_
->   V8;
1590|        bind_info28->supported_extensions       |= DRSUAPI_SUPPORTED_EXTENSION_GETCHGREPL
->   Y_V5;
1591|        bind_info28->supported_extensions       |= DRSUAPI_SUPPORTED_EXTENSION_GETCHGREPL
->   Y_V6;
1592|        bind_info28->supported_extensions       |= DRSUAPI_SUPPORTED_EXTENSION_ADDENTRYRE
->   PLY_V3;
1593|        bind_info28->supported_extensions       |= DRSUAPI_SUPPORTED_EXTENSION_GETCHGREPL
->   Y_V7;
1594|        bind_info28->supported_extensions       |= DRSUAPI_SUPPORTED_EXTENSION_VERIFY_OBJ
->   ECT;
1595|#if 0 /* we don't support XPRESS compression yet */
1596|        bind_info28->supported_extensions       |= DRSUAPI_SUPPORTED_EXTENSION_XPRESS_COM
->   PRESS;
1597|#endif
1598|        bind_info28->site_guid                  = s->dest_dsa.site_guid;
1599|        if (s->domain.behavior_version == 2) {
1600|                /* TODO: find out how this is really triggered! */
1601|                bind_info28->u1                         = 528;
1602|        } else {
1603|                bind_info28->u1                         = 516;
1604|        }
1605|        bind_info28->repl_epoch                = 0;
1606|
1607|        drsuapi->bind_info_ctr.length          = 28;
1608|        drsuapi->bind_info_ctr.info.info28     = *bind_info28;
1609|
1610|        drsuapi->bind_r.in.bind_guid = &drsuapi->bind_guid;
1611|        drsuapi->bind_r.in.bind_info = &drsuapi->bind_info_ctr;
1612|        drsuapi->bind_r.out.bind_handle = &drsuapi->bind_handle;
1613|
1614|        req = dcerpc_drsuapi_DsBind_send(drsuapi->pipe, s, &drsuapi->bind_r);
1615|        composite_continue_rpc(c, req, recv_fn, s);
1616|}
1617|
1618|static WERROR becomeDC_drsuapi_bind_recv(struct libnet_BecomeDC_state *s,
1619|                                         struct becomeDC_drsuapi *drsuapi)
1620|{
1621|        if (!W_ERROR_IS_OK(drsuapi->bind_r.out.result)) {
1622|                return drsuapi->bind_r.out.result;
1623|        }
1624|
1625|        ZERO_STRUCT(drsuapi->remote_info28);
1626|        if (drsuapi->bind_r.out.bind_info) {
1627|                switch (drsuapi->bind_r.out.bind_info->length) {
1628|                case 24: {
1629|                        struct drsuapi_DsBindInfo24 *info24;
1630|                        info24 = &drsuapi->bind_r.out.bind_info->info.info24;
1631|                        drsuapi->remote_info28.supported_extensions     =
->    info24->supported_extensions;
1632|                        drsuapi->remote_info28.site_guid                =
->    info24->site_guid;
1633|                        drsuapi->remote_info28.u1                       = info24->u1;
1634|                        drsuapi->remote_info28.repl_epoch               = 0;
1635|                        break;
1636|                }
1637|                case 28:
1638|                        drsuapi->remote_info28 = drsuapi->bind_r.out.bind_info->info.info
->   28;
1639|                        break;
```

```
1640|                }
1641|        }
1642|
1643|        return WERR_OK;
1644|}
1645|
1646|static void becomeDC_drsuapi1_add_entry_send(struct libnet_BecomeDC_state *s);
1647|
1648|static void becomeDC_drsuapi1_bind_recv(struct rpc_request *req)
1649|{
1650|        struct libnet_BecomeDC_state *s = talloc_get_type(req->async.private,
1651|                                                        struct libnet_BecomeDC_state);
1652|        struct composite_context *c = s->creq;
1653|        WERROR status;
1654|
1655|        bool print = false;
1656|
1657|        if (req->p->conn->flags & DCERPC_DEBUG_PRINT_OUT) {
1658|                print = true;
1659|        }
1660|
1661|        c->status = dcerpc_ndr_request_recv(req);
1662|        if (!composite_is_ok(c)) return;
1663|
1664|        if (print) {
1665|                NDR_PRINT_OUT_DEBUG(drsuapi_DsBind, &s->drsuapi1.bind_r);
1666|        }
1667|
1668|        status = becomeDC_drsuapi_bind_recv(s, &s->drsuapi1);
1669|        if (!W_ERROR_IS_OK(status)) {
1670|                composite_error(c, werror_to_ntstatus(status));
1671|                return;
1672|        }
1673|
1674|        becomeDC_drsuapi1_add_entry_send(s);
1675|}
1676|
1677|static void becomeDC_drsuapi1_add_entry_recv(struct rpc_request *req);
1678|
1679|static void becomeDC_drsuapi1_add_entry_send(struct libnet_BecomeDC_state *s)
1680|{
1681|        struct composite_context *c = s->creq;
1682|        struct rpc_request *req;
1683|        struct drsuapi_DsAddEntry *r;
1684|        struct drsuapi_DsReplicaObjectIdentifier *identifier;
1685|        uint32_t num_attrs, i = 0;
1686|        struct drsuapi_DsReplicaAttribute *attrs;
1687|        bool w2k3;
1688|
1689|        /* choose a random invocationId */
1690|        s->dest_dsa.invocation_id = GUID_random();
1691|
1692|        /*
1693|         * if the schema version indicates w2k3, then
1694|         * also send some w2k3 specific attributes
1695|         */
1696|        if (s->forest.schema_object_version >= 30) {
1697|                w2k3 = true;
1698|        } else {
1699|                w2k3 = false;
1700|        }
1701|
1702|        r = talloc_zero(s, struct drsuapi_DsAddEntry);
1703|        if (composite_nomem(r, c)) return;
1704|
1705|        /* setup identifier */
1706|        identifier              = talloc(r, struct drsuapi_DsReplicaObjectIdentifier);
1707|        if (composite_nomem(identifier, c)) return;
1708|        identifier->guid        = GUID_zero();
1709|        identifier->sid         = s->zero_sid;
1710|        identifier->dn          = talloc_asprintf(identifier, "CN=NTDS Settings,%s",
1711|                                                  s->dest_dsa.server_dn_str);
1712|        if (composite_nomem(identifier->dn, c)) return;
1713|
1714|        /* allocate attribute array */
1715|        num_attrs       = 11;
1716|        attrs           = talloc_array(r, struct drsuapi_DsReplicaAttribute, num_attrs);
1717|        if (composite_nomem(attrs, c)) return;
1718|
1719|        /* ntSecurityDescriptor */
1720|        {
1721|                struct drsuapi_DsAttributeValue *vs;
1722|                DATA_BLOB *vd;
1723|                struct security_descriptor *v;
1724|                struct dom_sid *domain_admins_sid;
1725|                const char *domain_admins_sid_str;
1726|
```

```
1727|                vs = talloc_array(attrs, struct drsuapi_DsAttributeValue, 1);
1728|                if (composite_nomem(vs, c)) return;
1729|
1730|                vd = talloc_array(vs, DATA_BLOB, 1);
1731|                if (composite_nomem(vd, c)) return;
1732|
1733|                domain_admins_sid = dom_sid_add_rid(vs, s->domain.sid,
->    DOMAIN_RID_ADMINS);
1734|                if (composite_nomem(domain_admins_sid, c)) return;
1735|
1736|                domain_admins_sid_str = dom_sid_string(domain_admins_sid,
->    domain_admins_sid);
1737|                if (composite_nomem(domain_admins_sid_str, c)) return;
1738|
1739|                v = security_descriptor_create(vd,
1740|                                               /* owner: domain admins */
1741|                                               domain_admins_sid_str,
1742|                                               /* owner group: domain admins */
1743|                                               domain_admins_sid_str,
1744|                                               /* authenticated users */
1745|                                               SID_NT_AUTHENTICATED_USERS,
1746|                                               SEC_ACE_TYPE_ACCESS_ALLOWED,
1747|                                               SEC_STD_READ_CONTROL |
1748|                                               SEC_ADS_LIST |
1749|                                               SEC_ADS_READ_PROP |
1750|                                               SEC_ADS_LIST_OBJECT,
1751|                                               0,
1752|                                               /* domain admins */
1753|                                               domain_admins_sid_str,
1754|                                               SEC_ACE_TYPE_ACCESS_ALLOWED,
1755|                                               SEC_STD_REQUIRED |
1756|                                               SEC_ADS_CREATE_CHILD |
1757|                                               SEC_ADS_LIST |
1758|                                               SEC_ADS_SELF_WRITE |
1759|                                               SEC_ADS_READ_PROP |
1760|                                               SEC_ADS_WRITE_PROP |
1761|                                               SEC_ADS_DELETE_TREE |
1762|                                               SEC_ADS_LIST_OBJECT |
1763|                                               SEC_ADS_CONTROL_ACCESS,
1764|                                               0,
1765|                                               /* system */
1766|                                               SID_NT_SYSTEM,
1767|                                               SEC_ACE_TYPE_ACCESS_ALLOWED,
1768|                                               SEC_STD_REQUIRED |
1769|                                               SEC_ADS_CREATE_CHILD |
1770|                                               SEC_ADS_DELETE_CHILD |
1771|                                               SEC_ADS_LIST |
1772|                                               SEC_ADS_SELF_WRITE |
1773|                                               SEC_ADS_READ_PROP |
1774|                                               SEC_ADS_WRITE_PROP |
1775|                                               SEC_ADS_DELETE_TREE |
1776|                                               SEC_ADS_LIST_OBJECT |
1777|                                               SEC_ADS_CONTROL_ACCESS,
1778|                                               0,
1779|                                               /* end */
1780|                                               NULL);
1781|                if (composite_nomem(v, c)) return;
1782|
1783|                c->status = ndr_push_struct_blob(&vd[0], vd,
->    v,(ndr_push_flags_fn_t)ndr_push_security_descriptor);
1784|                if (!composite_is_ok(c)) return;
1785|
1786|                vs[0].blob              = &vd[0];
1787|
1788|                attrs[i].attid                  = DRSUAPI_ATTRIBUTE_ntSecurityDescriptor;
->
1789|                attrs[i].value_ctr.num_values   = 1;
1790|                attrs[i].value_ctr.values       = vs;
1791|
1792|                i++;
1793|        }
1794|
1795|        /* objectClass: nTDSDSA */
1796|        {
1797|                struct drsuapi_DsAttributeValue *vs;
1798|                DATA_BLOB *vd;
1799|
1800|                vs = talloc_array(attrs, struct drsuapi_DsAttributeValue, 1);
1801|                if (composite_nomem(vs, c)) return;
1802|
1803|                vd = talloc_array(vs, DATA_BLOB, 1);
1804|                if (composite_nomem(vd, c)) return;
1805|
1806|                vd[0] = data_blob_talloc(vd, NULL, 4);
1807|                if (composite_nomem(vd[0].data, c)) return;
1808|
1809|                /* value for nTDSDSA */
```

```
1810|                SIVAL(vd[0].data, 0, 0x0017002F);
1811|
1812|                vs[0].blob              = &vd[0];
1813|
1814|                attrs[i].attid                  = DRSUAPI_ATTRIBUTE_objectClass;
1815|                attrs[i].value_ctr.num_values   = 1;
1816|                attrs[i].value_ctr.values       = vs;
1817|
1818|                i++;
1819|        }
1820|
1821|        /* objectCategory: CN=NTDS-DSA,CN=Schema,... */
1822|        {
1823|                struct drsuapi_DsAttributeValue *vs;
1824|                DATA_BLOB *vd;
1825|                struct drsuapi_DsReplicaObjectIdentifier3 v[1];
1826|
1827|                vs = talloc_array(attrs, struct drsuapi_DsAttributeValue, 1);
1828|                if (composite_nomem(vs, c)) return;
1829|
1830|                vd = talloc_array(vs, DATA_BLOB, 1);
1831|                if (composite_nomem(vd, c)) return;
1832|
1833|                v[0].guid               = GUID_zero();
1834|                v[0].sid                = s->zero_sid;
1835|                v[0].dn                 = talloc_asprintf(vd, "CN=NTDS-DSA,%s",
1836|                                                          s->forest.schema_dn_str);
1837|                if (composite_nomem(v[0].dn, c)) return;
1838|
1839|                c->status = ndr_push_struct_blob(&vd[0], vd, &v[0],
1840|                                                 (ndr_push_flags_fn_t)ndr_push_drsuapi_Ds
->   ReplicaObjectIdentifier3);
1841|                if (!composite_is_ok(c)) return;
1842|
1843|                vs[0].blob              = &vd[0];
1844|
1845|                attrs[i].attid                  = DRSUAPI_ATTRIBUTE_objectCategory;
1846|                attrs[i].value_ctr.num_values   = 1;
1847|                attrs[i].value_ctr.values       = vs;
1848|
1849|                i++;
1850|        }
1851|
1852|        /* invocationId: random guid */
1853|        {
1854|                struct drsuapi_DsAttributeValue *vs;
1855|                DATA_BLOB *vd;
1856|                const struct GUID *v;
1857|
1858|                vs = talloc_array(attrs, struct drsuapi_DsAttributeValue, 1);
1859|                if (composite_nomem(vs, c)) return;
1860|
1861|                vd = talloc_array(vs, DATA_BLOB, 1);
1862|                if (composite_nomem(vd, c)) return;
1863|
1864|                v = &s->dest_dsa.invocation_id;
1865|
1866|                c->status = ndr_push_struct_blob(&vd[0], vd, v,
->    (ndr_push_flags_fn_t)ndr_push_GUID);
1867|                if (!composite_is_ok(c)) return;
1868|
1869|                vs[0].blob              = &vd[0];
1870|
1871|                attrs[i].attid                  = DRSUAPI_ATTRIBUTE_invocationId;
1872|                attrs[i].value_ctr.num_values   = 1;
1873|                attrs[i].value_ctr.values       = vs;
1874|
1875|                i++;
1876|        }
1877|
1878|        /* hasMasterNCs: ... */
1879|        {
1880|                struct drsuapi_DsAttributeValue *vs;
1881|                DATA_BLOB *vd;
1882|                struct drsuapi_DsReplicaObjectIdentifier3 v[3];
1883|
1884|                vs = talloc_array(attrs, struct drsuapi_DsAttributeValue, 3);
1885|                if (composite_nomem(vs, c)) return;
1886|
1887|                vd = talloc_array(vs, DATA_BLOB, 3);
1888|                if (composite_nomem(vd, c)) return;
1889|
1890|                v[0].guid               = GUID_zero();
1891|                v[0].sid                = s->zero_sid;
1892|                v[0].dn                 = s->forest.config_dn_str;
1893|
1894|                v[1].guid               = GUID_zero();
```

```
1895|                v[1].sid                = s->zero_sid;
1896|                v[1].dn                 = s->domain.dn_str;
1897|
1898|                v[2].guid               = GUID_zero();
1899|                v[2].sid                = s->zero_sid;
1900|                v[2].dn                 = s->forest.schema_dn_str;
1901|
1902|                c->status = ndr_push_struct_blob(&vd[0], vd, &v[0],
1903|                                                 (ndr_push_flags_fn_t)ndr_push_drsuapi_Ds
->   ReplicaObjectIdentifier3);
1904|                if (!composite_is_ok(c)) return;
1905|
1906|                c->status = ndr_push_struct_blob(&vd[1], vd, &v[1],
1907|                                                 (ndr_push_flags_fn_t)ndr_push_drsuapi_Ds
->   ReplicaObjectIdentifier3);
1908|                if (!composite_is_ok(c)) return;
1909|
1910|                c->status = ndr_push_struct_blob(&vd[2], vd, &v[2],
1911|                                                 (ndr_push_flags_fn_t)ndr_push_drsuapi_Ds
->   ReplicaObjectIdentifier3);
1912|                if (!composite_is_ok(c)) return;
1913|
1914|                vs[0].blob              = &vd[0];
1915|                vs[1].blob              = &vd[1];
1916|                vs[2].blob              = &vd[2];
1917|
1918|                attrs[i].attid                  = DRSUAPI_ATTRIBUTE_hasMasterNCs;
1919|                attrs[i].value_ctr.num_values   = 3;
1920|                attrs[i].value_ctr.values       = vs;
1921|
1922|                i++;
1923|        }
1924|
1925|        /* msDS-hasMasterNCs: ... */
1926|        if (w2k3) {
1927|                struct drsuapi_DsAttributeValue *vs;
1928|                DATA_BLOB *vd;
1929|                struct drsuapi_DsReplicaObjectIdentifier3 v[3];
1930|
1931|                vs = talloc_array(attrs, struct drsuapi_DsAttributeValue, 3);
1932|                if (composite_nomem(vs, c)) return;
1933|
1934|                vd = talloc_array(vs, DATA_BLOB, 3);
1935|                if (composite_nomem(vd, c)) return;
1936|
1937|                v[0].guid               = GUID_zero();
1938|                v[0].sid                = s->zero_sid;
1939|                v[0].dn                 = s->forest.config_dn_str;
1940|
1941|                v[1].guid               = GUID_zero();
1942|                v[1].sid                = s->zero_sid;
1943|                v[1].dn                 = s->domain.dn_str;
1944|
1945|                v[2].guid               = GUID_zero();
1946|                v[2].sid                = s->zero_sid;
1947|                v[2].dn                 = s->forest.schema_dn_str;
1948|
1949|                c->status = ndr_push_struct_blob(&vd[0], vd, &v[0],
1950|                                                 (ndr_push_flags_fn_t)ndr_push_drsuapi_Ds
->   ReplicaObjectIdentifier3);
1951|                if (!composite_is_ok(c)) return;
1952|
1953|                c->status = ndr_push_struct_blob(&vd[1], vd, &v[1],
1954|                                                 (ndr_push_flags_fn_t)ndr_push_drsuapi_Ds
->   ReplicaObjectIdentifier3);
1955|                if (!composite_is_ok(c)) return;
1956|
1957|                c->status = ndr_push_struct_blob(&vd[2], vd, &v[2],
1958|                                                 (ndr_push_flags_fn_t)ndr_push_drsuapi_Ds
->   ReplicaObjectIdentifier3);
1959|                if (!composite_is_ok(c)) return;
1960|
1961|                vs[0].blob              = &vd[0];
1962|                vs[1].blob              = &vd[1];
1963|                vs[2].blob              = &vd[2];
1964|
1965|                attrs[i].attid                  = DRSUAPI_ATTRIBUTE_msDS_hasMasterNCs;
1966|                attrs[i].value_ctr.num_values   = 3;
1967|                attrs[i].value_ctr.values       = vs;
1968|
1969|                i++;
1970|        }
1971|
1972|        /* dMDLocation: CN=Schema,... */
1973|        {
1974|                struct drsuapi_DsAttributeValue *vs;
1975|                DATA_BLOB *vd;
```

```
1976|                struct drsuapi_DsReplicaObjectIdentifier3 v[1];
1977|
1978|                vs = talloc_array(attrs, struct drsuapi_DsAttributeValue, 1);
1979|                if (composite_nomem(vs, c)) return;
1980|
1981|                vd = talloc_array(vs, DATA_BLOB, 1);
1982|                if (composite_nomem(vd, c)) return;
1983|
1984|                v[0].guid               = GUID_zero();
1985|                v[0].sid                = s->zero_sid;
1986|                v[0].dn                 = s->forest.schema_dn_str;
1987|
1988|                c->status = ndr_push_struct_blob(&vd[0], vd, &v[0],
1989|                                                 (ndr_push_flags_fn_t)ndr_push_drsuapi_Ds
->   ReplicaObjectIdentifier3);
1990|                if (!composite_is_ok(c)) return;
1991|
1992|                vs[0].blob              = &vd[0];
1993|
1994|                attrs[i].attid                  = DRSUAPI_ATTRIBUTE_dMDLocation;
1995|                attrs[i].value_ctr.num_values   = 1;
1996|                attrs[i].value_ctr.values       = vs;
1997|
1998|                i++;
1999|        }
2000|
2001|        /* msDS-HasDomainNCs: <domain_partition> */
2002|        if (w2k3) {
2003|                struct drsuapi_DsAttributeValue *vs;
2004|                DATA_BLOB *vd;
2005|                struct drsuapi_DsReplicaObjectIdentifier3 v[1];
2006|
2007|                vs = talloc_array(attrs, struct drsuapi_DsAttributeValue, 1);
2008|                if (composite_nomem(vs, c)) return;
2009|
2010|                vd = talloc_array(vs, DATA_BLOB, 1);
2011|                if (composite_nomem(vd, c)) return;
2012|
2013|                v[0].guid               = GUID_zero();
2014|                v[0].sid                = s->zero_sid;
2015|                v[0].dn                 = s->domain.dn_str;
2016|
2017|                c->status = ndr_push_struct_blob(&vd[0], vd, &v[0],
2018|                                                 (ndr_push_flags_fn_t)ndr_push_drsuapi_Ds
->   ReplicaObjectIdentifier3);
2019|                if (!composite_is_ok(c)) return;
2020|
2021|                vs[0].blob              = &vd[0];
2022|
2023|                attrs[i].attid                  = DRSUAPI_ATTRIBUTE_msDS_HasDomainNCs;
2024|                attrs[i].value_ctr.num_values   = 1;
2025|                attrs[i].value_ctr.values       = vs;
2026|
2027|                i++;
2028|        }
2029|
2030|        /* msDS-Behavior-Version */
2031|        if (w2k3) {
2032|                struct drsuapi_DsAttributeValue *vs;
2033|                DATA_BLOB *vd;
2034|
2035|                vs = talloc_array(attrs, struct drsuapi_DsAttributeValue, 1);
2036|                if (composite_nomem(vs, c)) return;
2037|
2038|                vd = talloc_array(vs, DATA_BLOB, 1);
2039|                if (composite_nomem(vd, c)) return;
2040|
2041|                vd[0] = data_blob_talloc(vd, NULL, 4);
2042|                if (composite_nomem(vd[0].data, c)) return;
2043|
2044|                SIVAL(vd[0].data, 0, DS_BEHAVIOR_WIN2003);
2045|
2046|                vs[0].blob              = &vd[0];
2047|
2048|                attrs[i].attid                  = DRSUAPI_ATTRIBUTE_msDS_Behavior_Version
->   ;
2049|                attrs[i].value_ctr.num_values   = 1;
2050|                attrs[i].value_ctr.values       = vs;
2051|
2052|                i++;
2053|        }
2054|
2055|        /* systemFlags */
2056|        {
2057|                struct drsuapi_DsAttributeValue *vs;
2058|                DATA_BLOB *vd;
2059|
```

```
2060|                vs = talloc_array(attrs, struct drsuapi_DsAttributeValue, 1);
2061|                if (composite_nomem(vs, c)) return;
2062|
2063|                vd = talloc_array(vs, DATA_BLOB, 1);
2064|                if (composite_nomem(vd, c)) return;
2065|
2066|                vd[0] = data_blob_talloc(vd, NULL, 4);
2067|                if (composite_nomem(vd[0].data, c)) return;
2068|
2069|                SIVAL(vd[0].data, 0, SYSTEM_FLAG_DISALLOW_MOVE_ON_DELETE);
2070|
2071|                vs[0].blob              = &vd[0];
2072|
2073|                attrs[i].attid                  = DRSUAPI_ATTRIBUTE_systemFlags;
2074|                attrs[i].value_ctr.num_values   = 1;
2075|                attrs[i].value_ctr.values       = vs;
2076|
2077|                i++;
2078|        }
2079|
2080|        /* serverReference: ... */
2081|        {
2082|                struct drsuapi_DsAttributeValue *vs;
2083|                DATA_BLOB *vd;
2084|                struct drsuapi_DsReplicaObjectIdentifier3 v[1];
2085|
2086|                vs = talloc_array(attrs, struct drsuapi_DsAttributeValue, 1);
2087|                if (composite_nomem(vs, c)) return;
2088|
2089|                vd = talloc_array(vs, DATA_BLOB, 1);
2090|                if (composite_nomem(vd, c)) return;
2091|
2092|                v[0].guid               = GUID_zero();
2093|                v[0].sid                = s->zero_sid;
2094|                v[0].dn                 = s->dest_dsa.computer_dn_str;
2095|
2096|                c->status = ndr_push_struct_blob(&vd[0], vd, &v[0],
2097|                                                 (ndr_push_flags_fn_t)ndr_push_drsuapi_Ds
->   ReplicaObjectIdentifier3);
2098|                if (!composite_is_ok(c)) return;
2099|
2100|                vs[0].blob              = &vd[0];
2101|
2102|                attrs[i].attid                  = DRSUAPI_ATTRIBUTE_serverReference;
2103|                attrs[i].value_ctr.num_values   = 1;
2104|                attrs[i].value_ctr.values       = vs;
2105|
2106|                i++;
2107|        }
2108|
2109|        /* truncate the attribute list to the attribute count we have filled in */
2110|        num_attrs = i;
2111|
2112|        /* setup request structure */
2113|        r->in.bind_handle                                                  =
->    &s->drsuapi1.bind_handle;
2114|        r->in.level                                                       = 2;
2115|        r->in.req.req2.first_object.next_object                           = NULL;
2116|        r->in.req.req2.first_object.object.identifier                     = identifier;
2117|        r->in.req.req2.first_object.object.unknown1                       = 0x00000000;
2118|        r->in.req.req2.first_object.object.attribute_ctr.num_attributes = num_attrs;
2119|        r->in.req.req2.first_object.object.attribute_ctr.attributes     = attrs;
2120|
2121|        req = dcerpc_drsuapi_DsAddEntry_send(s->drsuapi1.pipe, r, r);
2122|        composite_continue_rpc(c, req, becomeDC_drsuapi1_add_entry_recv, s);
2123|}
2124|
2125|static void becomeDC_drsuapi2_connect_recv(struct composite_context *req);
2126|static NTSTATUS becomeDC_prepare_db(struct libnet_BecomeDC_state *s);
2127|
2128|static void becomeDC_drsuapi1_add_entry_recv(struct rpc_request *req)
2129|{
2130|        struct libnet_BecomeDC_state *s = talloc_get_type(req->async.private,
2131|                                           struct libnet_BecomeDC_state);
2132|        struct composite_context *c = s->creq;
2133|        struct drsuapi_DsAddEntry *r = talloc_get_type(req->ndr.struct_ptr,
2134|                                         struct drsuapi_DsAddEntry);
2135|        char *binding_str;
2136|        bool print = false;
2137|
2138|        if (req->p->conn->flags & DCERPC_DEBUG_PRINT_OUT) {
2139|                print = true;
2140|        }
2141|
2142|        c->status = dcerpc_ndr_request_recv(req);
2143|        if (!composite_is_ok(c)) return;
2144|
```

```
2145|        if (print) {
2146|                NDR_PRINT_OUT_DEBUG(drsuapi_DsAddEntry, r);
2147|        }
2148|
2149|        if (!W_ERROR_IS_OK(r->out.result)) {
2150|                composite_error(c, werror_to_ntstatus(r->out.result));
2151|                return;
2152|        }
2153|
2154|        if (r->out.level == 3) {
2155|                if (r->out.ctr.ctr3.count != 1) {
2156|                        WERROR status;
2157|
2158|                        if (r->out.ctr.ctr3.level != 1) {
2159|                                composite_error(c, NT_STATUS_INVALID_NETWORK_RESPONSE);
2160|                                return;
2161|                        }
2162|
2163|                        if (!r->out.ctr.ctr3.error) {
2164|                                composite_error(c, NT_STATUS_INVALID_NETWORK_RESPONSE);
2165|                                return;
2166|                        }
2167|
2168|                        status = r->out.ctr.ctr3.error->info1.status;
2169|
2170|                        if (!r->out.ctr.ctr3.error->info1.info) {
2171|                                composite_error(c, werror_to_ntstatus(status));
2172|                                return;
2173|                        }
2174|
2175|                        /* see if we can get a more detailed error */
2176|                        switch (r->out.ctr.ctr3.error->info1.level) {
2177|                        case 1:
2178|                                status = r->out.ctr.ctr3.error->info1.info->error1.status
->   ;
2179|                                break;
2180|                        case 4:
2181|                        case 5:
2182|                        case 6:
2183|                        case 7:
2184|                                status = r->out.ctr.ctr3.error->info1.info->errorX.status
->   ;
2185|                                break;
2186|                        }
2187|
2188|                        composite_error(c, werror_to_ntstatus(status));
2189|                        return;
2190|                }
2191|
2192|                s->dest_dsa.ntds_guid   = r->out.ctr.ctr3.objects[0].guid;
2193|        } else if (r->out.level == 2) {
2194|                if (r->out.ctr.ctr2.count != 1) {
2195|                        composite_error(c, werror_to_ntstatus(r->out.ctr.ctr2.error.statu
->   s));
2196|                        return;
2197|                }
2198|
2199|                s->dest_dsa.ntds_guid   = r->out.ctr.ctr2.objects[0].guid;
2200|        } else {
2201|                composite_error(c, NT_STATUS_INVALID_NETWORK_RESPONSE);
2202|                return;
2203|        }
2204|
2205|        talloc_free(r);
2206|
2207|        s->dest_dsa.ntds_dn_str = talloc_asprintf(s, "CN=NTDS Settings,%s",
2208|                                                    s->dest_dsa.server_dn_str);
2209|        if (composite_nomem(s->dest_dsa.ntds_dn_str, c)) return;
2210|
2211|        c->status = becomeDC_prepare_db(s);
2212|        if (!composite_is_ok(c)) return;
2213|
2214|        /* this avoids the epmapper lookup on the 2nd connection */
2215|        binding_str = dcerpc_binding_string(s, s->drsuapi1.binding);
2216|        if (composite_nomem(binding_str, c)) return;
2217|
2218|        c->status = dcerpc_parse_binding(s, binding_str, &s->drsuapi2.binding);
2219|        talloc_free(binding_str);
2220|        if (!composite_is_ok(c)) return;
2221|
2222|        /* w2k3 uses the same assoc_group_id as on the first connection, so we do */
2223|        s->drsuapi2.binding->assoc_group_id     = s->drsuapi1.pipe->assoc_group_id;
2224|
2225|        becomeDC_drsuapi_connect_send(s, &s->drsuapi2, becomeDC_drsuapi2_connect_recv);
2226|}
2227|
2228|static NTSTATUS becomeDC_prepare_db(struct libnet_BecomeDC_state *s)
```

```
2229|{
2230|        if (!s->callbacks.prepare_db) return NT_STATUS_OK;
2231|
2232|        s->_pp.domain           = &s->domain;
2233|        s->_pp.forest           = &s->forest;
2234|        s->_pp.source_dsa       = &s->source_dsa;
2235|        s->_pp.dest_dsa         = &s->dest_dsa;
2236|
2237|        return s->callbacks.prepare_db(s->callbacks.private_data, &s->_pp);
2238|}
2239|
2240|static void becomeDC_drsuapi2_bind_recv(struct rpc_request *req);
2241|
2242|static void becomeDC_drsuapi2_connect_recv(struct composite_context *req)
2243|{
2244|        struct libnet_BecomeDC_state *s = talloc_get_type(req->async.private_data,
2245|                                             struct libnet_BecomeDC_state);
2246|        struct composite_context *c = s->creq;
2247|
2248|        c->status = dcerpc_pipe_connect_b_recv(req, s, &s->drsuapi2.pipe);
2249|        if (!composite_is_ok(c)) return;
2250|
2251|        c->status = gensec_session_key(s->drsuapi2.pipe->conn->security_state.generic_sta
->   te,
2252|                                        &s->drsuapi2.gensec_skey);
2253|        if (!composite_is_ok(c)) return;
2254|
2255|        becomeDC_drsuapi_bind_send(s, &s->drsuapi2, becomeDC_drsuapi2_bind_recv);
2256|}
2257|
2258|static void becomeDC_drsuapi3_connect_recv(struct composite_context *req);
2259|
2260|static void becomeDC_drsuapi2_bind_recv(struct rpc_request *req)
2261|{
2262|        struct libnet_BecomeDC_state *s = talloc_get_type(req->async.private,
2263|                                             struct libnet_BecomeDC_state);
2264|        struct composite_context *c = s->creq;
2265|        char *binding_str;
2266|        WERROR status;
2267|
2268|        bool print = false;
2269|
2270|        if (req->p->conn->flags & DCERPC_DEBUG_PRINT_OUT) {
2271|                print = true;
2272|        }
2273|
2274|        c->status = dcerpc_ndr_request_recv(req);
2275|        if (!composite_is_ok(c)) return;
2276|
2277|        if (print) {
2278|                NDR_PRINT_OUT_DEBUG(drsuapi_DsBind, &s->drsuapi2.bind_r);
2279|        }
2280|
2281|        status = becomeDC_drsuapi_bind_recv(s, &s->drsuapi2);
2282|        if (!W_ERROR_IS_OK(status)) {
2283|                composite_error(c, werror_to_ntstatus(status));
2284|                return;
2285|        }
2286|
2287|        /* this avoids the epmapper lookup on the 3rd connection */
2288|        binding_str = dcerpc_binding_string(s, s->drsuapi1.binding);
2289|        if (composite_nomem(binding_str, c)) return;
2290|
2291|        c->status = dcerpc_parse_binding(s, binding_str, &s->drsuapi3.binding);
2292|        talloc_free(binding_str);
2293|        if (!composite_is_ok(c)) return;
2294|
2295|        /* w2k3 uses the same assoc_group_id as on the first connection, so we do */
2296|        s->drsuapi3.binding->assoc_group_id     = s->drsuapi1.pipe->assoc_group_id;
2297|        /* w2k3 uses the concurrent multiplex feature on the 3rd connection, so we do */
2298|        s->drsuapi3.binding->flags              |= DCERPC_CONCURRENT_MULTIPLEX;
2299|
2300|        becomeDC_drsuapi_connect_send(s, &s->drsuapi3, becomeDC_drsuapi3_connect_recv);
2301|}
2302|
2303|static void becomeDC_drsuapi3_pull_schema_send(struct libnet_BecomeDC_state *s);
2304|
2305|static void becomeDC_drsuapi3_connect_recv(struct composite_context *req)
2306|{
2307|        struct libnet_BecomeDC_state *s = talloc_get_type(req->async.private_data,
2308|                                             struct libnet_BecomeDC_state);
2309|        struct composite_context *c = s->creq;
2310|
2311|        c->status = dcerpc_pipe_connect_b_recv(req, s, &s->drsuapi3.pipe);
2312|        if (!composite_is_ok(c)) return;
2313|
2314|        c->status = gensec_session_key(s->drsuapi3.pipe->conn->security_state.generic_sta
```

```
->   te,
2315|                                       &s->drsuapi3.gensec_skey);
2316|        if (!composite_is_ok(c)) return;
2317|
2318|        becomeDC_drsuapi3_pull_schema_send(s);
2319|}
2320|
2321|static void becomeDC_drsuapi_pull_partition_send(struct libnet_BecomeDC_state *s,
2322|                                                 struct becomeDC_drsuapi *drsuapi_h,
2323|                                                 struct becomeDC_drsuapi *drsuapi_p,
2324|                                                 struct libnet_BecomeDC_Partition
->    *partition,
2325|                                                 void (*recv_fn)(struct rpc_request
->    *req))
2326|{
2327|        struct composite_context *c = s->creq;
2328|        struct rpc_request *req;
2329|        struct drsuapi_DsGetNCChanges *r;
2330|
2331|        r = talloc(s, struct drsuapi_DsGetNCChanges);
2332|        if (composite_nomem(r, c)) return;
2333|
2334|        r->in.level = talloc(r, int32_t);
2335|        if (composite_nomem(r->in.level, c)) return;
2336|        r->out.level = talloc(r, int32_t);
2337|        if (composite_nomem(r->out.level, c)) return;
2338|
2339|        r->in.bind_handle       = &drsuapi_h->bind_handle;
2340|        if (drsuapi_h->remote_info28.supported_extensions &
->    DRSUAPI_SUPPORTED_EXTENSION_GETCHGREQ_V8) {
2341|                *r->in.level                              = 8;
2342|                r->in.req.req8.destination_dsa_guid     = partition->destination_dsa_guid
->   ;
2343|                r->in.req.req8.source_dsa_invocation_id = partition->source_dsa_invocatio
->   n_id;
2344|                r->in.req.req8.naming_context           = &partition->nc;
2345|                r->in.req.req8.highwatermark            = partition->highwatermark;
2346|                r->in.req.req8.uptodateness_vector      = NULL;
2347|                r->in.req.req8.replica_flags            = partition->replica_flags;
2348|                r->in.req.req8.max_object_count         = 133;
2349|                r->in.req.req8.max_ndr_size             = 1336811;
2350|                r->in.req.req8.unknown4                 = 0;
2351|                r->in.req.req8.h1                       = 0;
2352|                r->in.req.req8.unique_ptr1              = 0;
2353|                r->in.req.req8.unique_ptr2              = 0;
2354|                r->in.req.req8.mapping_ctr.num_mappings = 0;
2355|                r->in.req.req8.mapping_ctr.mappings     = NULL;
2356|        } else {
2357|                *r->in.level                              = 5;
2358|                r->in.req.req5.destination_dsa_guid     = partition->destination_dsa_guid
->   ;
2359|                r->in.req.req5.source_dsa_invocation_id = partition->source_dsa_invocatio
->   n_id;
2360|                r->in.req.req5.naming_context           = &partition->nc;
2361|                r->in.req.req5.highwatermark            = partition->highwatermark;
2362|                r->in.req.req5.uptodateness_vector      = NULL;
2363|                r->in.req.req5.replica_flags            = partition->replica_flags;
2364|                r->in.req.req5.max_object_count         = 133;
2365|                r->in.req.req5.max_ndr_size             = 1336770;
2366|                r->in.req.req5.unknown4                 = 0;
2367|                r->in.req.req5.h1                       = 0;
2368|        }
2369|
2370|        /*
2371|         * we should try to use the drsuapi_p->pipe here, as w2k3 does
2372|         * but it seems that some extra flags in the DCERPC Bind call
2373|         * are needed for it. Or the same KRB5 TGS is needed on both
2374|         * connections.
2375|         */
2376|        req = dcerpc_drsuapi_DsGetNCChanges_send(drsuapi_p->pipe, r, r);
2377|        composite_continue_rpc(c, req, recv_fn, s);
2378|}
2379|
2380|static WERROR becomeDC_drsuapi_pull_partition_recv(struct libnet_BecomeDC_state *s,
2381|                                                   struct becomeDC_drsuapi *drsuapi_h,
2382|                                                   struct becomeDC_drsuapi *drsuapi_p,
2383|                                                   struct libnet_BecomeDC_Partition
->    *partition,
2384|                                                   struct drsuapi_DsGetNCChanges *r)
2385|{
2386|        uint32_t ctr_level = 0;
2387|        struct drsuapi_DsGetNCChangesCtr1 *ctr1 = NULL;
2388|        struct drsuapi_DsGetNCChangesCtr6 *ctr6 = NULL;
2389|        struct GUID *source_dsa_guid;
2390|        struct GUID *source_dsa_invocation_id;
2391|        struct drsuapi_DsReplicaHighWaterMark *new_highwatermark;
2392|        NTSTATUS nt_status;
```

```
2393|
2394|        if (!W_ERROR_IS_OK(r->out.result)) {
2395|                return r->out.result;
2396|        }
2397|
2398|        if (*r->out.level == 1) {
2399|                ctr_level = 1;
2400|                ctr1 = &r->out.ctr.ctr1;
2401|        } else if (*r->out.level == 2) {
2402|                ctr_level = 1;
2403|                ctr1 = r->out.ctr.ctr2.ctr.mszip1.ctr1;
2404|        } else if (*r->out.level == 6) {
2405|                ctr_level = 6;
2406|                ctr6 = &r->out.ctr.ctr6;
2407|        } else if (*r->out.level == 7 &&
2408|                   r->out.ctr.ctr7.level == 6 &&
2409|                   r->out.ctr.ctr7.type == DRSUAPI_COMPRESSION_TYPE_MSZIP) {
2410|                ctr_level = 6;
2411|                ctr6 = r->out.ctr.ctr7.ctr.mszip6.ctr6;
2412|        } else {
2413|                return WERR_BAD_NET_RESP;
2414|        }
2415|
2416|        switch (ctr_level) {
2417|        case 1:
2418|                source_dsa_guid                 = &ctr1->source_dsa_guid;
2419|                source_dsa_invocation_id        = &ctr1->source_dsa_invocation_id;
2420|                new_highwatermark               = &ctr1->new_highwatermark;
2421|                break;
2422|        case 6:
2423|                source_dsa_guid                 = &ctr6->source_dsa_guid;
2424|                source_dsa_invocation_id        = &ctr6->source_dsa_invocation_id;
2425|                new_highwatermark               = &ctr6->new_highwatermark;
2426|                break;
2427|        }
2428|
2429|        partition->highwatermark              = *new_highwatermark;
2430|        partition->source_dsa_guid            = *source_dsa_guid;
2431|        partition->source_dsa_invocation_id    = *source_dsa_invocation_id;
2432|
2433|        if (!partition->store_chunk) return WERR_OK;
2434|
2435|        s->_sc.domain          = &s->domain;
2436|        s->_sc.forest          = &s->forest;
2437|        s->_sc.source_dsa      = &s->source_dsa;
2438|        s->_sc.dest_dsa        = &s->dest_dsa;
2439|        s->_sc.partition       = partition;
2440|        s->_sc.ctr_level       = ctr_level;
2441|        s->_sc.ctr1            = ctr1;
2442|        s->_sc.ctr6            = ctr6;
2443|        /*
2444|         * we need to use the drsuapi_p->gensec_skey here,
2445|         * when we use drsuapi_p->pipe in the for this request
2446|         */
2447|        s->_sc.gensec_skey      = &drsuapi_p->gensec_skey;
2448|
2449|        nt_status = partition->store_chunk(s->callbacks.private_data, &s->_sc);
2450|        if (!NT_STATUS_IS_OK(nt_status)) {
2451|                return ntstatus_to_werror(nt_status);
2452|        }
2453|
2454|        return WERR_OK;
2455|}
2456|
2457|static void becomeDC_drsuapi3_pull_schema_recv(struct rpc_request *req);
2458|
2459|static void becomeDC_drsuapi3_pull_schema_send(struct libnet_BecomeDC_state *s)
2460|{
2461|        s->schema_part.nc.guid  = GUID_zero();
2462|        s->schema_part.nc.sid   = s->zero_sid;
2463|        s->schema_part.nc.dn    = s->forest.schema_dn_str;
2464|
2465|        s->schema_part.destination_dsa_guid     = s->drsuapi2.bind_guid;
2466|
2467|        s->schema_part.replica_flags    = DRSUAPI_DS_REPLICA_NEIGHBOUR_WRITEABLE
2468|                                        | DRSUAPI_DS_REPLICA_NEIGHBOUR_SYNC_ON_STARTUP
2469|                                        | DRSUAPI_DS_REPLICA_NEIGHBOUR_DO_SCHEDULED_SYNCS
->
2470|                                        | DRSUAPI_DS_REPLICA_NEIGHBOUR_FULL_IN_PROGRESS
2471|                                        | DRSUAPI_DS_REPLICA_NEIGHBOUR_NEVER_SYNCED
2472|                                        | DRSUAPI_DS_REPLICA_NEIGHBOUR_COMPRESS_CHANGES;
2473|
2474|        s->schema_part.store_chunk      = s->callbacks.schema_chunk;
2475|
2476|        becomeDC_drsuapi_pull_partition_send(s, &s->drsuapi2, &s->drsuapi3,
->    &s->schema_part,
2477|                                            becomeDC_drsuapi3_pull_schema_recv);
```

```
2478|}
2479|
2480|static void becomeDC_drsuapi3_pull_config_send(struct libnet_BecomeDC_state *s);
2481|
2482|static void becomeDC_drsuapi3_pull_schema_recv(struct rpc_request *req)
2483|{
2484|        struct libnet_BecomeDC_state *s = talloc_get_type(req->async.private,
2485|                                        struct libnet_BecomeDC_state);
2486|        struct composite_context *c = s->creq;
2487|        struct drsuapi_DsGetNCChanges *r = talloc_get_type(req->ndr.struct_ptr,
2488|                                         struct drsuapi_DsGetNCChanges);
2489|        WERROR status;
2490|
2491|        bool print = false;
2492|
2493|        if (req->p->conn->flags & DCERPC_DEBUG_PRINT_OUT) {
2494|                print = true;
2495|        }
2496|
2497|        c->status = dcerpc_ndr_request_recv(req);
2498|        if (!composite_is_ok(c)) return;
2499|
2500|        if (print) {
2501|                NDR_PRINT_OUT_DEBUG(drsuapi_DsGetNCChanges, r);
2502|        }
2503|
2504|        status = becomeDC_drsuapi_pull_partition_recv(s, &s->drsuapi2, &s->drsuapi3,
->    &s->schema_part, r);
2505|        if (!W_ERROR_IS_OK(status)) {
2506|                composite_error(c, werror_to_ntstatus(status));
2507|                return;
2508|        }
2509|
2510|        talloc_free(r);
2511|
2512|        if (s->schema_part.highwatermark.tmp_highest_usn >
->    s->schema_part.highwatermark.highest_usn) {
2513|                becomeDC_drsuapi_pull_partition_send(s, &s->drsuapi2, &s->drsuapi3,
->    &s->schema_part,
2514|                                                     becomeDC_drsuapi3_pull_schema_recv);
->
2515|                return;
2516|        }
2517|
2518|        becomeDC_drsuapi3_pull_config_send(s);
2519|}
2520|
2521|static void becomeDC_drsuapi3_pull_config_recv(struct rpc_request *req);
2522|
2523|static void becomeDC_drsuapi3_pull_config_send(struct libnet_BecomeDC_state *s)
2524|{
2525|        s->config_part.nc.guid  = GUID_zero();
2526|        s->config_part.nc.sid   = s->zero_sid;
2527|        s->config_part.nc.dn    = s->forest.config_dn_str;
2528|
2529|        s->config_part.destination_dsa_guid     = s->drsuapi2.bind_guid;
2530|
2531|        s->config_part.replica_flags    = DRSUAPI_DS_REPLICA_NEIGHBOUR_WRITEABLE
2532|                                        | DRSUAPI_DS_REPLICA_NEIGHBOUR_SYNC_ON_STARTUP
2533|                                        | DRSUAPI_DS_REPLICA_NEIGHBOUR_DO_SCHEDULED_SYNCS
->
2534|                                        | DRSUAPI_DS_REPLICA_NEIGHBOUR_FULL_IN_PROGRESS
2535|                                        | DRSUAPI_DS_REPLICA_NEIGHBOUR_NEVER_SYNCED
2536|                                        | DRSUAPI_DS_REPLICA_NEIGHBOUR_COMPRESS_CHANGES;
2537|
2538|        s->config_part.store_chunk      = s->callbacks.config_chunk;
2539|
2540|        becomeDC_drsuapi_pull_partition_send(s, &s->drsuapi2, &s->drsuapi3,
->    &s->config_part,
2541|                                           becomeDC_drsuapi3_pull_config_recv);
2542|}
2543|
2544|static void becomeDC_drsuapi3_pull_config_recv(struct rpc_request *req)
2545|{
2546|        struct libnet_BecomeDC_state *s = talloc_get_type(req->async.private,
2547|                                        struct libnet_BecomeDC_state);
2548|        struct composite_context *c = s->creq;
2549|        struct drsuapi_DsGetNCChanges *r = talloc_get_type(req->ndr.struct_ptr,
2550|                                         struct drsuapi_DsGetNCChanges);
2551|        WERROR status;
2552|
2553|        bool print = false;
2554|
2555|        if (req->p->conn->flags & DCERPC_DEBUG_PRINT_OUT) {
2556|                print = true;
2557|        }
2558|
```

```
2559|        c->status = dcerpc_ndr_request_recv(req);
2560|        if (!composite_is_ok(c)) return;
2561|
2562|        if (print) {
2563|                NDR_PRINT_OUT_DEBUG(drsuapi_DsGetNCChanges, r);
2564|        }
2565|
2566|        status = becomeDC_drsuapi_pull_partition_recv(s, &s->drsuapi2, &s->drsuapi3,
->    &s->config_part, r);
2567|        if (!W_ERROR_IS_OK(status)) {
2568|                composite_error(c, werror_to_ntstatus(status));
2569|                return;
2570|        }
2571|
2572|        talloc_free(r);
2573|
2574|        if (s->config_part.highwatermark.tmp_highest_usn >
->    s->config_part.highwatermark.highest_usn) {
2575|                becomeDC_drsuapi_pull_partition_send(s, &s->drsuapi2, &s->drsuapi3,
->    &s->config_part,
2576|                                                     becomeDC_drsuapi3_pull_config_recv);
->
2577|                return;
2578|        }
2579|
2580|        becomeDC_connect_ldap2(s);
2581|}
2582|
2583|static void becomeDC_drsuapi3_pull_domain_recv(struct rpc_request *req);
2584|
2585|static void becomeDC_drsuapi3_pull_domain_send(struct libnet_BecomeDC_state *s)
2586|{
2587|        s->domain_part.nc.guid  = GUID_zero();
2588|        s->domain_part.nc.sid   = s->zero_sid;
2589|        s->domain_part.nc.dn    = s->domain.dn_str;
2590|
2591|        s->domain_part.destination_dsa_guid     = s->drsuapi2.bind_guid;
2592|
2593|        s->domain_part.replica_flags    = DRSUAPI_DS_REPLICA_NEIGHBOUR_WRITEABLE
2594|                                        | DRSUAPI_DS_REPLICA_NEIGHBOUR_SYNC_ON_STARTUP
2595|                                        | DRSUAPI_DS_REPLICA_NEIGHBOUR_DO_SCHEDULED_SYNCS
->
2596|                                        | DRSUAPI_DS_REPLICA_NEIGHBOUR_FULL_IN_PROGRESS
2597|                                        | DRSUAPI_DS_REPLICA_NEIGHBOUR_NEVER_SYNCED
2598|                                        | DRSUAPI_DS_REPLICA_NEIGHBOUR_COMPRESS_CHANGES;
2599|
2600|        s->domain_part.store_chunk      = s->callbacks.domain_chunk;
2601|
2602|        becomeDC_drsuapi_pull_partition_send(s, &s->drsuapi2, &s->drsuapi3,
->    &s->domain_part,
2603|                                            becomeDC_drsuapi3_pull_domain_recv);
2604|}
2605|
2606|static void becomeDC_drsuapi_update_refs_send(struct libnet_BecomeDC_state *s,
2607|                                              struct becomeDC_drsuapi *drsuapi,
2608|                                              struct libnet_BecomeDC_Partition
->    *partition,
2609|                                              void (*recv_fn)(struct rpc_request *req));
2610|static void becomeDC_drsuapi2_update_refs_schema_recv(struct rpc_request *req);
2611|
2612|static void becomeDC_drsuapi3_pull_domain_recv(struct rpc_request *req)
2613|{
2614|        struct libnet_BecomeDC_state *s = talloc_get_type(req->async.private,
2615|                                                      struct libnet_BecomeDC_state);
2616|        struct composite_context *c = s->creq;
2617|        struct drsuapi_DsGetNCChanges *r = talloc_get_type(req->ndr.struct_ptr,
2618|                                               struct drsuapi_DsGetNCChanges);
2619|        WERROR status;
2620|        bool print = false;
2621|
2622|        if (req->p->conn->flags & DCERPC_DEBUG_PRINT_OUT) {
2623|                print = true;
2624|        }
2625|
2626|        c->status = dcerpc_ndr_request_recv(req);
2627|        if (!composite_is_ok(c)) return;
2628|
2629|        if (print) {
2630|                NDR_PRINT_OUT_DEBUG(drsuapi_DsGetNCChanges, r);
2631|        }
2632|
2633|        status = becomeDC_drsuapi_pull_partition_recv(s, &s->drsuapi2, &s->drsuapi3,
->    &s->domain_part, r);
2634|        if (!W_ERROR_IS_OK(status)) {
2635|                composite_error(c, werror_to_ntstatus(status));
2636|                return;
2637|        }
```

```
2638|
2639|        talloc_free(r);
2640|
2641|        if (s->domain_part.highwatermark.tmp_highest_usn >
->    s->domain_part.highwatermark.highest_usn) {
2642|                becomeDC_drsuapi_pull_partition_send(s, &s->drsuapi2, &s->drsuapi3,
->    &s->domain_part,
2643|                                                     becomeDC_drsuapi3_pull_domain_recv);
->
2644|                return;
2645|        }
2646|
2647|        becomeDC_drsuapi_update_refs_send(s, &s->drsuapi2, &s->schema_part,
2648|                                           becomeDC_drsuapi2_update_refs_schema_recv);
2649|}
2650|
2651|static void becomeDC_drsuapi_update_refs_send(struct libnet_BecomeDC_state *s,
2652|                                              struct becomeDC_drsuapi *drsuapi,
2653|                                              struct libnet_BecomeDC_Partition
->    *partition,
2654|                                              void (*recv_fn)(struct rpc_request *req))
2655|{
2656|        struct composite_context *c = s->creq;
2657|        struct rpc_request *req;
2658|        struct drsuapi_DsReplicaUpdateRefs *r;
2659|        const char *ntds_guid_str;
2660|        const char *ntds_dns_name;
2661|
2662|        r = talloc(s, struct drsuapi_DsReplicaUpdateRefs);
2663|        if (composite_nomem(r, c)) return;
2664|
2665|        ntds_guid_str = GUID_string(r, &s->dest_dsa.ntds_guid);
2666|        if (composite_nomem(ntds_guid_str, c)) return;
2667|
2668|        ntds_dns_name = talloc_asprintf(r, "%s._msdcs.%s",
2669|                                        ntds_guid_str,
2670|                                        s->domain.dns_name);
2671|        if (composite_nomem(ntds_dns_name, c)) return;
2672|
2673|        r->in.bind_handle               = &drsuapi->bind_handle;
2674|        r->in.level                    = 1;
2675|        r->in.req.req1.naming_context   = &partition->nc;
2676|        r->in.req.req1.dest_dsa_dns_name= ntds_dns_name;
2677|        r->in.req.req1.dest_dsa_guid    = s->dest_dsa.ntds_guid;
2678|        r->in.req.req1.options          = DRSUAPI_DS_REPLICA_UPDATE_ADD_REFERENCE
2679|                                        | DRSUAPI_DS_REPLICA_UPDATE_DELETE_REFERENCE
2680|                                        | DRSUAPI_DS_REPLICA_UPDATE_0x00000010;
2681|
2682|        req = dcerpc_drsuapi_DsReplicaUpdateRefs_send(drsuapi->pipe, r, r);
2683|        composite_continue_rpc(c, req, recv_fn, s);
2684|}
2685|
2686|static void becomeDC_drsuapi2_update_refs_config_recv(struct rpc_request *req);
2687|
2688|static void becomeDC_drsuapi2_update_refs_schema_recv(struct rpc_request *req)
2689|{
2690|        struct libnet_BecomeDC_state *s = talloc_get_type(req->async.private,
2691|                                          struct libnet_BecomeDC_state);
2692|        struct composite_context *c = s->creq;
2693|        struct drsuapi_DsReplicaUpdateRefs *r = talloc_get_type(req->ndr.struct_ptr,
2694|                                           struct drsuapi_DsReplicaUpdateRefs);
2695|        bool print = false;
2696|
2697|        if (req->p->conn->flags & DCERPC_DEBUG_PRINT_OUT) {
2698|                print = true;
2699|        }
2700|
2701|        c->status = dcerpc_ndr_request_recv(req);
2702|        if (!composite_is_ok(c)) return;
2703|
2704|        if (print) {
2705|                NDR_PRINT_OUT_DEBUG(drsuapi_DsReplicaUpdateRefs, r);
2706|        }
2707|
2708|        if (!W_ERROR_IS_OK(r->out.result)) {
2709|                composite_error(c, werror_to_ntstatus(r->out.result));
2710|                return;
2711|        }
2712|
2713|        talloc_free(r);
2714|
2715|        becomeDC_drsuapi_update_refs_send(s, &s->drsuapi2, &s->config_part,
2716|                                           becomeDC_drsuapi2_update_refs_config_recv);
2717|}
2718|
2719|static void becomeDC_drsuapi2_update_refs_domain_recv(struct rpc_request *req);
2720|
```

```
2721|static void becomeDC_drsuapi2_update_refs_config_recv(struct rpc_request *req)
2722|{
2723|        struct libnet_BecomeDC_state *s = talloc_get_type(req->async.private,
2724|                                         struct libnet_BecomeDC_state);
2725|        struct composite_context *c = s->creq;
2726|        struct drsuapi_DsReplicaUpdateRefs *r = talloc_get_type(req->ndr.struct_ptr,
2727|                                          struct drsuapi_DsReplicaUpdateRefs);
2728|
2729|        c->status = dcerpc_ndr_request_recv(req);
2730|        if (!composite_is_ok(c)) return;
2731|
2732|        if (!W_ERROR_IS_OK(r->out.result)) {
2733|                composite_error(c, werror_to_ntstatus(r->out.result));
2734|                return;
2735|        }
2736|
2737|        talloc_free(r);
2738|
2739|        becomeDC_drsuapi_update_refs_send(s, &s->drsuapi2, &s->domain_part,
2740|                                          becomeDC_drsuapi2_update_refs_domain_recv);
2741|}
2742|
2743|static void becomeDC_drsuapi2_update_refs_domain_recv(struct rpc_request *req)
2744|{
2745|        struct libnet_BecomeDC_state *s = talloc_get_type(req->async.private,
2746|                                         struct libnet_BecomeDC_state);
2747|        struct composite_context *c = s->creq;
2748|        struct drsuapi_DsReplicaUpdateRefs *r = talloc_get_type(req->ndr.struct_ptr,
2749|                                          struct drsuapi_DsReplicaUpdateRefs);
2750|
2751|        c->status = dcerpc_ndr_request_recv(req);
2752|        if (!composite_is_ok(c)) return;
2753|
2754|        if (!W_ERROR_IS_OK(r->out.result)) {
2755|                composite_error(c, werror_to_ntstatus(r->out.result));
2756|                return;
2757|        }
2758|
2759|        talloc_free(r);
2760|
2761|        /* TODO: use DDNS updates and register dns names */
2762|        composite_done(c);
2763|}
2764|
2765|static NTSTATUS becomeDC_ldap2_modify_computer(struct libnet_BecomeDC_state *s)
2766|{
2767|        int ret;
2768|        struct ldb_message *msg;
2769|        uint32_t i;
2770|        uint32_t user_account_control = UF_SERVER_TRUST_ACCOUNT |
2771|                                        UF_TRUSTED_FOR_DELEGATION;
2772|
2773|        /* as the value is already as we want it to be, we're done */
2774|        if (s->dest_dsa.user_account_control == user_account_control) {
2775|                return NT_STATUS_OK;
2776|        }
2777|
2778|        /* make a 'modify' msg, and only for serverReference */
2779|        msg = ldb_msg_new(s);
2780|        NT_STATUS_HAVE_NO_MEMORY(msg);
2781|        msg->dn = ldb_dn_new(msg, s->ldap2.ldb, s->dest_dsa.computer_dn_str);
2782|        NT_STATUS_HAVE_NO_MEMORY(msg->dn);
2783|
2784|        ret = ldb_msg_add_fmt(msg, "userAccountControl", "%u", user_account_control);
2785|        if (ret != 0) {
2786|                talloc_free(msg);
2787|                return NT_STATUS_NO_MEMORY;
2788|        }
2789|
2790|        /* mark all the message elements (should be just one)
2791|           as LDB_FLAG_MOD_REPLACE */
2792|        for (i=0;i<msg->num_elements;i++) {
2793|                msg->elements[i].flags = LDB_FLAG_MOD_REPLACE;
2794|        }
2795|
2796|        ret = ldb_modify(s->ldap2.ldb, msg);
2797|        talloc_free(msg);
2798|        if (ret != LDB_SUCCESS) {
2799|                return NT_STATUS_LDAP(ret);
2800|        }
2801|
2802|        s->dest_dsa.user_account_control = user_account_control;
2803|
2804|        return NT_STATUS_OK;
2805|}
2806|
2807|static NTSTATUS becomeDC_ldap2_move_computer(struct libnet_BecomeDC_state *s)
```

```
2808|{
2809|        int ret;
2810|        struct ldb_result *r;
2811|        struct ldb_dn *basedn;
2812|        struct ldb_dn *old_dn;
2813|        struct ldb_dn *new_dn;
2814|        static const char *_1_1_attrs[] = {
2815|                "1.1",
2816|                NULL
2817|        };
2818|
2819|        basedn = ldb_dn_new_fmt(s, s->ldap2.ldb, "<WKGUID=a361b2ffffd211d1aa4b00c04fd7d83
->   a,%s>",
2820|                                 s->domain.dn_str);
2821|        NT_STATUS_HAVE_NO_MEMORY(basedn);
2822|
2823|        ret = ldb_search(s->ldap2.ldb, basedn, LDB_SCOPE_BASE,
2824|                          "(objectClass=*)", _1_1_attrs, &r);
2825|        talloc_free(basedn);
2826|        if (ret != LDB_SUCCESS) {
2827|                return NT_STATUS_LDAP(ret);
2828|        } else if (r->count != 1) {
2829|                talloc_free(r);
2830|                return NT_STATUS_INVALID_NETWORK_RESPONSE;
2831|        }
2832|
2833|        old_dn = ldb_dn_new(r, s->ldap2.ldb, s->dest_dsa.computer_dn_str);
2834|        NT_STATUS_HAVE_NO_MEMORY(old_dn);
2835|
2836|        new_dn = r->msgs[0]->dn;
2837|
2838|        if (!ldb_dn_add_child_fmt(new_dn, "CN=%s", s->dest_dsa.netbios_name)) {
2839|                talloc_free(r);
2840|                return NT_STATUS_NO_MEMORY;
2841|        }
2842|
2843|        if (ldb_dn_compare(old_dn, new_dn) == 0) {
2844|                /* we don't need to rename if the old and new dn match */
2845|                talloc_free(r);
2846|                return NT_STATUS_OK;
2847|        }
2848|
2849|        ret = ldb_rename(s->ldap2.ldb, old_dn, new_dn);
2850|        if (ret != LDB_SUCCESS) {
2851|                talloc_free(r);
2852|                return NT_STATUS_LDAP(ret);
2853|        }
2854|
2855|        s->dest_dsa.computer_dn_str = ldb_dn_alloc_linearized(s, new_dn);
2856|        NT_STATUS_HAVE_NO_MEMORY(s->dest_dsa.computer_dn_str);
2857|
2858|        talloc_free(r);
2859|
2860|        return NT_STATUS_OK;
2861|}
2862|
2863|static void becomeDC_connect_ldap2(struct libnet_BecomeDC_state *s)
2864|{
2865|        struct composite_context *c = s->creq;
2866|
2867|        c->status = becomeDC_ldap_connect(s, &s->ldap2);
2868|        if (!composite_is_ok(c)) return;
2869|
2870|        c->status = becomeDC_ldap2_modify_computer(s);
2871|        if (!composite_is_ok(c)) return;
2872|
2873|        c->status = becomeDC_ldap2_move_computer(s);
2874|        if (!composite_is_ok(c)) return;
2875|
2876|        becomeDC_drsuapi3_pull_domain_send(s);
2877|}
2878|
2879|struct composite_context *libnet_BecomeDC_send(struct libnet_context *ctx, TALLOC_CTX
->    *mem_ctx, struct libnet_BecomeDC *r)
2880|{
2881|        struct composite_context *c;
2882|        struct libnet_BecomeDC_state *s;
2883|        char *tmp_name;
2884|
2885|        c = composite_create(mem_ctx, ctx->event_ctx);
2886|        if (c == NULL) return NULL;
2887|
2888|        s = talloc_zero(c, struct libnet_BecomeDC_state);
2889|        if (composite_nomem(s, c)) return c;
2890|        c->private_data = s;
2891|        s->creq         = c;
2892|        s->libnet       = ctx;
```

```
2893|
2894|        /* Domain input */
2895|        s->domain.dns_name      = talloc_strdup(s, r->in.domain_dns_name);
2896|        if (composite_nomem(s->domain.dns_name, c)) return c;
2897|        s->domain.netbios_name  = talloc_strdup(s, r->in.domain_netbios_name);
2898|        if (composite_nomem(s->domain.netbios_name, c)) return c;
2899|        s->domain.sid           = dom_sid_dup(s, r->in.domain_sid);
2900|        if (composite_nomem(s->domain.sid, c)) return c;
2901|
2902|        /* Source DSA input */
2903|        s->source_dsa.address   = talloc_strdup(s, r->in.source_dsa_address);
2904|        if (composite_nomem(s->source_dsa.address, c)) return c;
2905|
2906|        /* Destination DSA input */
2907|        s->dest_dsa.netbios_name= talloc_strdup(s, r->in.dest_dsa_netbios_name);
2908|        if (composite_nomem(s->dest_dsa.netbios_name, c)) return c;
2909|
2910|        /* Destination DSA dns_name construction */
2911|        tmp_name        = strlower_talloc(s, s->dest_dsa.netbios_name);
2912|        if (composite_nomem(tmp_name, c)) return c;
2913|        tmp_name        = talloc_asprintf_append(tmp_name, ".%s",s->domain.dns_name);
2914|        if (composite_nomem(tmp_name, c)) return c;
2915|        s->dest_dsa.dns_name    = tmp_name;
2916|
2917|        /* Callback function pointers */
2918|        s->callbacks = r->in.callbacks;
2919|
2920|        becomeDC_send_cldap(s);
2921|        return c;
2922|}
2923|
2924|NTSTATUS libnet_BecomeDC_recv(struct composite_context *c, TALLOC_CTX *mem_ctx, struct
->    libnet_BecomeDC *r)
2925|{
2926|        NTSTATUS status;
2927|
2928|        status = composite_wait(c);
2929|
2930|        ZERO_STRUCT(r->out);
2931|
2932|        talloc_free(c);
2933|        return status;
2934|}
2935|
2936|NTSTATUS libnet_BecomeDC(struct libnet_context *ctx, TALLOC_CTX *mem_ctx, struct
->    libnet_BecomeDC *r)
2937|{
2938|        NTSTATUS status;
2939|        struct composite_context *c;
2940|        c = libnet_BecomeDC_send(ctx, mem_ctx, r);
2941|        status = libnet_BecomeDC_recv(c, mem_ctx, r);
2942|        return status;
2943|}
```

A9. source/libnet/libnet_unbecome_dc.c

source/libnet/libnet_unbecome_dc.c:

```
  1|/*
  2|   Unix SMB/CIFS implementation.
  3|
  4|   Copyright (C) Stefan Metzmacher <metze@samba.org> 2006
  5|
  6|   This program is free software; you can redistribute it and/or modify
  7|   it under the terms of the GNU General Public License as published by
  8|   the Free Software Foundation; either version 2 of the License, or
  9|   (at your option) any later version.
 10|
 11|   This program is distributed in the hope that it will be useful,
 12|   but WITHOUT ANY WARRANTY; without even the implied warranty of
 13|   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 14|   GNU General Public License for more details.
 15|
 16|   You should have received a copy of the GNU General Public License
 17|   along with this program; if not, write to the Free Software
 18|   Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 19|*/
 20|
 21|#include "includes.h"
 22|#include "libnet/libnet.h"
 23|#include "libcli/composite/composite.h"
 24|#include "libcli/cldap/cldap.h"
 25|#include "lib/ldb/include/ldb.h"
 26|#include "lib/ldb/include/ldb_errors.h"
 27|#include "lib/db_wrap.h"
 28|#include "dsdb/samdb/samdb.h"
 29|#include "dsdb/common/flags.h"
 30|#include "librpc/gen_ndr/ndr_drsuapi_c.h"
 31|
 32|/****************************************************************************
 33| * Windows 2003 (w2k3) does the following steps when changing the server role
 34| * from domain controller back to domain member
 35| *
 36| * We mostly do the same.
 37| ****************************************************************************/
 38|
 39|/*
 40| * lookup DC:
 41| * - using nbt name<1C> request and a samlogon mailslot request
 42| * or
 43| * - using a DNS SRV _ldap._tcp.dc._msdcs. request and a CLDAP netlogon request
 44| *
 45| * see: unbecomeDC_send_cldap() and unbecomeDC_recv_cldap()
 46| */
 47|
 48|/*
 49| * Open 1st LDAP connection to the DC using admin credentials
 50| *
 51| * see: unbecomeDC_ldap_connect()
 52| */
 53|
 54|/*
 55| * LDAP search 1st LDAP connection:
 56| *
 57| * see: unbecomeDC_ldap_rootdse()
 58| *
 59| * Request:
 60| *      basedn: ""
 61| *      scope:  base
 62| *      filter: (objectClass=*)
 63| *      attrs:  defaultNamingContext
 64| *              configurationNamingContext
 65| * Result:
 66| *      ""
 67| *              defaultNamingContext:   <domain_partition>
 68| *              configurationNamingContext:CN=Configuration,<domain_partition>
 69| */
 70|
 71|/*
 72| * LDAP search 1st LDAP connection:
 73| *
 74| * see: unbecomeDC_ldap_computer_object()
 75| *
 76| * Request:
 77| *      basedn: <domain_partition>
 78| *      scope:  sub
 79| *      filter: (&(|(objectClass=user)(objectClass=computer))(sAMAccountName=<new_dc_acco
```

```
->   unt_name>))
  80| *      attrs:  distinguishedName
  81| *              userAccountControl
  82| * Result:
  83| *      CN=<new_dc_netbios_name>,CN=Domain Controllers,<domain_partition>
  84| *              distinguishedName:      CN=<new_dc_netbios_name>,CN=Domain
->    Controllers,<domain_partition>
  85| *              userAccoountControl:    532480 <0x82000>
  86| */
  87|
  88|/*
  89| * LDAP search 1st LDAP connection:
  90| *
  91| * see: unbecomeDC_ldap_modify_computer()
  92| *
  93| * Request:
  94| *      basedn: CN=<new_dc_netbios_name>,CN=Computers,<domain_partition>
  95| *      scope:  base
  96| *      filter: (objectClass=*)
  97| *      attrs:  userAccountControl
  98| * Result:
  99| *      CN=<new_dc_netbios_name>,CN=Computers,<domain_partition>
 100| *              userAccoountControl:    532480 <0x82000>
 101| */
 102|
 103|/*
 104| * LDAP modify 1st LDAP connection:
 105| *
 106| * see: unbecomeDC_ldap_modify_computer()
 107| *
 108| * Request (replace):
 109| *      CN=<new_dc_netbios_name>,CN=Computers,<domain_partition>
 110| *      userAccoountControl:    4096 <0x1000>
 111| * Result:
 112| *      <success>
 113| */
 114|
 115|/*
 116| * LDAP search 1st LDAP connection:
 117| *
 118| * see: unbecomeDC_ldap_move_computer()
 119| *
 120| * Request:
 121| *      basedn: <WKGUID=aa312825768811d1aded00c04fd8d5cd,<domain_partition>>
 122| *      scope:  base
 123| *      filter: (objectClass=*)
 124| *      attrs:  1.1
 125| * Result:
 126| *      CN=Computers,<domain_partition>
 127| */
 128|
 129|/*
 130| * LDAP search 1st LDAP connection:
 131| *
 132| * not implemented because it doesn't give any new information
 133| *
 134| * Request:
 135| *      basedn: CN=Computers,<domain_partition>
 136| *      scope:  base
 137| *      filter: (objectClass=*)
 138| *      attrs:  distinguishedName
 139| * Result:
 140| *      CN=Computers,<domain_partition>
 141| *              distinguishedName:      CN=Computers,<domain_partition>
 142| */
 143|
 144|/*
 145| * LDAP modifyRDN 1st LDAP connection:
 146| *
 147| * see: unbecomeDC_ldap_move_computer()
 148| *
 149| * Request:
 150| *      entry:          CN=<new_dc_netbios_name>,CN=Domain
->    Controllers,<domain_partition>
 151| *      newrdn:         CN=<new_dc_netbios_name>
 152| *      deleteoldrdn:   TRUE
 153| *      newparent:      CN=Computers,<domain_partition>
 154| * Result:
 155| *      <success>
 156| */
 157|
 158|/*
 159| * LDAP unbind on the 1st LDAP connection
 160| *
 161| * not implemented, because it's not needed...
 162| */
 163|
```

```
 164|/*
 165| * Open 1st DRSUAPI connection to the DC using admin credentials
 166| * DsBind with DRSUAPI_DS_BIND_GUID ("e24d201a-4fd6-11d1-a3da-0000f875ae0d")
 167| *
 168| * see: unbecomeDC_drsuapi_connect_send(), unbecomeDC_drsuapi_connect_recv(),
 169| *      unbecomeDC_drsuapi_bind_send() and unbecomeDC_drsuapi_bind_recv()
 170| */
 171|
 172|/*
 173| * DsRemoveDsServer to remove the
 174| * CN=<machine_name>,CN=Servers,CN=<site_name>,CN=Configuration,<domain_partition>
 175| * and CN=NTDS Settings,CN=<machine_name>,CN=Servers,CN=<site_name>,CN=Configuration,<dom
->   ain_partition>
 176| * on the 1st DRSUAPI connection
 177| *
 178| * see: unbecomeDC_drsuapi_remove_ds_server_send() and unbecomeDC_drsuapi_remove_ds_serve
->   r_recv()
 179| */
 180|
 181|/*
 182| * DsUnbind on the 1st DRSUAPI connection
 183| *
 184| * not implemented, because it's not needed...
 185| */
 186|
 187|
 188|struct libnet_UnbecomeDC_state {
 189|        struct composite_context *creq;
 190|
 191|        struct libnet_context *libnet;
 192|
 193|        struct {
 194|                struct cldap_socket *sock;
 195|                struct cldap_netlogon io;
 196|                struct nbt_cldap_netlogon_5 netlogon5;
 197|        } cldap;
 198|
 199|        struct {
 200|                struct ldb_context *ldb;
 201|        } ldap;
 202|
 203|        struct {
 204|                struct dcerpc_binding *binding;
 205|                struct dcerpc_pipe *pipe;
 206|                struct drsuapi_DsBind bind_r;
 207|                struct GUID bind_guid;
 208|                struct drsuapi_DsBindInfoCtr bind_info_ctr;
 209|                struct drsuapi_DsBindInfo28 local_info28;
 210|                struct drsuapi_DsBindInfo28 remote_info28;
 211|                struct policy_handle bind_handle;
 212|                struct drsuapi_DsRemoveDSServer rm_ds_srv_r;
 213|        } drsuapi;
 214|
 215|        struct {
 216|                /* input */
 217|                const char *dns_name;
 218|                const char *netbios_name;
 219|
 220|                /* constructed */
 221|                struct GUID guid;
 222|                const char *dn_str;
 223|        } domain;
 224|
 225|        struct {
 226|                /* constructed */
 227|                const char *config_dn_str;
 228|        } forest;
 229|
 230|        struct {
 231|                /* input */
 232|                const char *address;
 233|
 234|                /* constructed */
 235|                const char *dns_name;
 236|                const char *netbios_name;
 237|                const char *site_name;
 238|        } source_dsa;
 239|
 240|        struct {
 241|                /* input */
 242|                const char *netbios_name;
 243|
 244|                /* constructed */
 245|                const char *dns_name;
 246|                const char *site_name;
 247|                const char *computer_dn_str;
 248|                const char *server_dn_str;
```

```
249|                uint32_t user_account_control;
250|        } dest_dsa;
251|};
252|
253|static void unbecomeDC_recv_cldap(struct cldap_request *req);
254|
255|static void unbecomeDC_send_cldap(struct libnet_UnbecomeDC_state *s)
256|{
257|        struct composite_context *c = s->creq;
258|        struct cldap_request *req;
259|
260|        s->cldap.io.in.dest_address     = s->source_dsa.address;
261|        s->cldap.io.in.realm           = s->domain.dns_name;
262|        s->cldap.io.in.host            = s->dest_dsa.netbios_name;
263|        s->cldap.io.in.user            = NULL;
264|        s->cldap.io.in.domain_guid     = NULL;
265|        s->cldap.io.in.domain_sid      = NULL;
266|        s->cldap.io.in.acct_control    = -1;
267|        s->cldap.io.in.version         = 6;
268|
269|        s->cldap.sock = cldap_socket_init(s, s->libnet->event_ctx);
270|        if (composite_nomem(s->cldap.sock, c)) return;
271|
272|        req = cldap_netlogon_send(s->cldap.sock, &s->cldap.io);
273|        if (composite_nomem(req, c)) return;
274|        req->async.fn          = unbecomeDC_recv_cldap;
275|        req->async.private     = s;
276|}
277|
278|static void unbecomeDC_connect_ldap(struct libnet_UnbecomeDC_state *s);
279|
280|static void unbecomeDC_recv_cldap(struct cldap_request *req)
281|{
282|        struct libnet_UnbecomeDC_state *s = talloc_get_type(req->async.private,
283|                                                            struct libnet_UnbecomeDC_state);
284|        struct composite_context *c = s->creq;
285|
286|        c->status = cldap_netlogon_recv(req, s, &s->cldap.io);
287|        if (!composite_is_ok(c)) return;
288|
289|        s->cldap.netlogon5 = s->cldap.io.out.netlogon.logon5;
290|
291|        s->domain.dns_name             = s->cldap.netlogon5.dns_domain;
292|        s->domain.netbios_name         = s->cldap.netlogon5.domain;
293|        s->domain.guid                 = s->cldap.netlogon5.domain_uuid;
294|
295|        s->source_dsa.dns_name         = s->cldap.netlogon5.pdc_dns_name;
296|        s->source_dsa.netbios_name     = s->cldap.netlogon5.pdc_name;
297|        s->source_dsa.site_name        = s->cldap.netlogon5.server_site;
298|
299|        s->dest_dsa.site_name          = s->cldap.netlogon5.client_site;
300|
301|        unbecomeDC_connect_ldap(s);
302|}
303|
304|static NTSTATUS unbecomeDC_ldap_connect(struct libnet_UnbecomeDC_state *s)
305|{
306|        char *url;
307|
308|        url = talloc_asprintf(s, "ldap://%s/", s->source_dsa.dns_name);
309|        NT_STATUS_HAVE_NO_MEMORY(url);
310|
311|        s->ldap.ldb = ldb_wrap_connect(s, url,
312|                                        NULL,
313|                                        s->libnet->cred,
314|                                        0, NULL);
315|        talloc_free(url);
316|        if (s->ldap.ldb == NULL) {
317|                return NT_STATUS_UNEXPECTED_NETWORK_ERROR;
318|        }
319|
320|        return NT_STATUS_OK;
321|}
322|
323|static NTSTATUS unbecomeDC_ldap_rootdse(struct libnet_UnbecomeDC_state *s)
324|{
325|        int ret;
326|        struct ldb_result *r;
327|        struct ldb_dn *basedn;
328|        static const char *attrs[] = {
329|                "defaultNamingContext",
330|                "configurationNamingContext",
331|                NULL
332|        };
333|
334|        basedn = ldb_dn_new(s, s->ldap.ldb, NULL);
335|        NT_STATUS_HAVE_NO_MEMORY(basedn);
```

```
 336|
 337|        ret = ldb_search(s->ldap.ldb, basedn, LDB_SCOPE_BASE,
 338|                         "(objectClass=*)", attrs, &r);
 339|        talloc_free(basedn);
 340|        if (ret != LDB_SUCCESS) {
 341|                return NT_STATUS_LDAP(ret);
 342|        } else if (r->count != 1) {
 343|                talloc_free(r);
 344|                return NT_STATUS_INVALID_NETWORK_RESPONSE;
 345|        }
 346|        talloc_steal(s, r);
 347|
 348|        s->domain.dn_str        = ldb_msg_find_attr_as_string(r->msgs[0],
->    "defaultNamingContext", NULL);
 349|        if (!s->domain.dn_str) return NT_STATUS_INVALID_NETWORK_RESPONSE;
 350|        talloc_steal(s, s->domain.dn_str);
 351|
 352|        s->forest.config_dn_str = ldb_msg_find_attr_as_string(r->msgs[0],
->    "configurationNamingContext", NULL);
 353|        if (!s->forest.config_dn_str) return NT_STATUS_INVALID_NETWORK_RESPONSE;
 354|        talloc_steal(s, s->forest.config_dn_str);
 355|
 356|        s->dest_dsa.server_dn_str = talloc_asprintf(s, "CN=%s,CN=Servers,CN=%s,CN=Sites,%
->   s",
 357|                                                     s->dest_dsa.netbios_name,
 358|                                                     s->dest_dsa.site_name,
 359|                                                     s->forest.config_dn_str);
 360|        NT_STATUS_HAVE_NO_MEMORY(s->dest_dsa.server_dn_str);
 361|
 362|        talloc_free(r);
 363|        return NT_STATUS_OK;
 364|}
 365|
 366|static NTSTATUS unbecomeDC_ldap_computer_object(struct libnet_UnbecomeDC_state *s)
 367|{
 368|        int ret;
 369|        struct ldb_result *r;
 370|        struct ldb_dn *basedn;
 371|        char *filter;
 372|        static const char *attrs[] = {
 373|                "distinguishedName",
 374|                "userAccountControl",
 375|                NULL
 376|        };
 377|
 378|        basedn = ldb_dn_new(s, s->ldap.ldb, s->domain.dn_str);
 379|        NT_STATUS_HAVE_NO_MEMORY(basedn);
 380|
 381|        filter = talloc_asprintf(basedn, "(&(|(objectClass=user)(objectClass=computer))(s
->   AMAccountName=%s$))",
 382|                                  s->dest_dsa.netbios_name);
 383|        NT_STATUS_HAVE_NO_MEMORY(filter);
 384|
 385|        ret = ldb_search(s->ldap.ldb, basedn, LDB_SCOPE_SUBTREE,
 386|                         filter, attrs, &r);
 387|        talloc_free(basedn);
 388|        if (ret != LDB_SUCCESS) {
 389|                return NT_STATUS_LDAP(ret);
 390|        } else if (r->count != 1) {
 391|                talloc_free(r);
 392|                return NT_STATUS_INVALID_NETWORK_RESPONSE;
 393|        }
 394|
 395|        s->dest_dsa.computer_dn_str     = samdb_result_string(r->msgs[0],
->    "distinguishedName", NULL);
 396|        if (!s->dest_dsa.computer_dn_str) return NT_STATUS_INVALID_NETWORK_RESPONSE;
 397|        talloc_steal(s, s->dest_dsa.computer_dn_str);
 398|
 399|        s->dest_dsa.user_account_control = samdb_result_uint(r->msgs[0],
->    "userAccountControl", 0);
 400|
 401|        talloc_free(r);
 402|        return NT_STATUS_OK;
 403|}
 404|
 405|static NTSTATUS unbecomeDC_ldap_modify_computer(struct libnet_UnbecomeDC_state *s)
 406|{
 407|        int ret;
 408|        struct ldb_message *msg;
 409|        uint32_t user_account_control = UF_WORKSTATION_TRUST_ACCOUNT;
 410|        uint32_t i;
 411|
 412|        /* as the value is already as we want it to be, we're done */
 413|        if (s->dest_dsa.user_account_control == user_account_control) {
 414|                return NT_STATUS_OK;
 415|        }
 416|
```

```
417|        /* make a 'modify' msg, and only for serverReference */
418|        msg = ldb_msg_new(s);
419|        NT_STATUS_HAVE_NO_MEMORY(msg);
420|        msg->dn = ldb_dn_new(msg, s->ldap.ldb, s->dest_dsa.computer_dn_str);
421|        NT_STATUS_HAVE_NO_MEMORY(msg->dn);
422|
423|        ret = ldb_msg_add_fmt(msg, "userAccountControl", "%u", user_account_control);
424|        if (ret != 0) {
425|                talloc_free(msg);
426|                return NT_STATUS_NO_MEMORY;
427|        }
428|
429|        /* mark all the message elements (should be just one)
430|           as LDB_FLAG_MOD_REPLACE */
431|        for (i=0;i<msg->num_elements;i++) {
432|                msg->elements[i].flags = LDB_FLAG_MOD_REPLACE;
433|        }
434|
435|        ret = ldb_modify(s->ldap.ldb, msg);
436|        talloc_free(msg);
437|        if (ret != LDB_SUCCESS) {
438|                return NT_STATUS_LDAP(ret);
439|        }
440|
441|        s->dest_dsa.user_account_control = user_account_control;
442|
443|        return NT_STATUS_OK;
444|}
445|
446|static NTSTATUS unbecomeDC_ldap_move_computer(struct libnet_UnbecomeDC_state *s)
447|{
448|        int ret;
449|        struct ldb_result *r;
450|        struct ldb_dn *basedn;
451|        struct ldb_dn *old_dn;
452|        struct ldb_dn *new_dn;
453|        static const char *_1_1_attrs[] = {
454|                "1.1",
455|                NULL
456|        };
457|
458|        basedn = ldb_dn_new_fmt(s, s->ldap.ldb, "<WKGUID=aa312825768811d1aded00c04fd8d5cd
->   ,%s>",
459|                                 s->domain.dn_str);
460|        NT_STATUS_HAVE_NO_MEMORY(basedn);
461|
462|        ret = ldb_search(s->ldap.ldb, basedn, LDB_SCOPE_BASE,
463|                           "(objectClass=*)", _1_1_attrs, &r);
464|        talloc_free(basedn);
465|        if (ret != LDB_SUCCESS) {
466|                return NT_STATUS_LDAP(ret);
467|        } else if (r->count != 1) {
468|                talloc_free(r);
469|                return NT_STATUS_INVALID_NETWORK_RESPONSE;
470|        }
471|
472|        old_dn = ldb_dn_new(r, s->ldap.ldb, s->dest_dsa.computer_dn_str);
473|        NT_STATUS_HAVE_NO_MEMORY(old_dn);
474|
475|        new_dn = r->msgs[0]->dn;
476|
477|        if (!ldb_dn_add_child_fmt(new_dn, "CN=%s", s->dest_dsa.netbios_name)) {
478|                talloc_free(r);
479|                return NT_STATUS_NO_MEMORY;
480|        }
481|
482|        if (ldb_dn_compare(old_dn, new_dn) == 0) {
483|                /* we don't need to rename if the old and new dn match */
484|                talloc_free(r);
485|                return NT_STATUS_OK;
486|        }
487|
488|        ret = ldb_rename(s->ldap.ldb, old_dn, new_dn);
489|        if (ret != LDB_SUCCESS) {
490|                talloc_free(r);
491|                return NT_STATUS_LDAP(ret);
492|        }
493|
494|        s->dest_dsa.computer_dn_str = ldb_dn_alloc_linearized(s, new_dn);
495|        NT_STATUS_HAVE_NO_MEMORY(s->dest_dsa.computer_dn_str);
496|
497|        talloc_free(r);
498|
499|        return NT_STATUS_OK;
500|}
501|
502|static void unbecomeDC_drsuapi_connect_send(struct libnet_UnbecomeDC_state *s);
```

```
 503|
 504|static void unbecomeDC_connect_ldap(struct libnet_UnbecomeDC_state *s)
 505|{
 506|        struct composite_context *c = s->creq;
 507|
 508|        c->status = unbecomeDC_ldap_connect(s);
 509|        if (!composite_is_ok(c)) return;
 510|
 511|        c->status = unbecomeDC_ldap_rootdse(s);
 512|        if (!composite_is_ok(c)) return;
 513|
 514|        c->status = unbecomeDC_ldap_computer_object(s);
 515|        if (!composite_is_ok(c)) return;
 516|
 517|        c->status = unbecomeDC_ldap_modify_computer(s);
 518|        if (!composite_is_ok(c)) return;
 519|
 520|        c->status = unbecomeDC_ldap_move_computer(s);
 521|        if (!composite_is_ok(c)) return;
 522|
 523|        unbecomeDC_drsuapi_connect_send(s);
 524|}
 525|
 526|static void unbecomeDC_drsuapi_connect_recv(struct composite_context *creq);
 527|
 528|static void unbecomeDC_drsuapi_connect_send(struct libnet_UnbecomeDC_state *s)
 529|{
 530|        struct composite_context *c = s->creq;
 531|        struct composite_context *creq;
 532|        char *binding_str;
 533|
 534|        binding_str = talloc_asprintf(s, "ncacn_ip_tcp:%s[seal]",
->     s->source_dsa.dns_name);
 535|        if (composite_nomem(binding_str, c)) return;
 536|
 537|        c->status = dcerpc_parse_binding(s, binding_str, &s->drsuapi.binding);
 538|        talloc_free(binding_str);
 539|        if (!composite_is_ok(c)) return;
 540|
 541|        creq = dcerpc_pipe_connect_b_send(s, s->drsuapi.binding, &dcerpc_table_drsuapi,
 542|                                          s->libnet->cred, s->libnet->event_ctx);
 543|        composite_continue(c, creq, unbecomeDC_drsuapi_connect_recv, s);
 544|}
 545|
 546|static void unbecomeDC_drsuapi_bind_send(struct libnet_UnbecomeDC_state *s);
 547|
 548|static void unbecomeDC_drsuapi_connect_recv(struct composite_context *req)
 549|{
 550|        struct libnet_UnbecomeDC_state *s = talloc_get_type(req->async.private_data,
 551|                                             struct libnet_UnbecomeDC_state);
 552|        struct composite_context *c = s->creq;
 553|
 554|        c->status = dcerpc_pipe_connect_b_recv(req, s, &s->drsuapi.pipe);
 555|        if (!composite_is_ok(c)) return;
 556|
 557|        unbecomeDC_drsuapi_bind_send(s);
 558|}
 559|
 560|static void unbecomeDC_drsuapi_bind_recv(struct rpc_request *req);
 561|
 562|static void unbecomeDC_drsuapi_bind_send(struct libnet_UnbecomeDC_state *s)
 563|{
 564|        struct composite_context *c = s->creq;
 565|        struct rpc_request *req;
 566|        struct drsuapi_DsBindInfo28 *bind_info28;
 567|
 568|        GUID_from_string(DRSUAPI_DS_BIND_GUID, &s->drsuapi.bind_guid);
 569|
 570|        bind_info28                              = &s->drsuapi.local_info28;
 571|        bind_info28->supported_extensions       = 0;
 572|        bind_info28->site_guid                  = GUID_zero();
 573|        bind_info28->u1                          = 508;
 574|        bind_info28->repl_epoch                 = 0;
 575|
 576|        s->drsuapi.bind_info_ctr.length         = 28;
 577|        s->drsuapi.bind_info_ctr.info.info28    = *bind_info28;
 578|
 579|        s->drsuapi.bind_r.in.bind_guid = &s->drsuapi.bind_guid;
 580|        s->drsuapi.bind_r.in.bind_info = &s->drsuapi.bind_info_ctr;
 581|        s->drsuapi.bind_r.out.bind_handle = &s->drsuapi.bind_handle;
 582|
 583|        req = dcerpc_drsuapi_DsBind_send(s->drsuapi.pipe, s, &s->drsuapi.bind_r);
 584|        composite_continue_rpc(c, req, unbecomeDC_drsuapi_bind_recv, s);
 585|}
 586|
 587|static void unbecomeDC_drsuapi_remove_ds_server_send(struct libnet_UnbecomeDC_state *s);
 588|
```

```
 589|static void unbecomeDC_drsuapi_bind_recv(struct rpc_request *req)
 590|{
 591|        struct libnet_UnbecomeDC_state *s = talloc_get_type(req->async.private,
 592|                                           struct libnet_UnbecomeDC_state);
 593|        struct composite_context *c = s->creq;
 594|
 595|        c->status = dcerpc_ndr_request_recv(req);
 596|        if (!composite_is_ok(c)) return;
 597|
 598|        if (!W_ERROR_IS_OK(s->drsuapi.bind_r.out.result)) {
 599|                composite_error(c, werror_to_ntstatus(s->drsuapi.bind_r.out.result));
 600|                return;
 601|        }
 602|
 603|        ZERO_STRUCT(s->drsuapi.remote_info28);
 604|        if (s->drsuapi.bind_r.out.bind_info) {
 605|                switch (s->drsuapi.bind_r.out.bind_info->length) {
 606|                case 24: {
 607|                        struct drsuapi_DsBindInfo24 *info24;
 608|                        info24 = &s->drsuapi.bind_r.out.bind_info->info.info24;
 609|                        s->drsuapi.remote_info28.supported_extensions   =
->    info24->supported_extensions;
 610|                        s->drsuapi.remote_info28.site_guid              =
->    info24->site_guid;
 611|                        s->drsuapi.remote_info28.u1                     = info24->u1;
 612|                        s->drsuapi.remote_info28.repl_epoch             = 0;
 613|                        break;
 614|                }
 615|                case 28:
 616|                        s->drsuapi.remote_info28 = s->drsuapi.bind_r.out.bind_info->info.
->   info28;
 617|                        break;
 618|                }
 619|        }
 620|
 621|        unbecomeDC_drsuapi_remove_ds_server_send(s);
 622|}
 623|
 624|static void unbecomeDC_drsuapi_remove_ds_server_recv(struct rpc_request *req);
 625|
 626|static void unbecomeDC_drsuapi_remove_ds_server_send(struct libnet_UnbecomeDC_state *s)
 627|{
 628|        struct composite_context *c = s->creq;
 629|        struct rpc_request *req;
 630|        struct drsuapi_DsRemoveDSServer *r = &s->drsuapi.rm_ds_srv_r;
 631|
 632|        r->in.bind_handle       = &s->drsuapi.bind_handle;
 633|        r->in.level             = 1;
 634|        r->in.req.req1.server_dn= s->dest_dsa.server_dn_str;
 635|        r->in.req.req1.domain_dn= s->domain.dn_str;
 636|        r->in.req.req1.unknown  = 0x00000001;
 637|
 638|        req = dcerpc_drsuapi_DsRemoveDSServer_send(s->drsuapi.pipe, s, r);
 639|        composite_continue_rpc(c, req, unbecomeDC_drsuapi_remove_ds_server_recv, s);
 640|}
 641|
 642|static void unbecomeDC_drsuapi_remove_ds_server_recv(struct rpc_request *req)
 643|{
 644|        struct libnet_UnbecomeDC_state *s = talloc_get_type(req->async.private,
 645|                                           struct libnet_UnbecomeDC_state);
 646|        struct composite_context *c = s->creq;
 647|        struct drsuapi_DsRemoveDSServer *r = &s->drsuapi.rm_ds_srv_r;
 648|
 649|        c->status = dcerpc_ndr_request_recv(req);
 650|        if (!composite_is_ok(c)) return;
 651|
 652|        if (!W_ERROR_IS_OK(r->out.result)) {
 653|                composite_error(c, werror_to_ntstatus(r->out.result));
 654|                return;
 655|        }
 656|
 657|        if (r->out.level != 1) {
 658|                composite_error(c, NT_STATUS_INVALID_NETWORK_RESPONSE);
 659|                return;
 660|        }
 661|
 662|        if (!W_ERROR_IS_OK(r->out.res.res1.status)) {
 663|                composite_error(c, werror_to_ntstatus(r->out.res.res1.status));
 664|                return;
 665|        }
 666|
 667|        composite_done(c);
 668|}
 669|
 670|struct composite_context *libnet_UnbecomeDC_send(struct libnet_context *ctx, TALLOC_CTX
->    *mem_ctx, struct libnet_UnbecomeDC *r)
 671|{
```

```
672|        struct composite_context *c;
673|        struct libnet_UnbecomeDC_state *s;
674|        char *tmp_name;
675|
676|        c = composite_create(mem_ctx, ctx->event_ctx);
677|        if (c == NULL) return NULL;
678|
679|        s = talloc_zero(c, struct libnet_UnbecomeDC_state);
680|        if (composite_nomem(s, c)) return c;
681|        c->private_data = s;
682|        s->creq         = c;
683|        s->libnet       = ctx;
684|
685|        /* Domain input */
686|        s->domain.dns_name      = talloc_strdup(s, r->in.domain_dns_name);
687|        if (composite_nomem(s->domain.dns_name, c)) return c;
688|        s->domain.netbios_name  = talloc_strdup(s, r->in.domain_netbios_name);
689|        if (composite_nomem(s->domain.netbios_name, c)) return c;
690|
691|        /* Source DSA input */
692|        s->source_dsa.address   = talloc_strdup(s, r->in.source_dsa_address);
693|        if (composite_nomem(s->source_dsa.address, c)) return c;
694|
695|        /* Destination DSA input */
696|        s->dest_dsa.netbios_name= talloc_strdup(s, r->in.dest_dsa_netbios_name);
697|        if (composite_nomem(s->dest_dsa.netbios_name, c)) return c;
698|
699|        /* Destination DSA dns_name construction */
700|        tmp_name                = strlower_talloc(s, s->dest_dsa.netbios_name);
701|        if (composite_nomem(tmp_name, c)) return c;
702|        s->dest_dsa.dns_name    = talloc_asprintf_append(tmp_name, ".%s",
703|                                                         s->domain.dns_name);
704|        if (composite_nomem(s->dest_dsa.dns_name, c)) return c;
705|
706|        unbecomeDC_send_cldap(s);
707|        return c;
708|}
709|
710|NTSTATUS libnet_UnbecomeDC_recv(struct composite_context *c, TALLOC_CTX *mem_ctx, struct
->    libnet_UnbecomeDC *r)
711|{
712|        NTSTATUS status;
713|
714|        status = composite_wait(c);
715|
716|        ZERO_STRUCT(r->out);
717|
718|        talloc_free(c);
719|        return status;
720|}
721|
722|NTSTATUS libnet_UnbecomeDC(struct libnet_context *ctx, TALLOC_CTX *mem_ctx, struct
->    libnet_UnbecomeDC *r)
723|{
724|        NTSTATUS status;
725|        struct composite_context *c;
726|        c = libnet_UnbecomeDC_send(ctx, mem_ctx, r);
727|        status = libnet_UnbecomeDC_recv(c, mem_ctx, r);
728|        return status;
729|}
```

A10. source/dsdb/schema/schema_init.c

source/dsdb/schema/schema_init.c:

```
   1|/*
   2|   Unix SMB/CIFS mplementation.
   3|   DSDB schema header
   4|
   5|   Copyright (C) Stefan Metzmacher <metze@samba.org> 2006
   6|
   7|   This program is free software; you can redistribute it and/or modify
   8|   it under the terms of the GNU General Public License as published by
   9|   the Free Software Foundation; either version 2 of the License, or
  10|   (at your option) any later version.
  11|
  12|   This program is distributed in the hope that it will be useful,
  13|   but WITHOUT ANY WARRANTY; without even the implied warranty of
  14|   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  15|   GNU General Public License for more details.
  16|
  17|   You should have received a copy of the GNU General Public License
  18|   along with this program; if not, write to the Free Software
  19|   Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
  20|
  21|*/
  22|
  23|#include "includes.h"
  24|#include "dsdb/samdb/samdb.h"
  25|#include "lib/ldb/include/ldb_errors.h"
  26|#include "lib/util/dlinklist.h"
  27|#include "librpc/gen_ndr/ndr_misc.h"
  28|#include "librpc/gen_ndr/ndr_drsuapi.h"
  29|#include "librpc/gen_ndr/ndr_drsblobs.h"
  30|
  31|WERROR dsdb_load_oid_mappings_drsuapi(struct dsdb_schema *schema, const struct
->    drsuapi_DsReplicaOIDMapping_Ctr *ctr)
  32|{
  33|        uint32_t i,j;
  34|
  35|        schema->prefixes = talloc_array(schema, struct dsdb_schema_oid_prefix,
->    ctr->num_mappings);
  36|        W_ERROR_HAVE_NO_MEMORY(schema->prefixes);
  37|
  38|        for (i=0, j=0; i < ctr->num_mappings; i++) {
  39|                if (ctr->mappings[i].oid.oid == NULL) {
  40|                        return WERR_INVALID_PARAM;
  41|                }
  42|
  43|                if (strncasecmp(ctr->mappings[i].oid.oid, "ff", 2) == 0) {
  44|                        if (ctr->mappings[i].id_prefix != 0) {
  45|                                return WERR_INVALID_PARAM;
  46|                        }
  47|
  48|                        /* the magic value should be in the last array member */
  49|                        if (i != (ctr->num_mappings - 1)) {
  50|                                return WERR_INVALID_PARAM;
  51|                        }
  52|
  53|                        if (ctr->mappings[i].oid.__ndr_size != 21) {
  54|                                return WERR_INVALID_PARAM;
  55|                        }
  56|
  57|                        schema->schema_info = talloc_strdup(schema,
->    ctr->mappings[i].oid.oid);
  58|                        W_ERROR_HAVE_NO_MEMORY(schema->schema_info);
  59|                } else {
  60|                        /* the last array member should contain the magic value not a oid
->    */
  61|                        if (i == (ctr->num_mappings - 1)) {
  62|                                return WERR_INVALID_PARAM;
  63|                        }
  64|
  65|                        schema->prefixes[j].id  = ctr->mappings[i].id_prefix<<16;
  66|                        schema->prefixes[j].oid = talloc_asprintf(schema->prefixes,
->    "%s.",
  67|
->    ctr->mappings[i].oid.oid);
  68|                        W_ERROR_HAVE_NO_MEMORY(schema->prefixes[j].oid);
  69|                        schema->prefixes[j].oid_len = strlen(schema->prefixes[j].oid);
  70|                        j++;
  71|                }
  72|        }
  73|
```

```
  74|        schema->num_prefixes = j;
  75|        return WERR_OK;
  76|}
  77|
  78|WERROR dsdb_load_oid_mappings_ldb(struct dsdb_schema *schema,
  79|                                  const struct ldb_val *prefixMap,
  80|                                  const struct ldb_val *schemaInfo)
  81|{
  82|        WERROR status;
  83|        NTSTATUS nt_status;
  84|        struct prefixMapBlob pfm;
  85|        char *schema_info;
  86|
  87|        nt_status = ndr_pull_struct_blob(prefixMap, schema, &pfm,
  88|                                           (ndr_pull_flags_fn_t)ndr_pull_prefixMapBlob);
  89|        if (!NT_STATUS_IS_OK(nt_status)) {
  90|                return ntstatus_to_werror(nt_status);
  91|        }
  92|
  93|        if (pfm.version != PREFIX_MAP_VERSION_DSDB) {
  94|                return WERR_FOOBAR;
  95|        }
  96|
  97|        if (schemaInfo->length != 21 && schemaInfo->data[0] == 0xFF) {
  98|                return WERR_FOOBAR;
  99|        }
 100|
 101|        /* append the schema info as last element */
 102|        pfm.ctr.dsdb.num_mappings++;
 103|        pfm.ctr.dsdb.mappings = talloc_realloc(schema, pfm.ctr.dsdb.mappings,
 104|                                                struct drsuapi_DsReplicaOIDMapping,
 105|                                                pfm.ctr.dsdb.num_mappings);
 106|        W_ERROR_HAVE_NO_MEMORY(pfm.ctr.dsdb.mappings);
 107|
 108|        schema_info = data_blob_hex_string(pfm.ctr.dsdb.mappings, schemaInfo);
 109|        W_ERROR_HAVE_NO_MEMORY(schema_info);
 110|
 111|        pfm.ctr.dsdb.mappings[pfm.ctr.dsdb.num_mappings - 1].id_prefix          = 0;
 112|        pfm.ctr.dsdb.mappings[pfm.ctr.dsdb.num_mappings - 1].oid.__ndr_size     =
->    schemaInfo->length;
 113|        pfm.ctr.dsdb.mappings[pfm.ctr.dsdb.num_mappings - 1].oid.oid            =
->    schema_info;
 114|
 115|        /* call the drsuapi version */
 116|        status = dsdb_load_oid_mappings_drsuapi(schema, &pfm.ctr.dsdb);
 117|        talloc_free(pfm.ctr.dsdb.mappings);
 118|        W_ERROR_NOT_OK_RETURN(status);
 119|
 120|        return WERR_OK;
 121|}
 122|
 123|WERROR dsdb_get_oid_mappings_drsuapi(const struct dsdb_schema *schema,
 124|                                     bool include_schema_info,
 125|                                     TALLOC_CTX *mem_ctx,
 126|                                     struct drsuapi_DsReplicaOIDMapping_Ctr **_ctr)
 127|{
 128|        struct drsuapi_DsReplicaOIDMapping_Ctr *ctr;
 129|        uint32_t i;
 130|
 131|        ctr = talloc(mem_ctx, struct drsuapi_DsReplicaOIDMapping_Ctr);
 132|        W_ERROR_HAVE_NO_MEMORY(ctr);
 133|
 134|        ctr->num_mappings       = schema->num_prefixes;
 135|        if (include_schema_info) ctr->num_mappings++;
 136|        ctr->mappings = talloc_array(schema, struct drsuapi_DsReplicaOIDMapping,
->    ctr->num_mappings);
 137|        W_ERROR_HAVE_NO_MEMORY(ctr->mappings);
 138|
 139|        for (i=0; i < schema->num_prefixes; i++) {
 140|                ctr->mappings[i].id_prefix      = schema->prefixes[i].id>>16;
 141|                ctr->mappings[i].oid.oid        = talloc_strndup(ctr->mappings,
 142|
->    schema->prefixes[i].oid,
 143|
->    schema->prefixes[i].oid_len - 1);
 144|                W_ERROR_HAVE_NO_MEMORY(ctr->mappings[i].oid.oid);
 145|        }
 146|
 147|        if (include_schema_info) {
 148|                ctr->mappings[i].id_prefix      = 0;
 149|                ctr->mappings[i].oid.oid        = talloc_strdup(ctr->mappings,
 150|                                                                schema->schema_info);
 151|                W_ERROR_HAVE_NO_MEMORY(ctr->mappings[i].oid.oid);
 152|        }
 153|
 154|        *_ctr = ctr;
 155|        return WERR_OK;
```

```
 156|}
 157|
 158|WERROR dsdb_get_oid_mappings_ldb(const struct dsdb_schema *schema,
 159|                                 TALLOC_CTX *mem_ctx,
 160|                                 struct ldb_val *prefixMap,
 161|                                 struct ldb_val *schemaInfo)
 162|{
 163|        WERROR status;
 164|        NTSTATUS nt_status;
 165|        struct drsuapi_DsReplicaOIDMapping_Ctr *ctr;
 166|        struct prefixMapBlob pfm;
 167|
 168|        status = dsdb_get_oid_mappings_drsuapi(schema, false, mem_ctx, &ctr);
 169|        W_ERROR_NOT_OK_RETURN(status);
 170|
 171|        pfm.version     = PREFIX_MAP_VERSION_DSDB;
 172|        pfm.ctr.dsdb    = *ctr;
 173|
 174|        nt_status = ndr_push_struct_blob(prefixMap, mem_ctx, &pfm,
 175|                                          (ndr_push_flags_fn_t)ndr_push_prefixMapBlob);
 176|        talloc_free(ctr);
 177|        if (!NT_STATUS_IS_OK(nt_status)) {
 178|                return ntstatus_to_werror(nt_status);
 179|        }
 180|
 181|        *schemaInfo = strhex_to_data_blob(schema->schema_info);
 182|        W_ERROR_HAVE_NO_MEMORY(schemaInfo->data);
 183|        talloc_steal(mem_ctx, schemaInfo->data);
 184|
 185|        return WERR_OK;
 186|}
 187|
 188|WERROR dsdb_verify_oid_mappings_drsuapi(const struct dsdb_schema *schema, const struct
->    drsuapi_DsReplicaOIDMapping_Ctr *ctr)
 189|{
 190|        uint32_t i,j;
 191|
 192|        for (i=0; i < ctr->num_mappings; i++) {
 193|                if (ctr->mappings[i].oid.oid == NULL) {
 194|                        return WERR_INVALID_PARAM;
 195|                }
 196|
 197|                if (strncasecmp(ctr->mappings[i].oid.oid, "ff", 2) == 0) {
 198|                        if (ctr->mappings[i].id_prefix != 0) {
 199|                                return WERR_INVALID_PARAM;
 200|                        }
 201|
 202|                        /* the magic value should be in the last array member */
 203|                        if (i != (ctr->num_mappings - 1)) {
 204|                                return WERR_INVALID_PARAM;
 205|                        }
 206|
 207|                        if (ctr->mappings[i].oid.__ndr_size != 21) {
 208|                                return WERR_INVALID_PARAM;
 209|                        }
 210|
 211|                        if (strcasecmp(schema->schema_info, ctr->mappings[i].oid.oid) !=
->    0) {
 212|                                return WERR_DS_DRA_SCHEMA_MISMATCH;
 213|                        }
 214|                } else {
 215|                        /* the last array member should contain the magic value not a oid
->    */
 216|                        if (i == (ctr->num_mappings - 1)) {
 217|                                return WERR_INVALID_PARAM;
 218|                        }
 219|
 220|                        for (j=0; j < schema->num_prefixes; j++) {
 221|                                size_t oid_len;
 222|                                if (schema->prefixes[j].id !=
->    (ctr->mappings[i].id_prefix<<16)) {
 223|                                        continue;
 224|                                }
 225|
 226|                                oid_len = strlen(ctr->mappings[i].oid.oid);
 227|
 228|                                if (oid_len != (schema->prefixes[j].oid_len - 1)) {
 229|                                        return WERR_DS_DRA_SCHEMA_MISMATCH;
 230|                                }
 231|
 232|                                if (strncmp(ctr->mappings[i].oid.oid,
->    schema->prefixes[j].oid, oid_len) != 0) {
 233|                                        return WERR_DS_DRA_SCHEMA_MISMATCH;
->
 234|                                }
 235|
 236|                                break;
```

```
 237|                        }
 238|
 239|                        if (j == schema->num_prefixes) {
 240|                                return WERR_DS_DRA_SCHEMA_MISMATCH;
->
 241|                        }
 242|                }
 243|        }
 244|
 245|        return WERR_OK;
 246|}
 247|
 248|WERROR dsdb_map_oid2int(const struct dsdb_schema *schema, const char *in, uint32_t *out)
 249|{
 250|        uint32_t i;
 251|
 252|        for (i=0; i < schema->num_prefixes; i++) {
 253|                const char *val_str;
 254|                char *end_str;
 255|                unsigned val;
 256|
 257|                if (strncmp(schema->prefixes[i].oid, in, schema->prefixes[i].oid_len) !=
->    0) {
 258|                        continue;
 259|                }
 260|
 261|                val_str = in + schema->prefixes[i].oid_len;
 262|                end_str = NULL;
 263|                errno = 0;
 264|
 265|                if (val_str[0] == '\0') {
 266|                        return WERR_INVALID_PARAM;
 267|                }
 268|
 269|                /* two '.' chars are invalid */
 270|                if (val_str[0] == '.') {
 271|                        return WERR_INVALID_PARAM;
 272|                }
 273|
 274|                val = strtoul(val_str, &end_str, 10);
 275|                if (end_str[0] == '.' && end_str[1] != '\0') {
 276|                        /*
 277|                         * if it's a '.' and not the last char
 278|                         * then maybe an other mapping apply
 279|                         */
 280|                        continue;
 281|                } else if (end_str[0] != '\0') {
 282|                        return WERR_INVALID_PARAM;
 283|                } else if (val > 0xFFFF) {
 284|                        return WERR_INVALID_PARAM;
 285|                }
 286|
 287|                *out = schema->prefixes[i].id | val;
 288|                return WERR_OK;
 289|        }
 290|
 291|        return WERR_DS_NO_MSDS_INTID;
 292|}
 293|
 294|WERROR dsdb_map_int2oid(const struct dsdb_schema *schema, uint32_t in, TALLOC_CTX
->    *mem_ctx, const char **out)
 295|{
 296|        uint32_t i;
 297|
 298|        for (i=0; i < schema->num_prefixes; i++) {
 299|                const char *val;
 300|                if (schema->prefixes[i].id != (in & 0xFFFF0000)) {
 301|                        continue;
 302|                }
 303|
 304|                val = talloc_asprintf(mem_ctx, "%s%u",
 305|                                      schema->prefixes[i].oid,
 306|                                      in & 0xFFFF);
 307|                W_ERROR_HAVE_NO_MEMORY(val);
 308|
 309|                *out = val;
 310|                return WERR_OK;
 311|        }
 312|
 313|        return WERR_DS_NO_MSDS_INTID;
 314|}
 315|
 316|#define GET_STRING_LDB(msg, attr, mem_ctx, p, elem, strict) do { \
 317|        (p)->elem = samdb_result_string(msg, attr, NULL);\
 318|        if (strict && (p)->elem == NULL) { \
 319|                d_printf("%s: %s == NULL\n", __location__, attr); \
 320|                return WERR_INVALID_PARAM; \
```

```
 321|        } \
 322|        talloc_steal(mem_ctx, (p)->elem); \
 323|} while (0)
 324|
 325|#define GET_BOOL_LDB(msg, attr, p, elem, strict) do { \
 326|        const char *str; \
 327|        str = samdb_result_string(msg, attr, NULL);\
 328|        if (str == NULL) { \
 329|                if (strict) { \
 330|                        d_printf("%s: %s == NULL\n", __location__, attr); \
 331|                        return WERR_INVALID_PARAM; \
 332|                } else { \
 333|                        (p)->elem = False; \
 334|                } \
 335|        } else if (strcasecmp("TRUE", str) == 0) { \
 336|                (p)->elem = True; \
 337|        } else if (strcasecmp("FALSE", str) == 0) { \
 338|                (p)->elem = False; \
 339|        } else { \
 340|                d_printf("%s: %s == %s\n", __location__, attr, str); \
 341|                return WERR_INVALID_PARAM; \
 342|        } \
 343|} while (0)
 344|
 345|#define GET_UINT32_LDB(msg, attr, p, elem) do { \
 346|        (p)->elem = samdb_result_uint(msg, attr, 0);\
 347|} while (0)
 348|
 349|#define GET_GUID_LDB(msg, attr, p, elem) do { \
 350|        (p)->elem = samdb_result_guid(msg, attr);\
 351|} while (0)
 352|
 353|#define GET_BLOB_LDB(msg, attr, mem_ctx, p, elem) do { \
 354|        const struct ldb_val *_val;\
 355|        _val = ldb_msg_find_ldb_val(msg, attr);\
 356|        if (_val) {\
 357|                (p)->elem = *_val;\
 358|                talloc_steal(mem_ctx, (p)->elem.data);\
 359|        } else {\
 360|                ZERO_STRUCT((p)->elem);\
 361|        }\
 362|} while (0)
 363|
 364|WERROR dsdb_attribute_from_ldb(const struct dsdb_schema *schema,
 365|                               struct ldb_message *msg,
 366|                               TALLOC_CTX *mem_ctx,
 367|                               struct dsdb_attribute *attr)
 368|{
 369|        WERROR status;
 370|
 371|        GET_STRING_LDB(msg, "cn", mem_ctx, attr, cn, True);
 372|        GET_STRING_LDB(msg, "lDAPDisplayName", mem_ctx, attr, lDAPDisplayName, True);
 373|        GET_STRING_LDB(msg, "attributeID", mem_ctx, attr, attributeID_oid, True);
 374|        if (schema->num_prefixes == 0) {
 375|                /* set an invalid value */
 376|                attr->attributeID_id = 0xFFFFFFFF;
 377|        } else {
 378|                status = dsdb_map_oid2int(schema, attr->attributeID_oid,
->    &attr->attributeID_id);
 379|                if (!W_ERROR_IS_OK(status)) {
 380|                        DEBUG(0,("%s: '%s': unable to map attributeID %s: %s\n",
 381|                                 __location__, attr->lDAPDisplayName,
->    attr->attributeID_oid,
 382|                                 win_errstr(status)));
 383|                        return status;
 384|                }
 385|        }
 386|        GET_GUID_LDB(msg, "schemaIDGUID", attr, schemaIDGUID);
 387|        GET_UINT32_LDB(msg, "mAPIID", attr, mAPIID);
 388|
 389|        GET_GUID_LDB(msg, "attributeSecurityGUID", attr, attributeSecurityGUID);
 390|
 391|        GET_UINT32_LDB(msg, "searchFlags", attr, searchFlags);
 392|        GET_UINT32_LDB(msg, "systemFlags", attr, systemFlags);
 393|        GET_BOOL_LDB(msg, "isMemberOfPartialAttributeSet", attr,
->    isMemberOfPartialAttributeSet, False);
 394|        GET_UINT32_LDB(msg, "linkID", attr, linkID);
 395|
 396|        GET_STRING_LDB(msg, "attributeSyntax", mem_ctx, attr, attributeSyntax_oid,
->    True);
 397|        if (schema->num_prefixes == 0) {
 398|                /* set an invalid value */
 399|                attr->attributeSyntax_id = 0xFFFFFFFF;
 400|        } else {
 401|                status = dsdb_map_oid2int(schema, attr->attributeSyntax_oid,
->    &attr->attributeSyntax_id);
 402|                if (!W_ERROR_IS_OK(status)) {
```

```
                        DEBUG(0,("%s: '%s': unable to map attributeSyntax_ %s: %s\n",
                                 __location__, attr->lDAPDisplayName,
->    attr->attributeSyntax_oid,
                                 win_errstr(status)));
                        return status;
                }
        }
        GET_UINT32_LDB(msg, "oMSyntax", attr, oMSyntax);
        GET_BLOB_LDB(msg, "oMObjectClass", mem_ctx, attr, oMObjectClass);

        GET_BOOL_LDB(msg, "isSingleValued", attr, isSingleValued, True);
        GET_UINT32_LDB(msg, "rangeLower", attr, rangeLower);
        GET_UINT32_LDB(msg, "rangeUpper", attr, rangeUpper);
        GET_BOOL_LDB(msg, "extendedCharsAllowed", attr, extendedCharsAllowed, False);

        GET_UINT32_LDB(msg, "schemaFlagsEx", attr, schemaFlagsEx);
        GET_BLOB_LDB(msg, "msDs-Schema-Extensions", mem_ctx, attr,
->    msDs_Schema_Extensions);

        GET_BOOL_LDB(msg, "showInAdvancedViewOnly", attr, showInAdvancedViewOnly,
->    False);
        GET_STRING_LDB(msg, "adminDisplayName", mem_ctx, attr, adminDisplayName, False);
        GET_STRING_LDB(msg, "adminDescription", mem_ctx, attr, adminDescription, False);
        GET_STRING_LDB(msg, "classDisplayName", mem_ctx, attr, classDisplayName, False);
        GET_BOOL_LDB(msg, "isEphemeral", attr, isEphemeral, False);
        GET_BOOL_LDB(msg, "isDefunct", attr, isDefunct, False);
        GET_BOOL_LDB(msg, "systemOnly", attr, systemOnly, False);

        attr->syntax = dsdb_syntax_for_attribute(attr);
        if (!attr->syntax) {
                return WERR_DS_ATT_SCHEMA_REQ_SYNTAX;
        }

        return WERR_OK;
}

WERROR dsdb_class_from_ldb(const struct dsdb_schema *schema,
                           struct ldb_message *msg,
                           TALLOC_CTX *mem_ctx,
                           struct dsdb_class *obj)
{
        WERROR status;

        GET_STRING_LDB(msg, "cn", mem_ctx, obj, cn, True);
        GET_STRING_LDB(msg, "lDAPDisplayName", mem_ctx, obj, lDAPDisplayName, True);
        GET_STRING_LDB(msg, "governsID", mem_ctx, obj, governsID_oid, True);
        if (schema->num_prefixes == 0) {
                /* set an invalid value */
                obj->governsID_id = 0xFFFFFFFF;
        } else {
                status = dsdb_map_oid2int(schema, obj->governsID_oid,
->    &obj->governsID_id);
                if (!W_ERROR_IS_OK(status)) {
                        DEBUG(0,("%s: '%s': unable to map governsID %s: %s\n",
                                 __location__, obj->lDAPDisplayName, obj->governsID_oid,
                                 win_errstr(status)));
                        return status;
                }
        }
        GET_GUID_LDB(msg, "schemaIDGUID", obj, schemaIDGUID);

        GET_UINT32_LDB(msg, "objectClassCategory", obj, objectClassCategory);
        GET_STRING_LDB(msg, "rDNAttID", mem_ctx, obj, rDNAttID, False);
        GET_STRING_LDB(msg, "defaultObjectCategory", mem_ctx, obj, defaultObjectCategory,
->    True);

        GET_STRING_LDB(msg, "subClassOf", mem_ctx, obj, subClassOf, True);

        obj->systemAuxiliaryClass       = NULL;
        obj->systemPossSuperiors        = NULL;
        obj->systemMustContain          = NULL;
        obj->systemMayContain           = NULL;

        obj->auxiliaryClass             = NULL;
        obj->possSuperiors              = NULL;
        obj->mustContain                = NULL;
        obj->mayContain                 = NULL;

        GET_STRING_LDB(msg, "defaultSecurityDescriptor", mem_ctx, obj,
->    defaultSecurityDescriptor, False);

        GET_UINT32_LDB(msg, "schemaFlagsEx", obj, schemaFlagsEx);
        GET_BLOB_LDB(msg, "msDs-Schema-Extensions", mem_ctx, obj,
->    msDs_Schema_Extensions);

        GET_BOOL_LDB(msg, "showInAdvancedViewOnly", obj, showInAdvancedViewOnly, False);
        GET_STRING_LDB(msg, "adminDisplayName", mem_ctx, obj, adminDisplayName, False);
```

```
483|        GET_STRING_LDB(msg, "adminDescription", mem_ctx, obj, adminDescription, False);
484|        GET_STRING_LDB(msg, "classDisplayName", mem_ctx, obj, classDisplayName, False);
485|        GET_BOOL_LDB(msg, "defaultHidingValue", obj, defaultHidingValue, False);
486|        GET_BOOL_LDB(msg, "isDefunct", obj, isDefunct, False);
487|        GET_BOOL_LDB(msg, "systemOnly", obj, systemOnly, False);
488|
489|        return WERR_OK;
490|}
491|
492|static const struct {
493|        const char *name;
494|        const char *oid;
495|} name_mappings[] = {
496|        { "cn",                                  "2.5.4.3" },
497|        { "name",                                "1.2.840.113556.1.4.1" },
498|        { "lDAPDisplayName",                     "1.2.840.113556.1.2.460" },
499|        { "attributeID",                         "1.2.840.113556.1.2.30" },
500|        { "schemaIDGUID",                       "1.2.840.113556.1.4.148" },
501|        { "mAPIID",                             "1.2.840.113556.1.2.49" },
502|        { "attributeSecurityGUID",              "1.2.840.113556.1.4.149" },
503|        { "searchFlags",                        "1.2.840.113556.1.2.334" },
504|        { "systemFlags",                        "1.2.840.113556.1.4.375" },
505|        { "isMemberOfPartialAttributeSet",      "1.2.840.113556.1.4.639" },
506|        { "linkID",                            "1.2.840.113556.1.2.50" },
507|        { "attributeSyntax",                   "1.2.840.113556.1.2.32" },
508|        { "oMSyntax",                          "1.2.840.113556.1.2.231" },
509|        { "oMObjectClass",                     "1.2.840.113556.1.2.218" },
510|        { "isSingleValued",                    "1.2.840.113556.1.2.33" },
511|        { "rangeLower",                        "1.2.840.113556.1.2.34" },
512|        { "rangeUpper",                        "1.2.840.113556.1.2.35" },
513|        { "extendedCharsAllowed",              "1.2.840.113556.1.2.380" },
514|        { "schemaFlagsEx",                     "1.2.840.113556.1.4.120" },
515|        { "msDs-Schema-Extensions",           "1.2.840.113556.1.4.1440" },
516|        { "showInAdvancedViewOnly",           "1.2.840.113556.1.2.169" },
517|        { "adminDisplayName",                  "1.2.840.113556.1.2.194" },
518|        { "adminDescription",                  "1.2.840.113556.1.2.226" },
519|        { "classDisplayName",                  "1.2.840.113556.1.4.610" },
520|        { "isEphemeral",                       "1.2.840.113556.1.4.1212" },
521|        { "isDefunct",                        "1.2.840.113556.1.4.661" },
522|        { "systemOnly",                       "1.2.840.113556.1.4.170" },
523|        { "governsID",                        "1.2.840.113556.1.2.22" },
524|        { "objectClassCategory",              "1.2.840.113556.1.2.370" },
525|        { "rDNAttID",                         "1.2.840.113556.1.2.26" },
526|        { "defaultObjectCategory",            "1.2.840.113556.1.4.783" },
527|        { "subClassOf",                       "1.2.840.113556.1.2.21" },
528|        { "systemAuxiliaryClass",             "1.2.840.113556.1.4.198" },
529|        { "systemPossSuperiors",              "1.2.840.113556.1.4.195" },
530|        { "systemMustContain",                "1.2.840.113556.1.4.197" },
531|        { "systemMayContain",                 "1.2.840.113556.1.4.196" },
532|        { "auxiliaryClass",                   "1.2.840.113556.1.2.351" },
533|        { "possSuperiors",                    "1.2.840.113556.1.2.8" },
534|        { "mustContain",                      "1.2.840.113556.1.2.24" },
535|        { "mayContain",                       "1.2.840.113556.1.2.25" },
536|        { "defaultSecurityDescriptor",        "1.2.840.113556.1.4.224" },
537|        { "defaultHidingValue",               "1.2.840.113556.1.4.518" },
538|};
539|
540|static struct drsuapi_DsReplicaAttribute *dsdb_find_object_attr_name(struct dsdb_schema
->    *schema,
541|                                                                     struct
->    drsuapi_DsReplicaObject *obj,
542|                                                                     const char *name,
543|                                                                     uint32_t *idx)
544|{
545|        WERROR status;
546|        uint32_t i, id;
547|        const char *oid = NULL;
548|
549|        for(i=0; i < ARRAY_SIZE(name_mappings); i++) {
550|                if (strcmp(name_mappings[i].name, name) != 0) continue;
551|
552|                oid = name_mappings[i].oid;
553|                break;
554|        }
555|
556|        if (!oid) {
557|                return NULL;
558|        }
559|
560|        status = dsdb_map_oid2int(schema, oid, &id);
561|        if (!W_ERROR_IS_OK(status)) {
562|                return NULL;
563|        }
564|
565|        for (i=0; i < obj->attribute_ctr.num_attributes; i++) {
566|                if (obj->attribute_ctr.attributes[i].attid != id) continue;
567|
```

```
568|                if (idx) *idx = i;
569|                return &obj->attribute_ctr.attributes[i];
570|        }
571|
572|        return NULL;
573|}
574|
575|#define GET_STRING_DS(s, r, attr, mem_ctx, p, elem, strict) do { \
576|        struct drsuapi_DsReplicaAttribute *_a; \
577|        _a = dsdb_find_object_attr_name(s, r, attr, NULL); \
578|        if (strict && !_a) { \
579|                d_printf("%s: %s == NULL\n", __location__, attr); \
580|                return WERR_INVALID_PARAM; \
581|        } \
582|        if (strict && _a->value_ctr.num_values != 1) { \
583|                d_printf("%s: %s num_values == %u\n", __location__, attr, \
584|                         _a->value_ctr.num_values); \
585|                return WERR_INVALID_PARAM; \
586|        } \
587|        if (_a && _a->value_ctr.num_values >= 1) { \
588|                ssize_t _ret; \
589|                _ret = convert_string_talloc(mem_ctx, CH_UTF16, CH_UNIX, \
590|                                             _a->value_ctr.values[0].blob->data, \
591|                                             _a->value_ctr.values[0].blob->length, \
592|                                             (void **)discard_const(&(p)->elem)); \
593|                if (_ret == -1) { \
594|                        DEBUG(0,("%s: invalid data!\n", attr)); \
595|                        dump_data(0, \
596|                                  _a->value_ctr.values[0].blob->data, \
597|                                  _a->value_ctr.values[0].blob->length); \
598|                        return WERR_FOOBAR; \
599|                } \
600|        } else { \
601|                (p)->elem = NULL; \
602|        } \
603|} while (0)
604|
605|#define GET_DN_DS(s, r, attr, mem_ctx, p, elem, strict) do { \
606|        struct drsuapi_DsReplicaAttribute *_a; \
607|        _a = dsdb_find_object_attr_name(s, r, attr, NULL); \
608|        if (strict && !_a) { \
609|                d_printf("%s: %s == NULL\n", __location__, attr); \
610|                return WERR_INVALID_PARAM; \
611|        } \
612|        if (strict && _a->value_ctr.num_values != 1) { \
613|                d_printf("%s: %s num_values == %u\n", __location__, attr, \
614|                         _a->value_ctr.num_values); \
615|                return WERR_INVALID_PARAM; \
616|        } \
617|        if (strict && !_a->value_ctr.values[0].blob) { \
618|                d_printf("%s: %s data == NULL\n", __location__, attr); \
619|                return WERR_INVALID_PARAM; \
620|        } \
621|        if (_a && _a->value_ctr.num_values >= 1 \
622|            && _a->value_ctr.values[0].blob) { \
623|                struct drsuapi_DsReplicaObjectIdentifier3 _id3; \
624|                NTSTATUS _nt_status; \
625|                _nt_status = ndr_pull_struct_blob_all(_a->value_ctr.values[0].blob, \
626|                                                      mem_ctx, &_id3,\
627|                                                      (ndr_pull_flags_fn_t)ndr_pull_drsua
->   pi_DsReplicaObjectIdentifier3);\
628|                if (!NT_STATUS_IS_OK(_nt_status)) { \
629|                        return ntstatus_to_werror(_nt_status); \
630|                } \
631|                (p)->elem = _id3.dn; \
632|        } else { \
633|                (p)->elem = NULL; \
634|        } \
635|} while (0)
636|
637|#define GET_BOOL_DS(s, r, attr, p, elem, strict) do { \
638|        struct drsuapi_DsReplicaAttribute *_a; \
639|        _a = dsdb_find_object_attr_name(s, r, attr, NULL); \
640|        if (strict && !_a) { \
641|                d_printf("%s: %s == NULL\n", __location__, attr); \
642|                return WERR_INVALID_PARAM; \
643|        } \
644|        if (strict && _a->value_ctr.num_values != 1) { \
645|                d_printf("%s: %s num_values == %u\n", __location__, attr, \
646|                         _a->value_ctr.num_values); \
647|                return WERR_INVALID_PARAM; \
648|        } \
649|        if (strict && !_a->value_ctr.values[0].blob) { \
650|                d_printf("%s: %s data == NULL\n", __location__, attr); \
651|                return WERR_INVALID_PARAM; \
652|        } \
653|        if (strict && _a->value_ctr.values[0].blob->length != 4) { \
```

```
654|                d_printf("%s: %s length == %u\n", __location__, attr, \
655|                        _a->value_ctr.values[0].blob->length); \
656|                return WERR_INVALID_PARAM; \
657|        } \
658|        if (_a && _a->value_ctr.num_values >= 1 \
659|            && _a->value_ctr.values[0].blob \
660|            && _a->value_ctr.values[0].blob->length == 4) { \
661|                (p)->elem = (IVAL(_a->value_ctr.values[0].blob->data,0)?True:False);\
662|        } else { \
663|                (p)->elem = False; \
664|        } \
665|} while (0)
666|
667|#define GET_UINT32_DS(s, r, attr, p, elem) do { \
668|        struct drsuapi_DsReplicaAttribute *_a; \
669|        _a = dsdb_find_object_attr_name(s, r, attr, NULL); \
670|        if (_a && _a->value_ctr.num_values >= 1 \
671|            && _a->value_ctr.values[0].blob \
672|            && _a->value_ctr.values[0].blob->length == 4) { \
673|                (p)->elem = IVAL(_a->value_ctr.values[0].blob->data,0);\
674|        } else { \
675|                (p)->elem = 0; \
676|        } \
677|} while (0)
678|
679|#define GET_GUID_DS(s, r, attr, mem_ctx, p, elem) do { \
680|        struct drsuapi_DsReplicaAttribute *_a; \
681|        _a = dsdb_find_object_attr_name(s, r, attr, NULL); \
682|        if (_a && _a->value_ctr.num_values >= 1 \
683|            && _a->value_ctr.values[0].blob \
684|            && _a->value_ctr.values[0].blob->length == 16) { \
685|                NTSTATUS _nt_status; \
686|                _nt_status = ndr_pull_struct_blob_all(_a->value_ctr.values[0].blob, \
687|                                                      mem_ctx, &(p)->elem, \
688|                                                      (ndr_pull_flags_fn_t)ndr_pull_GUID)
->   ; \
689|                if (!NT_STATUS_IS_OK(_nt_status)) { \
690|                        return ntstatus_to_werror(_nt_status); \
691|                } \
692|        } else { \
693|                ZERO_STRUCT((p)->elem);\
694|        } \
695|} while (0)
696|
697|#define GET_BLOB_DS(s, r, attr, mem_ctx, p, elem) do { \
698|        struct drsuapi_DsReplicaAttribute *_a; \
699|        _a = dsdb_find_object_attr_name(s, r, attr, NULL); \
700|        if (_a && _a->value_ctr.num_values >= 1 \
701|            && _a->value_ctr.values[0].blob) { \
702|                (p)->elem = *_a->value_ctr.values[0].blob;\
703|                talloc_steal(mem_ctx, (p)->elem.data); \
704|        } else { \
705|                ZERO_STRUCT((p)->elem);\
706|        }\
707|} while (0)
708|
709|WERROR dsdb_attribute_from_drsuapi(struct dsdb_schema *schema,
710|                                   struct drsuapi_DsReplicaObject *r,
711|                                   TALLOC_CTX *mem_ctx,
712|                                   struct dsdb_attribute *attr)
713|{
714|        WERROR status;
715|
716|        GET_STRING_DS(schema, r, "name", mem_ctx, attr, cn, True);
717|        GET_STRING_DS(schema, r, "lDAPDisplayName", mem_ctx, attr, lDAPDisplayName,
->    True);
718|        GET_UINT32_DS(schema, r, "attributeID", attr, attributeID_id);
719|        status = dsdb_map_int2oid(schema, attr->attributeID_id, mem_ctx,
->    &attr->attributeID_oid);
720|        if (!W_ERROR_IS_OK(status)) {
721|                DEBUG(0,("%s: '%s': unable to map attributeID 0x%08X: %s\n",
722|                        __location__, attr->lDAPDisplayName, attr->attributeID_id,
723|                        win_errstr(status)));
724|                return status;
725|        }
726|        GET_GUID_DS(schema, r, "schemaIDGUID", mem_ctx, attr, schemaIDGUID);
727|        GET_UINT32_DS(schema, r, "mAPIID", attr, mAPIID);
728|
729|        GET_GUID_DS(schema, r, "attributeSecurityGUID", mem_ctx, attr,
->    attributeSecurityGUID);
730|
731|        GET_UINT32_DS(schema, r, "searchFlags", attr, searchFlags);
732|        GET_UINT32_DS(schema, r, "systemFlags", attr, systemFlags);
733|        GET_BOOL_DS(schema, r, "isMemberOfPartialAttributeSet", attr,
->    isMemberOfPartialAttributeSet, False);
734|        GET_UINT32_DS(schema, r, "linkID", attr, linkID);
735|
```

```
 736|        GET_UINT32_DS(schema, r, "attributeSyntax", attr, attributeSyntax_id);
 737|        status = dsdb_map_int2oid(schema, attr->attributeSyntax_id, mem_ctx,
->    &attr->attributeSyntax_oid);
 738|        if (!W_ERROR_IS_OK(status)) {
 739|                DEBUG(0,("%s: '%s': unable to map attributeSyntax 0x%08X: %s\n",
 740|                        __location__, attr->lDAPDisplayName, attr->attributeSyntax_id,
 741|                        win_errstr(status)));
 742|                return status;
 743|        }
 744|        GET_UINT32_DS(schema, r, "oMSyntax", attr, oMSyntax);
 745|        GET_BLOB_DS(schema, r, "oMObjectClass", mem_ctx, attr, oMObjectClass);
 746|
 747|        GET_BOOL_DS(schema, r, "isSingleValued", attr, isSingleValued, True);
 748|        GET_UINT32_DS(schema, r, "rangeLower", attr, rangeLower);
 749|        GET_UINT32_DS(schema, r, "rangeUpper", attr, rangeUpper);
 750|        GET_BOOL_DS(schema, r, "extendedCharsAllowed", attr, extendedCharsAllowed,
->    False);
 751|
 752|        GET_UINT32_DS(schema, r, "schemaFlagsEx", attr, schemaFlagsEx);
 753|        GET_BLOB_DS(schema, r, "msDs-Schema-Extensions", mem_ctx, attr,
->    msDs_Schema_Extensions);
 754|
 755|        GET_BOOL_DS(schema, r, "showInAdvancedViewOnly", attr, showInAdvancedViewOnly,
->    False);
 756|        GET_STRING_DS(schema, r, "adminDisplayName", mem_ctx, attr, adminDisplayName,
->    False);
 757|        GET_STRING_DS(schema, r, "adminDescription", mem_ctx, attr, adminDescription,
->    False);
 758|        GET_STRING_DS(schema, r, "classDisplayName", mem_ctx, attr, classDisplayName,
->    False);
 759|        GET_BOOL_DS(schema, r, "isEphemeral", attr, isEphemeral, False);
 760|        GET_BOOL_DS(schema, r, "isDefunct", attr, isDefunct, False);
 761|        GET_BOOL_DS(schema, r, "systemOnly", attr, systemOnly, False);
 762|
 763|        attr->syntax = dsdb_syntax_for_attribute(attr);
 764|        if (!attr->syntax) {
 765|                return WERR_DS_ATT_SCHEMA_REQ_SYNTAX;
 766|        }
 767|
 768|        return WERR_OK;
 769|}
 770|
 771|WERROR dsdb_class_from_drsuapi(struct dsdb_schema *schema,
 772|                               struct drsuapi_DsReplicaObject *r,
 773|                               TALLOC_CTX *mem_ctx,
 774|                               struct dsdb_class *obj)
 775|{
 776|        WERROR status;
 777|
 778|        GET_STRING_DS(schema, r, "name", mem_ctx, obj, cn, True);
 779|        GET_STRING_DS(schema, r, "lDAPDisplayName", mem_ctx, obj, lDAPDisplayName,
->    True);
 780|        GET_UINT32_DS(schema, r, "governsID", obj, governsID_id);
 781|        status = dsdb_map_int2oid(schema, obj->governsID_id, mem_ctx,
->    &obj->governsID_oid);
 782|        if (!W_ERROR_IS_OK(status)) {
 783|                DEBUG(0,("%s: '%s': unable to map governsID 0x%08X: %s\n",
 784|                        __location__, obj->lDAPDisplayName, obj->governsID_id,
 785|                        win_errstr(status)));
 786|                return status;
 787|        }
 788|        GET_GUID_DS(schema, r, "schemaIDGUID", mem_ctx, obj, schemaIDGUID);
 789|
 790|        GET_UINT32_DS(schema, r, "objectClassCategory", obj, objectClassCategory);
 791|        GET_STRING_DS(schema, r, "rDNAttID", mem_ctx, obj, rDNAttID, False);
 792|        GET_DN_DS(schema, r, "defaultObjectCategory", mem_ctx, obj,
->    defaultObjectCategory, True);
 793|
 794|        GET_STRING_DS(schema, r, "subClassOf", mem_ctx, obj, subClassOf, True);
 795|
 796|        obj->systemAuxiliaryClass       = NULL;
 797|        obj->systemPossSuperiors        = NULL;
 798|        obj->systemMustContain          = NULL;
 799|        obj->systemMayContain           = NULL;
 800|
 801|        obj->auxiliaryClass             = NULL;
 802|        obj->possSuperiors              = NULL;
 803|        obj->mustContain                = NULL;
 804|        obj->mayContain                 = NULL;
 805|
 806|        GET_STRING_DS(schema, r, "defaultSecurityDescriptor", mem_ctx, obj,
->    defaultSecurityDescriptor, False);
 807|
 808|        GET_UINT32_DS(schema, r, "schemaFlagsEx", obj, schemaFlagsEx);
 809|        GET_BLOB_DS(schema, r, "msDs-Schema-Extensions", mem_ctx, obj,
->    msDs_Schema_Extensions);
 810|
```

```
 811|        GET_BOOL_DS(schema, r, "showInAdvancedViewOnly", obj, showInAdvancedViewOnly,
->    False);
 812|        GET_STRING_DS(schema, r, "adminDisplayName", mem_ctx, obj, adminDisplayName,
->    False);
 813|        GET_STRING_DS(schema, r, "adminDescription", mem_ctx, obj, adminDescription,
->    False);
 814|        GET_STRING_DS(schema, r, "classDisplayName", mem_ctx, obj, classDisplayName,
->    False);
 815|        GET_BOOL_DS(schema, r, "defaultHidingValue", obj, defaultHidingValue, False);
 816|        GET_BOOL_DS(schema, r, "isDefunct", obj, isDefunct, False);
 817|        GET_BOOL_DS(schema, r, "systemOnly", obj, systemOnly, False);
 818|
 819|        return WERR_OK;
 820|}
 821|
 822|const struct dsdb_attribute *dsdb_attribute_by_attributeID_id(const struct dsdb_schema
->    *schema,
 823|                                                              uint32_t id)
 824|{
 825|        struct dsdb_attribute *cur;
 826|
 827|        /*
 828|         * 0xFFFFFFFF is used as value when no mapping table is available,
 829|         * so don't try to match with it
 830|         */
 831|        if (id == 0xFFFFFFFF) return NULL;
 832|
 833|        /* TODO: add binary search */
 834|        for (cur = schema->attributes; cur; cur = cur->next) {
 835|                if (cur->attributeID_id != id) continue;
 836|
 837|                return cur;
 838|        }
 839|
 840|        return NULL;
 841|}
 842|
 843|const struct dsdb_attribute *dsdb_attribute_by_attributeID_oid(const struct dsdb_schema
->    *schema,
 844|                                                               const char *oid)
 845|{
 846|        struct dsdb_attribute *cur;
 847|
 848|        if (!oid) return NULL;
 849|
 850|        /* TODO: add binary search */
 851|        for (cur = schema->attributes; cur; cur = cur->next) {
 852|                if (strcmp(cur->attributeID_oid, oid) != 0) continue;
 853|
 854|                return cur;
 855|        }
 856|
 857|        return NULL;
 858|}
 859|
 860|const struct dsdb_attribute *dsdb_attribute_by_lDAPDisplayName(const struct dsdb_schema
->    *schema,
 861|                                                               const char *name)
 862|{
 863|        struct dsdb_attribute *cur;
 864|
 865|        if (!name) return NULL;
 866|
 867|        /* TODO: add binary search */
 868|        for (cur = schema->attributes; cur; cur = cur->next) {
 869|                if (strcasecmp(cur->lDAPDisplayName, name) != 0) continue;
 870|
 871|                return cur;
 872|        }
 873|
 874|        return NULL;
 875|}
 876|
 877|const struct dsdb_class *dsdb_class_by_governsID_id(const struct dsdb_schema *schema,
 878|                                                    uint32_t id)
 879|{
 880|        struct dsdb_class *cur;
 881|
 882|        /*
 883|         * 0xFFFFFFFF is used as value when no mapping table is available,
 884|         * so don't try to match with it
 885|         */
 886|        if (id == 0xFFFFFFFF) return NULL;
 887|
 888|        /* TODO: add binary search */
 889|        for (cur = schema->classes; cur; cur = cur->next) {
 890|                if (cur->governsID_id != id) continue;
```

```
891|
892|                 return cur;
893|         }
894|
895|         return NULL;
896|}
897|
898|const struct dsdb_class *dsdb_class_by_governsID_oid(const struct dsdb_schema *schema,
899|                                                     const char *oid)
900|{
901|         struct dsdb_class *cur;
902|
903|         if (!oid) return NULL;
904|
905|         /* TODO: add binary search */
906|         for (cur = schema->classes; cur; cur = cur->next) {
907|                 if (strcmp(cur->governsID_oid, oid) != 0) continue;
908|
909|                 return cur;
910|         }
911|
912|         return NULL;
913|}
914|
915|const struct dsdb_class *dsdb_class_by_lDAPDisplayName(const struct dsdb_schema *schema,
916|                                                       const char *name)
917|{
918|         struct dsdb_class *cur;
919|
920|         if (!name) return NULL;
921|
922|         /* TODO: add binary search */
923|         for (cur = schema->classes; cur; cur = cur->next) {
924|                 if (strcasecmp(cur->lDAPDisplayName, name) != 0) continue;
925|
926|                 return cur;
927|         }
928|
929|         return NULL;
930|}
931|
932|const char *dsdb_lDAPDisplayName_by_id(const struct dsdb_schema *schema,
933|                                       uint32_t id)
934|{
935|         const struct dsdb_attribute *a;
936|         const struct dsdb_class *c;
937|
938|         /* TODO: add binary search */
939|         a = dsdb_attribute_by_attributeID_id(schema, id);
940|         if (a) {
941|                 return a->lDAPDisplayName;
942|         }
943|
944|         c = dsdb_class_by_governsID_id(schema, id);
945|         if (c) {
946|                 return c->lDAPDisplayName;
947|         }
948|
949|         return NULL;
950|}
951|
952|int dsdb_set_schema(struct ldb_context *ldb, struct dsdb_schema *schema)
953|{
954|         int ret;
955|
956|         ret = ldb_set_opaque(ldb, "dsdb_schema", schema);
957|         if (ret != LDB_SUCCESS) {
958|                 return ret;
959|         }
960|
961|         talloc_steal(ldb, schema);
962|
963|         return LDB_SUCCESS;
964|}
965|
966|const struct dsdb_schema *dsdb_get_schema(struct ldb_context *ldb)
967|{
968|         const void *p;
969|         const struct dsdb_schema *schema;
970|
971|         /* see if we have a cached copy */
972|         p = ldb_get_opaque(ldb, "dsdb_schema");
973|         if (!p) {
974|                 return NULL;
975|         }
976|
977|         schema = talloc_get_type(p, struct dsdb_schema);
```

```
978|        if (!schema) {
979|                return NULL;
980|        }
981|
982|        return schema;
983|}
```

A11. source/dsdb/schema/schema_syntax.c

source/dsdb/schema/schema_syntax.c:

```
   1|/*
   2|   Unix SMB/CIFS mplementation.
   3|   DSDB schema syntaxes
   4|
   5|   Copyright (C) Stefan Metzmacher <metze@samba.org> 2006
   6|
   7|   This program is free software; you can redistribute it and/or modify
   8|   it under the terms of the GNU General Public License as published by
   9|   the Free Software Foundation; either version 2 of the License, or
  10|   (at your option) any later version.
  11|
  12|   This program is distributed in the hope that it will be useful,
  13|   but WITHOUT ANY WARRANTY; without even the implied warranty of
  14|   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  15|   GNU General Public License for more details.
  16|
  17|   You should have received a copy of the GNU General Public License
  18|   along with this program; if not, write to the Free Software
  19|   Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
  20|
  21|*/
  22|#include "includes.h"
  23|#include "dsdb/samdb/samdb.h"
  24|#include "librpc/gen_ndr/ndr_drsuapi.h"
  25|#include "lib/ldb/include/ldb.h"
  26|#include "system/time.h"
  27|#include "lib/charset/charset.h"
  28|#include "librpc/ndr/libndr.h"
  29|
  30|static WERROR dsdb_syntax_FOOBAR_drsuapi_to_ldb(const struct dsdb_schema *schema,
  31|                                                 const struct dsdb_attribute *attr,
  32|                                                 const struct drsuapi_DsReplicaAttribute
->    *in,
  33|                                                 TALLOC_CTX *mem_ctx,
  34|                                                 struct ldb_message_element *out)
  35|{
  36|        uint32_t i;
  37|
  38|        out->flags      = 0;
  39|        out->name       = talloc_strdup(mem_ctx, attr->lDAPDisplayName);
  40|        W_ERROR_HAVE_NO_MEMORY(out->name);
  41|
  42|        out->num_values = in->value_ctr.num_values;
  43|        out->values     = talloc_array(mem_ctx, struct ldb_val, out->num_values);
  44|        W_ERROR_HAVE_NO_MEMORY(out->values);
  45|
  46|        for (i=0; i < out->num_values; i++) {
  47|                char *str;
  48|
  49|                if (in->value_ctr.values[i].blob == NULL) {
  50|                        return WERR_FOOBAR;
  51|                }
  52|
  53|                str = talloc_asprintf(out->values, "%s: not implemented",
  54|                                      attr->syntax->name);
  55|                W_ERROR_HAVE_NO_MEMORY(str);
  56|
  57|                out->values[i] = data_blob_string_const(str);
  58|        }
  59|
  60|        return WERR_OK;
  61|}
  62|
  63|static WERROR dsdb_syntax_FOOBAR_ldb_to_drsuapi(const struct dsdb_schema *schema,
  64|                                                 const struct dsdb_attribute *attr,
  65|                                                 const struct ldb_message_element *in,
  66|                                                 TALLOC_CTX *mem_ctx,
  67|                                                 struct drsuapi_DsReplicaAttribute *out)
  68|{
  69|        return WERR_FOOBAR;
  70|}
  71|
  72|static WERROR dsdb_syntax_BOOL_drsuapi_to_ldb(const struct dsdb_schema *schema,
  73|                                               const struct dsdb_attribute *attr,
  74|                                               const struct drsuapi_DsReplicaAttribute
->    *in,
  75|                                               TALLOC_CTX *mem_ctx,
  76|                                               struct ldb_message_element *out)
  77|{
```

```
  78|        uint32_t i;
  79|
  80|        out->flags      = 0;
  81|        out->name       = talloc_strdup(mem_ctx, attr->lDAPDisplayName);
  82|        W_ERROR_HAVE_NO_MEMORY(out->name);
  83|
  84|        out->num_values = in->value_ctr.num_values;
  85|        out->values     = talloc_array(mem_ctx, struct ldb_val, out->num_values);
  86|        W_ERROR_HAVE_NO_MEMORY(out->values);
  87|
  88|        for (i=0; i < out->num_values; i++) {
  89|                uint32_t v;
  90|                char *str;
  91|
  92|                if (in->value_ctr.values[i].blob == NULL) {
  93|                        return WERR_FOOBAR;
  94|                }
  95|
  96|                if (in->value_ctr.values[i].blob->length != 4) {
  97|                        return WERR_FOOBAR;
  98|                }
  99|
 100|                v = IVAL(in->value_ctr.values[i].blob->data, 0);
 101|
 102|                if (v != 0) {
 103|                        str = talloc_strdup(out->values, "TRUE");
 104|                        W_ERROR_HAVE_NO_MEMORY(str);
 105|                } else {
 106|                        str = talloc_strdup(out->values, "FALSE");
 107|                        W_ERROR_HAVE_NO_MEMORY(str);
 108|                }
 109|
 110|                out->values[i] = data_blob_string_const(str);
 111|        }
 112|
 113|        return WERR_OK;
 114|}
 115|
 116|static WERROR dsdb_syntax_BOOL_ldb_to_drsuapi(const struct dsdb_schema *schema,
 117|                                               const struct dsdb_attribute *attr,
 118|                                               const struct ldb_message_element *in,
 119|                                               TALLOC_CTX *mem_ctx,
 120|                                               struct drsuapi_DsReplicaAttribute *out)
 121|{
 122|        uint32_t i;
 123|        DATA_BLOB *blobs;
 124|
 125|        if (attr->attributeID_id == 0xFFFFFFFF) {
 126|                return WERR_FOOBAR;
 127|        }
 128|
 129|        out->attid                      = attr->attributeID_id;
 130|        out->value_ctr.num_values       = in->num_values;
 131|        out->value_ctr.values           = talloc_array(mem_ctx,
 132|                                                       struct drsuapi_DsAttributeValue,
 133|                                                       in->num_values);
 134|        W_ERROR_HAVE_NO_MEMORY(out->value_ctr.values);
 135|
 136|        blobs = talloc_array(mem_ctx, DATA_BLOB, in->num_values);
 137|        W_ERROR_HAVE_NO_MEMORY(blobs);
 138|
 139|        for (i=0; i < in->num_values; i++) {
 140|                out->value_ctr.values[i].blob   = &blobs[i];
 141|
 142|                blobs[i] = data_blob_talloc(blobs, NULL, 4);
 143|                W_ERROR_HAVE_NO_MEMORY(blobs[i].data);
 144|
 145|                if (strcmp("TRUE", (const char *)in->values[i].data) == 0) {
 146|                        SIVAL(blobs[i].data, 0, 0x00000001);
 147|                } else if (strcmp("FALSE", (const char *)in->values[i].data) == 0) {
 148|                        SIVAL(blobs[i].data, 0, 0x00000000);
 149|                } else {
 150|                        return WERR_FOOBAR;
 151|                }
 152|        }
 153|
 154|        return WERR_OK;
 155|}
 156|
 157|static WERROR dsdb_syntax_INT32_drsuapi_to_ldb(const struct dsdb_schema *schema,
 158|                                                const struct dsdb_attribute *attr,
 159|                                                const struct drsuapi_DsReplicaAttribute
->    *in,
 160|                                                TALLOC_CTX *mem_ctx,
 161|                                                struct ldb_message_element *out)
 162|{
 163|        uint32_t i;
```

```
164|
165|        out->flags      = 0;
166|        out->name       = talloc_strdup(mem_ctx, attr->lDAPDisplayName);
167|        W_ERROR_HAVE_NO_MEMORY(out->name);
168|
169|        out->num_values = in->value_ctr.num_values;
170|        out->values     = talloc_array(mem_ctx, struct ldb_val, out->num_values);
171|        W_ERROR_HAVE_NO_MEMORY(out->values);
172|
173|        for (i=0; i < out->num_values; i++) {
174|                int32_t v;
175|                char *str;
176|
177|                if (in->value_ctr.values[i].blob == NULL) {
178|                        return WERR_FOOBAR;
179|                }
180|
181|                if (in->value_ctr.values[i].blob->length != 4) {
182|                        return WERR_FOOBAR;
183|                }
184|
185|                v = IVALS(in->value_ctr.values[i].blob->data, 0);
186|
187|                str = talloc_asprintf(out->values, "%d", v);
188|                W_ERROR_HAVE_NO_MEMORY(str);
189|
190|                out->values[i] = data_blob_string_const(str);
191|        }
192|
193|        return WERR_OK;
194|}
195|
196|static WERROR dsdb_syntax_INT32_ldb_to_drsuapi(const struct dsdb_schema *schema,
197|                                               const struct dsdb_attribute *attr,
198|                                               const struct ldb_message_element *in,
199|                                               TALLOC_CTX *mem_ctx,
200|                                               struct drsuapi_DsReplicaAttribute *out)
201|{
202|        uint32_t i;
203|        DATA_BLOB *blobs;
204|
205|        if (attr->attributeID_id == 0xFFFFFFFF) {
206|                return WERR_FOOBAR;
207|        }
208|
209|        out->attid                      = attr->attributeID_id;
210|        out->value_ctr.num_values       = in->num_values;
211|        out->value_ctr.values           = talloc_array(mem_ctx,
212|                                                       struct drsuapi_DsAttributeValue,
213|                                                       in->num_values);
214|        W_ERROR_HAVE_NO_MEMORY(out->value_ctr.values);
215|
216|        blobs = talloc_array(mem_ctx, DATA_BLOB, in->num_values);
217|        W_ERROR_HAVE_NO_MEMORY(blobs);
218|
219|        for (i=0; i < in->num_values; i++) {
220|                int32_t v;
221|
222|                out->value_ctr.values[i].blob   = &blobs[i];
223|
224|                blobs[i] = data_blob_talloc(blobs, NULL, 4);
225|                W_ERROR_HAVE_NO_MEMORY(blobs[i].data);
226|
227|                v = strtol((const char *)in->values[i].data, NULL, 10);
228|
229|                SIVALS(blobs[i].data, 0, v);
230|        }
231|
232|        return WERR_OK;
233|}
234|
235|static WERROR dsdb_syntax_INT64_drsuapi_to_ldb(const struct dsdb_schema *schema,
236|                                               const struct dsdb_attribute *attr,
237|                                               const struct drsuapi_DsReplicaAttribute
->    *in,
238|                                               TALLOC_CTX *mem_ctx,
239|                                               struct ldb_message_element *out)
240|{
241|        uint32_t i;
242|
243|        out->flags      = 0;
244|        out->name       = talloc_strdup(mem_ctx, attr->lDAPDisplayName);
245|        W_ERROR_HAVE_NO_MEMORY(out->name);
246|
247|        out->num_values = in->value_ctr.num_values;
248|        out->values     = talloc_array(mem_ctx, struct ldb_val, out->num_values);
249|        W_ERROR_HAVE_NO_MEMORY(out->values);
```

```
250|
251|        for (i=0; i < out->num_values; i++) {
252|                int64_t v;
253|                char *str;
254|
255|                if (in->value_ctr.values[i].blob == NULL) {
256|                        return WERR_FOOBAR;
257|                }
258|
259|                if (in->value_ctr.values[i].blob->length != 8) {
260|                        return WERR_FOOBAR;
261|                }
262|
263|                v = BVALS(in->value_ctr.values[i].blob->data, 0);
264|
265|                str = talloc_asprintf(out->values, "%lld", v);
266|                W_ERROR_HAVE_NO_MEMORY(str);
267|
268|                out->values[i] = data_blob_string_const(str);
269|        }
270|
271|        return WERR_OK;
272|}
273|
274|static WERROR dsdb_syntax_INT64_ldb_to_drsuapi(const struct dsdb_schema *schema,
275|                                               const struct dsdb_attribute *attr,
276|                                               const struct ldb_message_element *in,
277|                                               TALLOC_CTX *mem_ctx,
278|                                               struct drsuapi_DsReplicaAttribute *out)
279|{
280|        uint32_t i;
281|        DATA_BLOB *blobs;
282|
283|        if (attr->attributeID_id == 0xFFFFFFFF) {
284|                return WERR_FOOBAR;
285|        }
286|
287|        out->attid                      = attr->attributeID_id;
288|        out->value_ctr.num_values       = in->num_values;
289|        out->value_ctr.values           = talloc_array(mem_ctx,
290|                                                       struct drsuapi_DsAttributeValue,
291|                                                       in->num_values);
292|        W_ERROR_HAVE_NO_MEMORY(out->value_ctr.values);
293|
294|        blobs = talloc_array(mem_ctx, DATA_BLOB, in->num_values);
295|        W_ERROR_HAVE_NO_MEMORY(blobs);
296|
297|        for (i=0; i < in->num_values; i++) {
298|                int64_t v;
299|
300|                out->value_ctr.values[i].blob   = &blobs[i];
301|
302|                blobs[i] = data_blob_talloc(blobs, NULL, 8);
303|                W_ERROR_HAVE_NO_MEMORY(blobs[i].data);
304|
305|                v = strtoll((const char *)in->values[i].data, NULL, 10);
306|
307|                SBVALS(blobs[i].data, 0, v);
308|        }
309|
310|        return WERR_OK;
311|}
312|
313|static WERROR dsdb_syntax_NTTIME_UTC_drsuapi_to_ldb(const struct dsdb_schema *schema,
314|                                                    const struct dsdb_attribute *attr,
315|                                                    const struct
->    drsuapi_DsReplicaAttribute *in,
316|                                                    TALLOC_CTX *mem_ctx,
317|                                                    struct ldb_message_element *out)
318|{
319|        uint32_t i;
320|
321|        out->flags      = 0;
322|        out->name       = talloc_strdup(mem_ctx, attr->lDAPDisplayName);
323|        W_ERROR_HAVE_NO_MEMORY(out->name);
324|
325|        out->num_values = in->value_ctr.num_values;
326|        out->values     = talloc_array(mem_ctx, struct ldb_val, out->num_values);
327|        W_ERROR_HAVE_NO_MEMORY(out->values);
328|
329|        for (i=0; i < out->num_values; i++) {
330|                NTTIME v;
331|                time_t t;
332|                char *str;
333|
334|                if (in->value_ctr.values[i].blob == NULL) {
335|                        return WERR_FOOBAR;
```

```
336|                }
337|
338|                if (in->value_ctr.values[i].blob->length != 8) {
339|                        return WERR_FOOBAR;
340|                }
341|
342|                v = BVAL(in->value_ctr.values[i].blob->data, 0);
343|                v *= 10000000;
344|                t = nt_time_to_unix(v);
345|
346|                /*
347|                 * NOTE: On a w2k3 server you can set a GeneralizedTime string
348|                 *       via LDAP, but you get back an UTCTime string,
349|                 *       but via DRSUAPI you get back the NTTIME_1sec value
350|                 *       that represents the GeneralizedTime value!
351|                 *
352|                 *       So if we store the UTCTime string in our ldb
353|                 *       we'll loose information!
354|                 */
355|                str = ldb_timestring_utc(out->values, t);
356|                W_ERROR_HAVE_NO_MEMORY(str);
357|                out->values[i] = data_blob_string_const(str);
358|        }
359|
360|        return WERR_OK;
361|}
362|
363|static WERROR dsdb_syntax_NTTIME_UTC_ldb_to_drsuapi(const struct dsdb_schema *schema,
364|                                                    const struct dsdb_attribute *attr,
365|                                                    const struct ldb_message_element
->    *in,
366|                                                    TALLOC_CTX *mem_ctx,
367|                                                    struct drsuapi_DsReplicaAttribute
->    *out)
368|{
369|        uint32_t i;
370|        DATA_BLOB *blobs;
371|
372|        if (attr->attributeID_id == 0xFFFFFFFF) {
373|                return WERR_FOOBAR;
374|        }
375|
376|        out->attid                      = attr->attributeID_id;
377|        out->value_ctr.num_values       = in->num_values;
378|        out->value_ctr.values           = talloc_array(mem_ctx,
379|                                                       struct drsuapi_DsAttributeValue,
380|                                                       in->num_values);
381|        W_ERROR_HAVE_NO_MEMORY(out->value_ctr.values);
382|
383|        blobs = talloc_array(mem_ctx, DATA_BLOB, in->num_values);
384|        W_ERROR_HAVE_NO_MEMORY(blobs);
385|
386|        for (i=0; i < in->num_values; i++) {
387|                NTTIME v;
388|                time_t t;
389|
390|                out->value_ctr.values[i].blob   = &blobs[i];
391|
392|                blobs[i] = data_blob_talloc(blobs, NULL, 8);
393|                W_ERROR_HAVE_NO_MEMORY(blobs[i].data);
394|
395|                t = ldb_string_utc_to_time((const char *)in->values[i].data);
396|                unix_to_nt_time(&v, t);
397|                v /= 10000000;
398|
399|                SBVAL(blobs[i].data, 0, v);
400|        }
401|
402|        return WERR_OK;
403|}
404|
405|static WERROR dsdb_syntax_NTTIME_drsuapi_to_ldb(const struct dsdb_schema *schema,
406|                                                const struct dsdb_attribute *attr,
407|                                                const struct drsuapi_DsReplicaAttribute
->    *in,
408|                                                TALLOC_CTX *mem_ctx,
409|                                                struct ldb_message_element *out)
410|{
411|        uint32_t i;
412|
413|        out->flags      = 0;
414|        out->name       = talloc_strdup(mem_ctx, attr->lDAPDisplayName);
415|        W_ERROR_HAVE_NO_MEMORY(out->name);
416|
417|        out->num_values = in->value_ctr.num_values;
418|        out->values     = talloc_array(mem_ctx, struct ldb_val, out->num_values);
419|        W_ERROR_HAVE_NO_MEMORY(out->values);
```

```
420|
421|        for (i=0; i < out->num_values; i++) {
422|                NTTIME v;
423|                time_t t;
424|                char *str;
425|
426|                if (in->value_ctr.values[i].blob == NULL) {
427|                        return WERR_FOOBAR;
428|                }
429|
430|                if (in->value_ctr.values[i].blob->length != 8) {
431|                        return WERR_FOOBAR;
432|                }
433|
434|                v = BVAL(in->value_ctr.values[i].blob->data, 0);
435|                v *= 10000000;
436|                t = nt_time_to_unix(v);
437|
438|                str = ldb_timestring(out->values, t);
439|                W_ERROR_HAVE_NO_MEMORY(str);
440|
441|                out->values[i] = data_blob_string_const(str);
442|        }
443|
444|        return WERR_OK;
445|}
446|
447|static WERROR dsdb_syntax_NTTIME_ldb_to_drsuapi(const struct dsdb_schema *schema,
448|                                                const struct dsdb_attribute *attr,
449|                                                const struct ldb_message_element *in,
450|                                                TALLOC_CTX *mem_ctx,
451|                                                struct drsuapi_DsReplicaAttribute *out)
452|{
453|        uint32_t i;
454|        DATA_BLOB *blobs;
455|
456|        if (attr->attributeID_id == 0xFFFFFFFF) {
457|                return WERR_FOOBAR;
458|        }
459|
460|        out->attid                      = attr->attributeID_id;
461|        out->value_ctr.num_values       = in->num_values;
462|        out->value_ctr.values           = talloc_array(mem_ctx,
463|                                                       struct drsuapi_DsAttributeValue,
464|                                                       in->num_values);
465|        W_ERROR_HAVE_NO_MEMORY(out->value_ctr.values);
466|
467|        blobs = talloc_array(mem_ctx, DATA_BLOB, in->num_values);
468|        W_ERROR_HAVE_NO_MEMORY(blobs);
469|
470|        for (i=0; i < in->num_values; i++) {
471|                NTTIME v;
472|                time_t t;
473|
474|                out->value_ctr.values[i].blob   = &blobs[i];
475|
476|                blobs[i] = data_blob_talloc(blobs, NULL, 8);
477|                W_ERROR_HAVE_NO_MEMORY(blobs[i].data);
478|
479|                t = ldb_string_to_time((const char *)in->values[i].data);
480|                unix_to_nt_time(&v, t);
481|                v /= 10000000;
482|
483|                SBVAL(blobs[i].data, 0, v);
484|        }
485|
486|        return WERR_OK;
487|}
488|
489|static WERROR dsdb_syntax_DATA_BLOB_drsuapi_to_ldb(const struct dsdb_schema *schema,
490|                                                   const struct dsdb_attribute *attr,
491|                                                   const struct
->    drsuapi_DsReplicaAttribute *in,
492|                                                   TALLOC_CTX *mem_ctx,
493|                                                   struct ldb_message_element *out)
494|{
495|        uint32_t i;
496|
497|        out->flags      = 0;
498|        out->name       = talloc_strdup(mem_ctx, attr->lDAPDisplayName);
499|        W_ERROR_HAVE_NO_MEMORY(out->name);
500|
501|        out->num_values = in->value_ctr.num_values;
502|        out->values     = talloc_array(mem_ctx, struct ldb_val, out->num_values);
503|        W_ERROR_HAVE_NO_MEMORY(out->values);
504|
505|        for (i=0; i < out->num_values; i++) {
```

```
 506|                if (in->value_ctr.values[i].blob == NULL) {
 507|                        return WERR_FOOBAR;
 508|                }
 509|
 510|                if (in->value_ctr.values[i].blob->length == 0) {
 511|                        return WERR_FOOBAR;
 512|                }
 513|
 514|                out->values[i] = data_blob_dup_talloc(out->values,
 515|                                                      in->value_ctr.values[i].blob);
 516|                W_ERROR_HAVE_NO_MEMORY(out->values[i].data);
 517|        }
 518|
 519|        return WERR_OK;
 520|}
 521|
 522|static WERROR dsdb_syntax_DATA_BLOB_ldb_to_drsuapi(const struct dsdb_schema *schema,
 523|                                                    const struct dsdb_attribute *attr,
 524|                                                    const struct ldb_message_element *in,
 525|                                                    TALLOC_CTX *mem_ctx,
 526|                                                    struct drsuapi_DsReplicaAttribute
->    *out)
 527|{
 528|        uint32_t i;
 529|        DATA_BLOB *blobs;
 530|
 531|        if (attr->attributeID_id == 0xFFFFFFFF) {
 532|                return WERR_FOOBAR;
 533|        }
 534|
 535|        out->attid                      = attr->attributeID_id;
 536|        out->value_ctr.num_values       = in->num_values;
 537|        out->value_ctr.values           = talloc_array(mem_ctx,
 538|                                                       struct drsuapi_DsAttributeValue,
 539|                                                       in->num_values);
 540|        W_ERROR_HAVE_NO_MEMORY(out->value_ctr.values);
 541|
 542|        blobs = talloc_array(mem_ctx, DATA_BLOB, in->num_values);
 543|        W_ERROR_HAVE_NO_MEMORY(blobs);
 544|
 545|        for (i=0; i < in->num_values; i++) {
 546|                out->value_ctr.values[i].blob   = &blobs[i];
 547|
 548|                blobs[i] = data_blob_dup_talloc(blobs, &in->values[i]);
 549|                W_ERROR_HAVE_NO_MEMORY(blobs[i].data);
 550|        }
 551|
 552|        return WERR_OK;
 553|}
 554|
 555|static WERROR _dsdb_syntax_OID_obj_drsuapi_to_ldb(const struct dsdb_schema *schema,
 556|                                                  const struct dsdb_attribute *attr,
 557|                                                  const struct drsuapi_DsReplicaAttribute
->    *in,
 558|                                                  TALLOC_CTX *mem_ctx,
 559|                                                  struct ldb_message_element *out)
 560|{
 561|        uint32_t i;
 562|
 563|        out->flags      = 0;
 564|        out->name       = talloc_strdup(mem_ctx, attr->lDAPDisplayName);
 565|        W_ERROR_HAVE_NO_MEMORY(out->name);
 566|
 567|        out->num_values = in->value_ctr.num_values;
 568|        out->values     = talloc_array(mem_ctx, struct ldb_val, out->num_values);
 569|        W_ERROR_HAVE_NO_MEMORY(out->values);
 570|
 571|        for (i=0; i < out->num_values; i++) {
 572|                uint32_t v;
 573|                const struct dsdb_class *c;
 574|                const char *str;
 575|
 576|                if (in->value_ctr.values[i].blob == NULL) {
 577|                        return WERR_FOOBAR;
 578|                }
 579|
 580|                if (in->value_ctr.values[i].blob->length != 4) {
 581|                        return WERR_FOOBAR;
 582|                }
 583|
 584|                v = IVAL(in->value_ctr.values[i].blob->data, 0);
 585|
 586|                c = dsdb_class_by_governsID_id(schema, v);
 587|                if (!c) {
 588|                        return WERR_FOOBAR;
 589|                }
 590|
```

```
 591|                str = talloc_strdup(out->values, c->lDAPDisplayName);
 592|                W_ERROR_HAVE_NO_MEMORY(str);
 593|
 594|                /* the values need to be reversed */
 595|                out->values[out->num_values - (i + 1)] = data_blob_string_const(str);
 596|        }
 597|
 598|        return WERR_OK;
 599|}
 600|
 601|static WERROR _dsdb_syntax_OID_oid_drsuapi_to_ldb(const struct dsdb_schema *schema,
 602|                                                  const struct dsdb_attribute *attr,
 603|                                                  const struct drsuapi_DsReplicaAttribute
->    *in,
 604|                                                  TALLOC_CTX *mem_ctx,
 605|                                                  struct ldb_message_element *out)
 606|{
 607|        uint32_t i;
 608|
 609|        out->flags      = 0;
 610|        out->name       = talloc_strdup(mem_ctx, attr->lDAPDisplayName);
 611|        W_ERROR_HAVE_NO_MEMORY(out->name);
 612|
 613|        out->num_values = in->value_ctr.num_values;
 614|        out->values     = talloc_array(mem_ctx, struct ldb_val, out->num_values);
 615|        W_ERROR_HAVE_NO_MEMORY(out->values);
 616|
 617|        for (i=0; i < out->num_values; i++) {
 618|                uint32_t v;
 619|                WERROR status;
 620|                const char *str;
 621|
 622|                if (in->value_ctr.values[i].blob == NULL) {
 623|                        return WERR_FOOBAR;
 624|                }
 625|
 626|                if (in->value_ctr.values[i].blob->length != 4) {
 627|                        return WERR_FOOBAR;
 628|                }
 629|
 630|                v = IVAL(in->value_ctr.values[i].blob->data, 0);
 631|
 632|                status = dsdb_map_int2oid(schema, v, out->values, &str);
 633|                W_ERROR_NOT_OK_RETURN(status);
 634|
 635|                out->values[i] = data_blob_string_const(str);
 636|        }
 637|
 638|        return WERR_OK;
 639|}
 640|
 641|static WERROR dsdb_syntax_OID_drsuapi_to_ldb(const struct dsdb_schema *schema,
 642|                                              const struct dsdb_attribute *attr,
 643|                                              const struct drsuapi_DsReplicaAttribute
->    *in,
 644|                                              TALLOC_CTX *mem_ctx,
 645|                                              struct ldb_message_element *out)
 646|{
 647|        uint32_t i;
 648|
 649|        switch (attr->attributeID_id) {
 650|        case DRSUAPI_ATTRIBUTE_objectClass:
 651|                return _dsdb_syntax_OID_obj_drsuapi_to_ldb(schema, attr, in, mem_ctx,
->    out);
 652|        case DRSUAPI_ATTRIBUTE_governsID:
 653|        case DRSUAPI_ATTRIBUTE_attributeID:
 654|        case DRSUAPI_ATTRIBUTE_attributeSyntax:
 655|                return _dsdb_syntax_OID_oid_drsuapi_to_ldb(schema, attr, in, mem_ctx,
->    out);
 656|        }
 657|
 658|        out->flags      = 0;
 659|        out->name       = talloc_strdup(mem_ctx, attr->lDAPDisplayName);
 660|        W_ERROR_HAVE_NO_MEMORY(out->name);
 661|
 662|        out->num_values = in->value_ctr.num_values;
 663|        out->values     = talloc_array(mem_ctx, struct ldb_val, out->num_values);
 664|        W_ERROR_HAVE_NO_MEMORY(out->values);
 665|
 666|        for (i=0; i < out->num_values; i++) {
 667|                uint32_t v;
 668|                const char *name;
 669|                char *str;
 670|
 671|                if (in->value_ctr.values[i].blob == NULL) {
 672|                        return WERR_FOOBAR;
 673|                }
```

```
674|
675|                 if (in->value_ctr.values[i].blob->length != 4) {
676|                         return WERR_FOOBAR;
677|                 }
678|
679|                 v = IVAL(in->value_ctr.values[i].blob->data, 0);
680|
681|                 name = dsdb_lDAPDisplayName_by_id(schema, v);
682|                 if (!name) {
683|                         return WERR_FOOBAR;
684|                 }
685|
686|                 str = talloc_strdup(out->values, name);
687|                 W_ERROR_HAVE_NO_MEMORY(str);
688|
689|                 out->values[i] = data_blob_string_const(str);
690|         }
691|
692|         return WERR_OK;
693| }
694|
695| static WERROR dsdb_syntax_OID_ldb_to_drsuapi(const struct dsdb_schema *schema,
696|                                              const struct dsdb_attribute *attr,
697|                                              const struct ldb_message_element *in,
698|                                              TALLOC_CTX *mem_ctx,
699|                                              struct drsuapi_DsReplicaAttribute *out)
700| {
701|         uint32_t i;
702|         DATA_BLOB *blobs;
703|
704|         if (attr->attributeID_id == 0xFFFFFFFF) {
705|                 return WERR_FOOBAR;
706|         }
707|
708|         switch (attr->attributeID_id) {
709|         case DRSUAPI_ATTRIBUTE_objectClass:
710|         case DRSUAPI_ATTRIBUTE_governsID:
711|         case DRSUAPI_ATTRIBUTE_attributeID:
712|         case DRSUAPI_ATTRIBUTE_attributeSyntax:
713|                 return dsdb_syntax_FOOBAR_ldb_to_drsuapi(schema, attr, in, mem_ctx,
->    out);
714|         }
715|
716|         out->attid                     = attr->attributeID_id;
717|         out->value_ctr.num_values      = in->num_values;
718|         out->value_ctr.values          = talloc_array(mem_ctx,
719|                                                        struct drsuapi_DsAttributeValue,
720|                                                        in->num_values);
721|         W_ERROR_HAVE_NO_MEMORY(out->value_ctr.values);
722|
723|         blobs = talloc_array(mem_ctx, DATA_BLOB, in->num_values);
724|         W_ERROR_HAVE_NO_MEMORY(blobs);
725|
726|         for (i=0; i < in->num_values; i++) {
727|                 uint32_t v;
728|
729|                 out->value_ctr.values[i].blob   = &blobs[i];
730|
731|                 blobs[i] = data_blob_talloc(blobs, NULL, 4);
732|                 W_ERROR_HAVE_NO_MEMORY(blobs[i].data);
733|
734|                 v = strtol((const char *)in->values[i].data, NULL, 10);
735|
736|                 SIVAL(blobs[i].data, 0, v);
737|         }
738|
739|         return WERR_OK;
740| }
741|
742| static WERROR dsdb_syntax_UNICODE_drsuapi_to_ldb(const struct dsdb_schema *schema,
743|                                                  const struct dsdb_attribute *attr,
744|                                                  const struct drsuapi_DsReplicaAttribute
->    *in,
745|                                                  TALLOC_CTX *mem_ctx,
746|                                                  struct ldb_message_element *out)
747| {
748|         uint32_t i;
749|
750|         out->flags      = 0;
751|         out->name       = talloc_strdup(mem_ctx, attr->lDAPDisplayName);
752|         W_ERROR_HAVE_NO_MEMORY(out->name);
753|
754|         out->num_values = in->value_ctr.num_values;
755|         out->values     = talloc_array(mem_ctx, struct ldb_val, out->num_values);
756|         W_ERROR_HAVE_NO_MEMORY(out->values);
757|
758|         for (i=0; i < out->num_values; i++) {
```

```
759|                ssize_t ret;
760|                char *str;
761|
762|                if (in->value_ctr.values[i].blob == NULL) {
763|                        return WERR_FOOBAR;
764|                }
765|
766|                if (in->value_ctr.values[i].blob->length == 0) {
767|                        return WERR_FOOBAR;
768|                }
769|
770|                ret = convert_string_talloc(out->values, CH_UTF16, CH_UNIX,
771|                                            in->value_ctr.values[i].blob->data,
772|                                            in->value_ctr.values[i].blob->length,
773|                                            (void **)&str);
774|                if (ret == -1) {
775|                        return WERR_FOOBAR;
776|                }
777|
778|                out->values[i] = data_blob_string_const(str);
779|        }
780|
781|        return WERR_OK;
782|}
783|
784|static WERROR dsdb_syntax_UNICODE_ldb_to_drsuapi(const struct dsdb_schema *schema,
785|                                                 const struct dsdb_attribute *attr,
786|                                                 const struct ldb_message_element *in,
787|                                                 TALLOC_CTX *mem_ctx,
788|                                                 struct drsuapi_DsReplicaAttribute *out)
789|{
790|        uint32_t i;
791|        DATA_BLOB *blobs;
792|
793|        if (attr->attributeID_id == 0xFFFFFFFF) {
794|                return WERR_FOOBAR;
795|        }
796|
797|        out->attid                      = attr->attributeID_id;
798|        out->value_ctr.num_values       = in->num_values;
799|        out->value_ctr.values           = talloc_array(mem_ctx,
800|                                                       struct drsuapi_DsAttributeValue,
801|                                                       in->num_values);
802|        W_ERROR_HAVE_NO_MEMORY(out->value_ctr.values);
803|
804|        blobs = talloc_array(mem_ctx, DATA_BLOB, in->num_values);
805|        W_ERROR_HAVE_NO_MEMORY(blobs);
806|
807|        for (i=0; i < in->num_values; i++) {
808|                ssize_t ret;
809|
810|                out->value_ctr.values[i].blob   = &blobs[i];
811|
812|                ret = convert_string_talloc(blobs, CH_UNIX, CH_UTF16,
813|                                            in->values[i].data,
814|                                            in->values[i].length,
815|                                            (void **)&blobs[i].data);
816|                if (ret == -1) {
817|                        return WERR_FOOBAR;
818|                }
819|                blobs[i].length = ret;
820|        }
821|
822|        return WERR_OK;
823|}
824|
825|static WERROR dsdb_syntax_DN_drsuapi_to_ldb(const struct dsdb_schema *schema,
826|                                            const struct dsdb_attribute *attr,
827|                                            const struct drsuapi_DsReplicaAttribute *in,
828|                                            TALLOC_CTX *mem_ctx,
829|                                            struct ldb_message_element *out)
830|{
831|        uint32_t i;
832|
833|        out->flags      = 0;
834|        out->name       = talloc_strdup(mem_ctx, attr->lDAPDisplayName);
835|        W_ERROR_HAVE_NO_MEMORY(out->name);
836|
837|        out->num_values = in->value_ctr.num_values;
838|        out->values     = talloc_array(mem_ctx, struct ldb_val, out->num_values);
839|        W_ERROR_HAVE_NO_MEMORY(out->values);
840|
841|        for (i=0; i < out->num_values; i++) {
842|                struct drsuapi_DsReplicaObjectIdentifier3 id3;
843|                NTSTATUS status;
844|
845|                if (in->value_ctr.values[i].blob == NULL) {
```

```
846|                       return WERR_FOOBAR;
847|               }
848|
849|               if (in->value_ctr.values[i].blob->length == 0) {
850|                       return WERR_FOOBAR;
851|               }
852|
853|               status = ndr_pull_struct_blob_all(in->value_ctr.values[i].blob,
854|                                                  out->values, &id3,
855|                                                  (ndr_pull_flags_fn_t)ndr_pull_drsuapi_D
->   sReplicaObjectIdentifier3);
856|               if (!NT_STATUS_IS_OK(status)) {
857|                       return ntstatus_to_werror(status);
858|               }
859|
860|               /* TODO: handle id3.guid and id3.sid */
861|               out->values[i] = data_blob_string_const(id3.dn);
862|       }
863|
864|       return WERR_OK;
865|}
866|
867|static WERROR dsdb_syntax_DN_ldb_to_drsuapi(const struct dsdb_schema *schema,
868|                                            const struct dsdb_attribute *attr,
869|                                            const struct ldb_message_element *in,
870|                                            TALLOC_CTX *mem_ctx,
871|                                            struct drsuapi_DsReplicaAttribute *out)
872|{
873|       uint32_t i;
874|       DATA_BLOB *blobs;
875|
876|       if (attr->attributeID_id == 0xFFFFFFFF) {
877|               return WERR_FOOBAR;
878|       }
879|
880|       out->attid                      = attr->attributeID_id;
881|       out->value_ctr.num_values      = in->num_values;
882|       out->value_ctr.values          = talloc_array(mem_ctx,
883|                                                     struct drsuapi_DsAttributeValue,
884|                                                     in->num_values);
885|       W_ERROR_HAVE_NO_MEMORY(out->value_ctr.values);
886|
887|       blobs = talloc_array(mem_ctx, DATA_BLOB, in->num_values);
888|       W_ERROR_HAVE_NO_MEMORY(blobs);
889|
890|       for (i=0; i < in->num_values; i++) {
891|               NTSTATUS status;
892|               struct drsuapi_DsReplicaObjectIdentifier3 id3;
893|
894|               out->value_ctr.values[i].blob   = &blobs[i];
895|
896|               /* TODO: handle id3.guid and id3.sid */
897|               ZERO_STRUCT(id3);
898|               id3.dn = (const char *)in->values[i].data;
899|
900|               status = ndr_push_struct_blob(&blobs[i], blobs, &id3,
901|                                              (ndr_push_flags_fn_t)ndr_push_drsuapi_DsRep
->   licaObjectIdentifier3);
902|               if (!NT_STATUS_IS_OK(status)) {
903|                       return ntstatus_to_werror(status);
904|               }
905|       }
906|
907|       return WERR_OK;
908|}
909|
910|static WERROR dsdb_syntax_DN_BINARY_drsuapi_to_ldb(const struct dsdb_schema *schema,
911|                                                   const struct dsdb_attribute *attr,
912|                                                   const struct
->    drsuapi_DsReplicaAttribute *in,
913|                                                   TALLOC_CTX *mem_ctx,
914|                                                   struct ldb_message_element *out)
915|{
916|       uint32_t i;
917|
918|       out->flags      = 0;
919|       out->name       = talloc_strdup(mem_ctx, attr->lDAPDisplayName);
920|       W_ERROR_HAVE_NO_MEMORY(out->name);
921|
922|       out->num_values = in->value_ctr.num_values;
923|       out->values     = talloc_array(mem_ctx, struct ldb_val, out->num_values);
924|       W_ERROR_HAVE_NO_MEMORY(out->values);
925|
926|       for (i=0; i < out->num_values; i++) {
927|               struct drsuapi_DsReplicaObjectIdentifier3Binary id3b;
928|               char *binary;
929|               char *str;
```

```
 930|                NTSTATUS status;
 931|
 932|                if (in->value_ctr.values[i].blob == NULL) {
 933|                        return WERR_FOOBAR;
 934|                }
 935|
 936|                if (in->value_ctr.values[i].blob->length == 0) {
 937|                        return WERR_FOOBAR;
 938|                }
 939|
 940|                status = ndr_pull_struct_blob_all(in->value_ctr.values[i].blob,
 941|                                                  out->values, &id3b,
 942|                                                  (ndr_pull_flags_fn_t)ndr_pull_drsuapi_D
->   sReplicaObjectIdentifier3Binary);
 943|                if (!NT_STATUS_IS_OK(status)) {
 944|                        return ntstatus_to_werror(status);
 945|                }
 946|
 947|                /* TODO: handle id3.guid and id3.sid */
 948|                binary = data_blob_hex_string(out->values, &id3b.binary);
 949|                W_ERROR_HAVE_NO_MEMORY(binary);
 950|
 951|                str = talloc_asprintf(out->values, "B:%u:%s:%s",
 952|                                      id3b.binary.length * 2, /* because of 2 hex chars
->    per byte */
 953|                                      binary,
 954|                                      id3b.dn);
 955|                W_ERROR_HAVE_NO_MEMORY(str);
 956|
 957|                /* TODO: handle id3.guid and id3.sid */
 958|                out->values[i] = data_blob_string_const(str);
 959|        }
 960|
 961|        return WERR_OK;
 962|}
 963|
 964|static WERROR dsdb_syntax_DN_BINARY_ldb_to_drsuapi(const struct dsdb_schema *schema,
 965|                                                   const struct dsdb_attribute *attr,
 966|                                                   const struct ldb_message_element *in,
 967|                                                   TALLOC_CTX *mem_ctx,
 968|                                                   struct drsuapi_DsReplicaAttribute
->    *out)
 969|{
 970|        uint32_t i;
 971|        DATA_BLOB *blobs;
 972|
 973|        if (attr->attributeID_id == 0xFFFFFFFF) {
 974|                return WERR_FOOBAR;
 975|        }
 976|
 977|        out->attid                      = attr->attributeID_id;
 978|        out->value_ctr.num_values       = in->num_values;
 979|        out->value_ctr.values           = talloc_array(mem_ctx,
 980|                                                       struct drsuapi_DsAttributeValue,
 981|                                                       in->num_values);
 982|        W_ERROR_HAVE_NO_MEMORY(out->value_ctr.values);
 983|
 984|        blobs = talloc_array(mem_ctx, DATA_BLOB, in->num_values);
 985|        W_ERROR_HAVE_NO_MEMORY(blobs);
 986|
 987|        for (i=0; i < in->num_values; i++) {
 988|                NTSTATUS status;
 989|                struct drsuapi_DsReplicaObjectIdentifier3Binary id3b;
 990|
 991|                out->value_ctr.values[i].blob   = &blobs[i];
 992|
 993|                /* TODO: handle id3b.guid and id3b.sid, id3.binary */
 994|                ZERO_STRUCT(id3b);
 995|                id3b.dn         = (const char *)in->values[i].data;
 996|                id3b.binary     = data_blob(NULL, 0);
 997|
 998|                status = ndr_push_struct_blob(&blobs[i], blobs, &id3b,
 999|                                              (ndr_push_flags_fn_t)ndr_push_drsuapi_DsRep
->   licaObjectIdentifier3Binary);
1000|                if (!NT_STATUS_IS_OK(status)) {
1001|                        return ntstatus_to_werror(status);
1002|                }
1003|        }
1004|
1005|        return WERR_OK;
1006|}
1007|
1008|static WERROR dsdb_syntax_PRESENTATION_ADDRESS_drsuapi_to_ldb(const struct dsdb_schema
->    *schema,
1009|                                                              const struct dsdb_attribute
->    *attr,
1010|                                                              const struct
```

```
->    drsuapi_DsReplicaAttribute *in,
1011|                                                                 TALLOC_CTX *mem_ctx,
1012|                                                                 struct ldb_message_element
->    *out)
1013|{
1014|        uint32_t i;
1015|
1016|        out->flags      = 0;
1017|        out->name       = talloc_strdup(mem_ctx, attr->lDAPDisplayName);
1018|        W_ERROR_HAVE_NO_MEMORY(out->name);
1019|
1020|        out->num_values = in->value_ctr.num_values;
1021|        out->values     = talloc_array(mem_ctx, struct ldb_val, out->num_values);
1022|        W_ERROR_HAVE_NO_MEMORY(out->values);
1023|
1024|        for (i=0; i < out->num_values; i++) {
1025|                uint32_t len;
1026|                ssize_t ret;
1027|                char *str;
1028|
1029|                if (in->value_ctr.values[i].blob == NULL) {
1030|                        return WERR_FOOBAR;
1031|                }
1032|
1033|                if (in->value_ctr.values[i].blob->length < 4) {
1034|                        return WERR_FOOBAR;
1035|                }
1036|
1037|                len = IVAL(in->value_ctr.values[i].blob->data, 0);
1038|
1039|                if (len != in->value_ctr.values[i].blob->length) {
1040|                        return WERR_FOOBAR;
1041|                }
1042|
1043|                ret = convert_string_talloc(out->values, CH_UTF16, CH_UNIX,
1044|                                            in->value_ctr.values[i].blob->data+4,
1045|                                            in->value_ctr.values[i].blob->length-4,
1046|                                            (void **)&str);
1047|                if (ret == -1) {
1048|                        return WERR_FOOBAR;
1049|                }
1050|
1051|                out->values[i] = data_blob_string_const(str);
1052|        }
1053|
1054|        return WERR_OK;
1055|}
1056|
1057|static WERROR dsdb_syntax_PRESENTATION_ADDRESS_ldb_to_drsuapi(const struct dsdb_schema
->    *schema,
1058|                                                                 const struct dsdb_attribute
->    *attr,
1059|                                                                 const struct
->    ldb_message_element *in,
1060|                                                                 TALLOC_CTX *mem_ctx,
1061|                                                                 struct
->    drsuapi_DsReplicaAttribute *out)
1062|{
1063|        uint32_t i;
1064|        DATA_BLOB *blobs;
1065|
1066|        if (attr->attributeID_id == 0xFFFFFFFF) {
1067|                return WERR_FOOBAR;
1068|        }
1069|
1070|        out->attid                      = attr->attributeID_id;
1071|        out->value_ctr.num_values       = in->num_values;
1072|        out->value_ctr.values           = talloc_array(mem_ctx,
1073|                                                        struct drsuapi_DsAttributeValue,
1074|                                                        in->num_values);
1075|        W_ERROR_HAVE_NO_MEMORY(out->value_ctr.values);
1076|
1077|        blobs = talloc_array(mem_ctx, DATA_BLOB, in->num_values);
1078|        W_ERROR_HAVE_NO_MEMORY(blobs);
1079|
1080|        for (i=0; i < in->num_values; i++) {
1081|                uint8_t *data;
1082|                ssize_t ret;
1083|
1084|                out->value_ctr.values[i].blob   = &blobs[i];
1085|
1086|                ret = convert_string_talloc(blobs, CH_UNIX, CH_UTF16,
1087|                                            in->values[i].data,
1088|                                            in->values[i].length,
1089|                                            (void **)&data);
1090|                if (ret == -1) {
1091|                        return WERR_FOOBAR;
```

```
1092|                }
1093|
1094|                blobs[i] = data_blob_talloc(blobs, NULL, 4 + ret);
1095|                W_ERROR_HAVE_NO_MEMORY(blobs[i].data);
1096|
1097|                SIVAL(blobs[i].data, 0, 4 + ret);
1098|
1099|                if (ret > 0) {
1100|                        memcpy(blobs[i].data + 4, data, ret);
1101|                        talloc_free(data);
1102|                }
1103|        }
1104|
1105|        return WERR_OK;
1106|}
1107|
1108|
1109|#define OMOBJECTCLASS(val) { .length = sizeof(val) - 1, .data = discard_const_p(uint8_t,
->    val) }
1110|
1111|static const struct dsdb_syntax dsdb_syntaxes[] = {
1112|        {
1113|                .name                   = "Boolean",
1114|                .ldap_oid               = "1.3.6.1.4.1.1466.115.121.1.7",
1115|                .oMSyntax               = 1,
1116|                .attributeSyntax_oid    = "2.5.5.8",
1117|                .drsuapi_to_ldb         = dsdb_syntax_BOOL_drsuapi_to_ldb,
1118|                .ldb_to_drsuapi         = dsdb_syntax_BOOL_ldb_to_drsuapi,
1119|        },{
1120|                .name                   = "Integer",
1121|                .ldap_oid               = "1.3.6.1.4.1.1466.115.121.1.27",
1122|                .oMSyntax               = 2,
1123|                .attributeSyntax_oid    = "2.5.5.9",
1124|                .drsuapi_to_ldb         = dsdb_syntax_INT32_drsuapi_to_ldb,
1125|                .ldb_to_drsuapi         = dsdb_syntax_INT32_ldb_to_drsuapi,
1126|        },{
1127|                .name                   = "String(Octet)",
1128|                .ldap_oid               = "1.3.6.1.4.1.1466.115.121.1.40",
1129|                .oMSyntax               = 4,
1130|                .attributeSyntax_oid    = "2.5.5.10",
1131|                .drsuapi_to_ldb         = dsdb_syntax_DATA_BLOB_drsuapi_to_ldb,
1132|                .ldb_to_drsuapi         = dsdb_syntax_DATA_BLOB_ldb_to_drsuapi,
1133|        },{
1134|                .name                   = "String(Sid)",
1135|                .ldap_oid               = "1.3.6.1.4.1.1466.115.121.1.40",
1136|                .oMSyntax               = 4,
1137|                .attributeSyntax_oid    = "2.5.5.17",
1138|                .drsuapi_to_ldb         = dsdb_syntax_DATA_BLOB_drsuapi_to_ldb,
1139|                .ldb_to_drsuapi         = dsdb_syntax_DATA_BLOB_ldb_to_drsuapi,
1140|        },{
1141|                .name                   = "String(Object-Identifier)",
1142|                .ldap_oid               = "1.3.6.1.4.1.1466.115.121.1.38",
1143|                .oMSyntax               = 6,
1144|                .attributeSyntax_oid    = "2.5.5.2",
1145|                .drsuapi_to_ldb         = dsdb_syntax_OID_drsuapi_to_ldb,
1146|                .ldb_to_drsuapi         = dsdb_syntax_OID_ldb_to_drsuapi,
1147|        },{
1148|                .name                   = "Enumeration",
1149|                .ldap_oid               = "1.3.6.1.4.1.1466.115.121.1.27",
1150|                .oMSyntax               = 10,
1151|                .attributeSyntax_oid    = "2.5.5.9",
1152|                .drsuapi_to_ldb         = dsdb_syntax_INT32_drsuapi_to_ldb,
1153|                .ldb_to_drsuapi         = dsdb_syntax_INT32_ldb_to_drsuapi,
1154|        },{
1155|        /* not used in w2k3 forest */
1156|                .name                   = "String(Numeric)",
1157|                .ldap_oid               = "1.3.6.1.4.1.1466.115.121.1.36",
1158|                .oMSyntax               = 18,
1159|                .attributeSyntax_oid    = "2.5.5.6",
1160|                .drsuapi_to_ldb         = dsdb_syntax_DATA_BLOB_drsuapi_to_ldb,
1161|                .ldb_to_drsuapi         = dsdb_syntax_DATA_BLOB_ldb_to_drsuapi,
1162|        },{
1163|                .name                   = "String(Printable)",
1164|                .ldap_oid               = "1.3.6.1.4.1.1466.115.121.1.44",
1165|                .oMSyntax               = 19,
1166|                .attributeSyntax_oid    = "2.5.5.5",
1167|                .drsuapi_to_ldb         = dsdb_syntax_DATA_BLOB_drsuapi_to_ldb,
1168|                .ldb_to_drsuapi         = dsdb_syntax_DATA_BLOB_ldb_to_drsuapi,
1169|        },{
1170|                .name                   = "String(Teletex)",
1171|                .ldap_oid               = "1.2.840.113556.1.4.905",
1172|                .oMSyntax               = 20,
1173|                .attributeSyntax_oid    = "2.5.5.4",
1174|                .drsuapi_to_ldb         = dsdb_syntax_DATA_BLOB_drsuapi_to_ldb,
1175|                .ldb_to_drsuapi         = dsdb_syntax_DATA_BLOB_ldb_to_drsuapi,
1176|        },{
1177|                .name                   = "String(IA5)",
```

```
1178|                .ldap_oid               = "1.3.6.1.4.1.1466.115.121.1.26",
1179|                .oMSyntax               = 22,
1180|                .attributeSyntax_oid    = "2.5.5.5",
1181|                .drsuapi_to_ldb         = dsdb_syntax_DATA_BLOB_drsuapi_to_ldb,
1182|                .ldb_to_drsuapi         = dsdb_syntax_DATA_BLOB_ldb_to_drsuapi,
1183|        },{
1184|                .name                   = "String(UTC-Time)",
1185|                .ldap_oid               = "1.3.6.1.4.1.1466.115.121.1.53",
1186|                .oMSyntax               = 23,
1187|                .attributeSyntax_oid    = "2.5.5.11",
1188|                .drsuapi_to_ldb         = dsdb_syntax_NTTIME_UTC_drsuapi_to_ldb,
1189|                .ldb_to_drsuapi         = dsdb_syntax_NTTIME_UTC_ldb_to_drsuapi,
1190|        },{
1191|                .name                   = "String(Generalized-Time)",
1192|                .ldap_oid               = "1.3.6.1.4.1.1466.115.121.1.24",
1193|                .oMSyntax               = 24,
1194|                .attributeSyntax_oid    = "2.5.5.11",
1195|                .drsuapi_to_ldb         = dsdb_syntax_NTTIME_drsuapi_to_ldb,
1196|                .ldb_to_drsuapi         = dsdb_syntax_NTTIME_ldb_to_drsuapi,
1197|        },{
1198|        /* not used in w2k3 schema */
1199|                .name                   = "String(Case Sensitive)",
1200|                .ldap_oid               = "1.2.840.113556.1.4.1362",
1201|                .oMSyntax               = 27,
1202|                .attributeSyntax_oid    = "2.5.5.3",
1203|                .drsuapi_to_ldb         = dsdb_syntax_FOOBAR_drsuapi_to_ldb,
1204|                .ldb_to_drsuapi         = dsdb_syntax_FOOBAR_ldb_to_drsuapi,
1205|        },{
1206|                .name                   = "String(Unicode)",
1207|                .ldap_oid               = "1.3.6.1.4.1.1466.115.121.1.15",
1208|                .oMSyntax               = 64,
1209|                .attributeSyntax_oid    = "2.5.5.12",
1210|                .drsuapi_to_ldb         = dsdb_syntax_UNICODE_drsuapi_to_ldb,
1211|                .ldb_to_drsuapi         = dsdb_syntax_UNICODE_ldb_to_drsuapi,
1212|        },{
1213|                .name                   = "Interval/LargeInteger",
1214|                .ldap_oid               = "1.2.840.113556.1.4.906",
1215|                .oMSyntax               = 65,
1216|                .attributeSyntax_oid    = "2.5.5.16",
1217|                .drsuapi_to_ldb         = dsdb_syntax_INT64_drsuapi_to_ldb,
1218|                .ldb_to_drsuapi         = dsdb_syntax_INT64_ldb_to_drsuapi,
1219|        },{
1220|                .name                   = "String(NT-Sec-Desc)",
1221|                .ldap_oid               = "1.2.840.113556.1.4.907",
1222|                .oMSyntax               = 66,
1223|                .attributeSyntax_oid    = "2.5.5.15",
1224|                .drsuapi_to_ldb         = dsdb_syntax_DATA_BLOB_drsuapi_to_ldb,
1225|                .ldb_to_drsuapi         = dsdb_syntax_DATA_BLOB_ldb_to_drsuapi,
1226|        },{
1227|                .name                   = "Object(DS-DN)",
1228|                .ldap_oid               = "1.3.6.1.4.1.1466.115.121.1.12",
1229|                .oMSyntax               = 127,
1230|                .oMObjectClass          = OMOBJECTCLASS("\x2b\x0c\x02\x87\x73\x1c\x00\x85
->   \x4a"),
1231|                .attributeSyntax_oid    = "2.5.5.1",
1232|                .drsuapi_to_ldb         = dsdb_syntax_DN_drsuapi_to_ldb,
1233|                .ldb_to_drsuapi         = dsdb_syntax_DN_ldb_to_drsuapi,
1234|        },{
1235|                .name                   = "Object(DN-Binary)",
1236|                .ldap_oid               = "1.2.840.113556.1.4.903",
1237|                .oMSyntax               = 127,
1238|                .oMObjectClass          = OMOBJECTCLASS("\x2a\x86\x48\x86\xf7\x14\x01\x01
->   \x01\x0b"),
1239|                .attributeSyntax_oid    = "2.5.5.7",
1240|                .drsuapi_to_ldb         = dsdb_syntax_DN_BINARY_drsuapi_to_ldb,
1241|                .ldb_to_drsuapi         = dsdb_syntax_DN_BINARY_ldb_to_drsuapi,
1242|        },{
1243|        /* not used in w2k3 schema */
1244|                .name                   = "Object(OR-Name)",
1245|                .ldap_oid               = "1.2.840.113556.1.4.1221",
1246|                .oMSyntax               = 127,
1247|                .oMObjectClass          = OMOBJECTCLASS("\x56\x06\x01\x02\x05\x0b\x1D"),
1248|                .attributeSyntax_oid    = "2.5.5.7",
1249|                .drsuapi_to_ldb         = dsdb_syntax_FOOBAR_drsuapi_to_ldb,
1250|                .ldb_to_drsuapi         = dsdb_syntax_FOOBAR_ldb_to_drsuapi,
1251|        },{
1252|        /*
1253|         * TODO: verify if DATA_BLOB is correct here...!
1254|         *
1255|         *       repsFrom and repsTo are the only attributes using
1256|         *       this attribute syntax, but they're not replicated...
1257|         */
1258|                .name                   = "Object(Replica-Link)",
1259|                .ldap_oid               = "1.3.6.1.4.1.1466.115.121.1.40",
1260|                .oMSyntax               = 127,
1261|                .oMObjectClass          = OMOBJECTCLASS("\x2a\x86\x48\x86\xf7\x14\x01\x01
->   \x01\x06"),
```

```
1262|                .attributeSyntax_oid    = "2.5.5.10",
1263|                .drsuapi_to_ldb         = dsdb_syntax_DATA_BLOB_drsuapi_to_ldb,
1264|                .ldb_to_drsuapi         = dsdb_syntax_DATA_BLOB_ldb_to_drsuapi,
1265|        },{
1266|                .name                   = "Object(Presentation-Address)",
1267|                .ldap_oid               = "1.3.6.1.4.1.1466.115.121.1.43",
1268|                .oMSyntax               = 127,
1269|                .oMObjectClass          = OMOBJECTCLASS("\x2b\x0c\x02\x87\x73\x1c\x00\x85
->   \x5c"),
1270|                .attributeSyntax_oid    = "2.5.5.13",
1271|                .drsuapi_to_ldb         = dsdb_syntax_PRESENTATION_ADDRESS_drsuapi_to_ldb
->   ,
1272|                .ldb_to_drsuapi         = dsdb_syntax_PRESENTATION_ADDRESS_ldb_to_drsuapi
->   ,
1273|        },{
1274|        /* not used in w2k3 schema */
1275|                .name                   = "Object(Access-Point)",
1276|                .ldap_oid               = "1.3.6.1.4.1.1466.115.121.1.2",
1277|                .oMSyntax               = 127,
1278|                .oMObjectClass          = OMOBJECTCLASS("\x2b\x0c\x02\x87\x73\x1c\x00\x85
->   \x3e"),
1279|                .attributeSyntax_oid    = "2.5.5.14",
1280|                .drsuapi_to_ldb         = dsdb_syntax_FOOBAR_drsuapi_to_ldb,
1281|                .ldb_to_drsuapi         = dsdb_syntax_FOOBAR_ldb_to_drsuapi,
1282|        },{
1283|        /* not used in w2k3 schema */
1284|                .name                   = "Object(DN-String)",
1285|                .ldap_oid               = "1.2.840.113556.1.4.904",
1286|                .oMSyntax               = 127,
1287|                .oMObjectClass          = OMOBJECTCLASS("\x2a\x86\x48\x86\xf7\x14\x01\x01
->   \x01\x0c"),
1288|                .attributeSyntax_oid    = "2.5.5.14",
1289|                .drsuapi_to_ldb         = dsdb_syntax_FOOBAR_drsuapi_to_ldb,
1290|                .ldb_to_drsuapi         = dsdb_syntax_FOOBAR_ldb_to_drsuapi,
1291|        }
1292|};
1293|
1294|const struct dsdb_syntax *dsdb_syntax_for_attribute(const struct dsdb_attribute *attr)
1295|{
1296|        uint32_t i;
1297|
1298|        for (i=0; i < ARRAY_SIZE(dsdb_syntaxes); i++) {
1299|                if (attr->oMSyntax != dsdb_syntaxes[i].oMSyntax) continue;
1300|
1301|                if (attr->oMObjectClass.length != dsdb_syntaxes[i].oMObjectClass.length)
->    continue;
1302|
1303|                if (attr->oMObjectClass.length) {
1304|                        int ret;
1305|                        ret = memcmp(attr->oMObjectClass.data,
1306|                                     dsdb_syntaxes[i].oMObjectClass.data,
1307|                                     attr->oMObjectClass.length);
1308|                        if (ret != 0) continue;
1309|                }
1310|
1311|                if (strcmp(attr->attributeSyntax_oid, dsdb_syntaxes[i].attributeSyntax_oi
->   d) != 0) continue;
1312|
1313|                return &dsdb_syntaxes[i];
1314|        }
1315|
1316|        return NULL;
1317|}
1318|
1319|WERROR dsdb_attribute_drsuapi_to_ldb(const struct dsdb_schema *schema,
1320|                                     const struct drsuapi_DsReplicaAttribute *in,
1321|                                     TALLOC_CTX *mem_ctx,
1322|                                     struct ldb_message_element *out)
1323|{
1324|        const struct dsdb_attribute *sa;
1325|
1326|        sa = dsdb_attribute_by_attributeID_id(schema, in->attid);
1327|        if (!sa) {
1328|                return WERR_FOOBAR;
1329|        }
1330|
1331|        return sa->syntax->drsuapi_to_ldb(schema, sa, in, mem_ctx, out);
1332|}
1333|
1334|WERROR dsdb_attribute_ldb_to_drsuapi(const struct dsdb_schema *schema,
1335|                                     const struct ldb_message_element *in,
1336|                                     TALLOC_CTX *mem_ctx,
1337|                                     struct drsuapi_DsReplicaAttribute *out)
1338|{
1339|        const struct dsdb_attribute *sa;
1340|
1341|        sa = dsdb_attribute_by_lDAPDisplayName(schema, in->name);
```

```
1342|        if (!sa) {
1343|                return WERR_FOOBAR;
1344|        }
1345|
1346|        return sa->syntax->ldb_to_drsuapi(schema, sa, in, mem_ctx, out);
1347|}
```

A12. source/dsdb/repl/replicated_objects.c

source/dsdb/repl/replicated_objects.c:

```
   1|/*
   2|   Unix SMB/CIFS mplementation.
   3|   Helper functions for applying replicated objects
   4|
   5|   Copyright (C) Stefan Metzmacher <metze@samba.org> 2007
   6|
   7|   This program is free software; you can redistribute it and/or modify
   8|   it under the terms of the GNU General Public License as published by
   9|   the Free Software Foundation; either version 2 of the License, or
  10|   (at your option) any later version.
  11|
  12|   This program is distributed in the hope that it will be useful,
  13|   but WITHOUT ANY WARRANTY; without even the implied warranty of
  14|   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  15|   GNU General Public License for more details.
  16|
  17|   You should have received a copy of the GNU General Public License
  18|   along with this program; if not, write to the Free Software
  19|   Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
  20|
  21|*/
  22|
  23|#include "includes.h"
  24|#include "dsdb/samdb/samdb.h"
  25|#include "lib/ldb/include/ldb_errors.h"
  26|#include "lib/util/dlinklist.h"
  27|#include "librpc/gen_ndr/ndr_misc.h"
  28|#include "librpc/gen_ndr/ndr_drsuapi.h"
  29|#include "librpc/gen_ndr/ndr_drsblobs.h"
  30|#include "lib/crypto/crypto.h"
  31|#include "libcli/auth/libcli_auth.h"
  32|
  33|static WERROR dsdb_decrypt_attribute_value(TALLOC_CTX *mem_ctx,
  34|                                           const DATA_BLOB *gensec_skey,
  35|                                           bool rid_crypt,
  36|                                           uint32_t rid,
  37|                                           DATA_BLOB *in,
  38|                                           DATA_BLOB *out)
  39|{
  40|        DATA_BLOB confounder;
  41|        DATA_BLOB enc_buffer;
  42|
  43|        struct MD5Context md5;
  44|        uint8_t _enc_key[16];
  45|        DATA_BLOB enc_key;
  46|
  47|        DATA_BLOB dec_buffer;
  48|
  49|        uint32_t crc32_given;
  50|        uint32_t crc32_calc;
  51|        DATA_BLOB checked_buffer;
  52|
  53|        DATA_BLOB plain_buffer;
  54|
  55|        /*
  56|         * users with rid == 0 should not exist
  57|         */
  58|        if (rid_crypt && rid == 0) {
  59|                return WERR_DS_DRA_INVALID_PARAMETER;
  60|        }
  61|
  62|        /*
  63|         * the first 16 bytes at the beginning are the confounder
  64|         * followed by the 4 byte crc32 checksum
  65|         */
  66|        if (in->length < 20) {
  67|                return WERR_DS_DRA_INVALID_PARAMETER;
  68|        }
  69|        confounder = data_blob_const(in->data, 16);
  70|        enc_buffer = data_blob_const(in->data + 16, in->length - 16);
  71|
  72|        /*
  73|         * build the encryption key md5 over the session key followed
  74|         * by the confounder
  75|         *
  76|         * here the gensec session key is used and
  77|         * not the dcerpc ncacn_ip_tcp "SystemLibraryDTC" key!
  78|         */
  79|        enc_key = data_blob_const(_enc_key, sizeof(_enc_key));
```

```
  80|        MD5Init(&md5);
  81|        MD5Update(&md5, gensec_skey->data, gensec_skey->length);
  82|        MD5Update(&md5, confounder.data, confounder.length);
  83|        MD5Final(enc_key.data, &md5);
  84|
  85|        /*
  86|         * copy the encrypted buffer part and
  87|         * decrypt it using the created encryption key using arcfour
  88|         */
  89|        dec_buffer = data_blob_const(enc_buffer.data, enc_buffer.length);
  90|        arcfour_crypt_blob(dec_buffer.data, dec_buffer.length, &enc_key);
  91|
  92|        /*
  93|         * the first 4 byte are the crc32 checksum
  94|         * of the remaining bytes
  95|         */
  96|        crc32_given = IVAL(dec_buffer.data, 0);
  97|        crc32_calc = crc32_calc_buffer(dec_buffer.data + 4 , dec_buffer.length - 4);
  98|        if (crc32_given != crc32_calc) {
  99|                return WERR_SEC_E_DECRYPT_FAILURE;
 100|        }
 101|        checked_buffer = data_blob_const(dec_buffer.data + 4, dec_buffer.length - 4);
 102|
 103|        plain_buffer = data_blob_talloc(mem_ctx, checked_buffer.data,
->    checked_buffer.length);
 104|        W_ERROR_HAVE_NO_MEMORY(plain_buffer.data);
 105|
 106|        /*
 107|         * The following rid_crypt obfuscation isn't session specific
 108|         * and not really needed here, because we allways know the rid of the
 109|         * user account.
 110|         *
 111|         * But for the rest of samba it's easier when we remove this static
 112|         * obfuscation here
 113|         */
 114|        if (rid_crypt) {
 115|                uint32_t i, num_hashes;
 116|
 117|                if ((checked_buffer.length % 16) != 0) {
 118|                        return WERR_DS_DRA_INVALID_PARAMETER;
 119|                }
 120|
 121|                num_hashes = plain_buffer.length / 16;
 122|                for (i = 0; i < num_hashes; i++) {
 123|                        uint32_t offset = i * 16;
 124|                        sam_rid_crypt(rid, checked_buffer.data + offset,
->    plain_buffer.data + offset, 0);
 125|                }
 126|        }
 127|
 128|        *out = plain_buffer;
 129|        return WERR_OK;
 130|}
 131|
 132|static WERROR dsdb_decrypt_attribute(const DATA_BLOB *gensec_skey,
 133|                                     uint32_t rid,
 134|                                     struct drsuapi_DsReplicaAttribute *attr)
 135|{
 136|        WERROR status;
 137|        TALLOC_CTX *mem_ctx;
 138|        DATA_BLOB *enc_data;
 139|        DATA_BLOB plain_data;
 140|        bool rid_crypt = false;
 141|
 142|        if (attr->value_ctr.num_values == 0) {
 143|                return WERR_OK;
 144|        }
 145|
 146|        switch (attr->attid) {
 147|        case DRSUAPI_ATTRIBUTE_dBCSPwd:
 148|        case DRSUAPI_ATTRIBUTE_unicodePwd:
 149|        case DRSUAPI_ATTRIBUTE_ntPwdHistory:
 150|        case DRSUAPI_ATTRIBUTE_lmPwdHistory:
 151|                rid_crypt = true;
 152|                break;
 153|        case DRSUAPI_ATTRIBUTE_supplementalCredentials:
 154|        case DRSUAPI_ATTRIBUTE_priorValue:
 155|        case DRSUAPI_ATTRIBUTE_currentValue:
 156|        case DRSUAPI_ATTRIBUTE_trustAuthOutgoing:
 157|        case DRSUAPI_ATTRIBUTE_trustAuthIncoming:
 158|        case DRSUAPI_ATTRIBUTE_initialAuthOutgoing:
 159|        case DRSUAPI_ATTRIBUTE_initialAuthIncoming:
 160|                break;
 161|        default:
 162|                return WERR_OK;
 163|        }
 164|
```

```
165|        if (attr->value_ctr.num_values > 1) {
166|                return WERR_DS_DRA_INVALID_PARAMETER;
167|        }
168|
169|        if (!attr->value_ctr.values[0].blob) {
170|                return WERR_DS_DRA_INVALID_PARAMETER;
171|        }
172|
173|        mem_ctx         = attr->value_ctr.values[0].blob;
174|        enc_data        = attr->value_ctr.values[0].blob;
175|
176|        status = dsdb_decrypt_attribute_value(mem_ctx,
177|                                            gensec_skey,
178|                                            rid_crypt,
179|                                            rid,
180|                                            enc_data,
181|                                            &plain_data);
182|        W_ERROR_NOT_OK_RETURN(status);
183|
184|        talloc_free(attr->value_ctr.values[0].blob->data);
185|        *attr->value_ctr.values[0].blob = plain_data;
186|
187|        return WERR_OK;
188|}
189|
190|static WERROR dsdb_convert_object(struct ldb_context *ldb,
191|                                  const struct dsdb_schema *schema,
192|                                  struct dsdb_extended_replicated_objects *ctr,
193|                                  const struct drsuapi_DsReplicaObjectListItemEx *in,
194|                                  const DATA_BLOB *gensec_skey,
195|                                  TALLOC_CTX *mem_ctx,
196|                                  struct dsdb_extended_replicated_object *out)
197|{
198|        NTSTATUS nt_status;
199|        WERROR status;
200|        uint32_t i;
201|        struct ldb_message *msg;
202|        struct replPropertyMetaDataBlob *md;
203|        struct ldb_val guid_value;
204|        NTTIME whenChanged = 0;
205|        time_t whenChanged_t;
206|        const char *whenChanged_s;
207|        const char *rdn_name = NULL;
208|        const struct ldb_val *rdn_value = NULL;
209|        const struct dsdb_attribute *rdn_attr = NULL;
210|        uint32_t rdn_attid;
211|        struct drsuapi_DsReplicaAttribute *name_a = NULL;
212|        struct drsuapi_DsReplicaMetaData *name_d = NULL;
213|        struct replPropertyMetaData1 *rdn_m = NULL;
214|        struct dom_sid *sid = NULL;
215|        uint32_t rid = 0;
216|        int ret;
217|
218|        if (!in->object.identifier) {
219|                return WERR_FOOBAR;
220|        }
221|
222|        if (!in->object.identifier->dn || !in->object.identifier->dn[0]) {
223|                return WERR_FOOBAR;
224|        }
225|
226|        if (in->object.attribute_ctr.num_attributes != 0 && !in->meta_data_ctr) {
227|                return WERR_FOOBAR;
228|        }
229|
230|        if (in->object.attribute_ctr.num_attributes != in->meta_data_ctr->count) {
231|                return WERR_FOOBAR;
232|        }
233|
234|        sid = &in->object.identifier->sid;
235|        if (sid->num_auths > 0) {
236|                rid = sid->sub_auths[sid->num_auths - 1];
237|        }
238|
239|        msg = ldb_msg_new(mem_ctx);
240|        W_ERROR_HAVE_NO_MEMORY(msg);
241|
242|        msg->dn                 = ldb_dn_new(msg, ldb, in->object.identifier->dn);
243|        W_ERROR_HAVE_NO_MEMORY(msg->dn);
244|
245|        rdn_name        = ldb_dn_get_rdn_name(msg->dn);
246|        rdn_attr        = dsdb_attribute_by_lDAPDisplayName(schema, rdn_name);
247|        if (!rdn_attr) {
248|                return WERR_FOOBAR;
249|        }
250|        rdn_attid       = rdn_attr->attributeID_id;
251|        rdn_value       = ldb_dn_get_rdn_val(msg->dn);
```

```
252|
253|        msg->num_elements       = in->object.attribute_ctr.num_attributes;
254|        msg->elements           = talloc_array(msg, struct ldb_message_element,
255|                                               msg->num_elements);
256|        W_ERROR_HAVE_NO_MEMORY(msg->elements);
257|
258|        md = talloc(mem_ctx, struct replPropertyMetaDataBlob);
259|        W_ERROR_HAVE_NO_MEMORY(md);
260|
261|        md->version             = 1;
262|        md->reserved            = 0;
263|        md->ctr.ctr1.count      = in->meta_data_ctr->count;
264|        md->ctr.ctr1.reserved   = 0;
265|        md->ctr.ctr1.array      = talloc_array(mem_ctx,
266|                                               struct replPropertyMetaData1,
267|                                               md->ctr.ctr1.count + 1); /* +1 because of
->    the RDN attribute */
268|        W_ERROR_HAVE_NO_MEMORY(md->ctr.ctr1.array);
269|
270|        for (i=0; i < in->meta_data_ctr->count; i++) {
271|                struct drsuapi_DsReplicaAttribute *a;
272|                struct drsuapi_DsReplicaMetaData *d;
273|                struct replPropertyMetaData1 *m;
274|                struct ldb_message_element *e;
275|
276|                a = &in->object.attribute_ctr.attributes[i];
277|                d = &in->meta_data_ctr->meta_data[i];
278|                m = &md->ctr.ctr1.array[i];
279|                e = &msg->elements[i];
280|
281|                status = dsdb_decrypt_attribute(gensec_skey, rid, a);
282|                W_ERROR_NOT_OK_RETURN(status);
283|
284|                status = dsdb_attribute_drsuapi_to_ldb(schema, a, msg->elements, e);
285|                W_ERROR_NOT_OK_RETURN(status);
286|
287|                m->attid                        = a->attid;
288|                m->version                      = d->version;
289|                m->originating_change_time      = d->originating_change_time;
290|                m->originating_invocation_id    = d->originating_invocation_id;
291|                m->originating_usn              = d->originating_usn;
292|                m->local_usn                    = 0;
293|
294|                if (d->originating_change_time > whenChanged) {
295|                        whenChanged = d->originating_change_time;
296|                }
297|
298|                if (a->attid == DRSUAPI_ATTRIBUTE_name) {
299|                        name_a = a;
300|                        name_d = d;
301|                        rdn_m = &md->ctr.ctr1.array[md->ctr.ctr1.count];
302|                }
303|        }
304|
305|        if (rdn_m) {
306|                ret = ldb_msg_add_value(msg, rdn_attr->lDAPDisplayName, rdn_value,
->    NULL);
307|                if (ret != LDB_SUCCESS) {
308|                        return WERR_FOOBAR;
309|                }
310|
311|                rdn_m->attid                            = rdn_attid;
312|                rdn_m->version                          = name_d->version;
313|                rdn_m->originating_change_time          = name_d->originating_change_time
->    ;
314|                rdn_m->originating_invocation_id        = name_d->originating_invocation_
->   id;
315|                rdn_m->originating_usn                  = name_d->originating_usn;
316|                rdn_m->local_usn                        = 0;
317|                md->ctr.ctr1.count++;
318|
319|        }
320|
321|        whenChanged_t = nt_time_to_unix(whenChanged);
322|        whenChanged_s = ldb_timestring(msg, whenChanged_t);
323|        W_ERROR_HAVE_NO_MEMORY(whenChanged_s);
324|
325|        nt_status = ndr_push_struct_blob(&guid_value, msg, &in->object.identifier->guid,
326|                                        (ndr_push_flags_fn_t)ndr_push_GUID);
327|        if (!NT_STATUS_IS_OK(nt_status)) {
328|                return ntstatus_to_werror(nt_status);
329|        }
330|
331|        out->msg                = msg;
332|        out->guid_value         = guid_value;
333|        out->when_changed       = whenChanged_s;
334|        out->meta_data          = md;
```

```
 335|        return WERR_OK;
 336|}
 337|
 338|WERROR dsdb_extended_replicated_objects_commit(struct ldb_context *ldb,
 339|                                               const char *partition_dn,
 340|                                               const struct
->    drsuapi_DsReplicaOIDMapping_Ctr *mapping_ctr,
 341|                                               uint32_t object_count,
 342|                                               const struct
->    drsuapi_DsReplicaObjectListItemEx *first_object,
 343|                                               uint32_t linked_attributes_count,
 344|                                               const struct
->    drsuapi_DsReplicaLinkedAttribute *linked_attributes,
 345|                                               const struct repsFromTo1 *source_dsa,
 346|                                               const struct drsuapi_DsReplicaCursor2CtrEx
->    *uptodateness_vector,
 347|                                               const DATA_BLOB *gensec_skey,
 348|                                               TALLOC_CTX *mem_ctx,
 349|                                               struct dsdb_extended_replicated_objects
->    **_out)
 350|{
 351|        WERROR status;
 352|        const struct dsdb_schema *schema;
 353|        struct dsdb_extended_replicated_objects *out;
 354|        struct ldb_result *ext_res;
 355|        const struct drsuapi_DsReplicaObjectListItemEx *cur;
 356|        uint32_t i;
 357|        int ret;
 358|
 359|        schema = dsdb_get_schema(ldb);
 360|        if (!schema) {
 361|                return WERR_DS_SCHEMA_NOT_LOADED;
 362|        }
 363|
 364|        status = dsdb_verify_oid_mappings_drsuapi(schema, mapping_ctr);
 365|        W_ERROR_NOT_OK_RETURN(status);
 366|
 367|        out = talloc_zero(mem_ctx, struct dsdb_extended_replicated_objects);
 368|        W_ERROR_HAVE_NO_MEMORY(out);
 369|        out->version            = DSDB_EXTENDED_REPLICATED_OBJECTS_VERSION;
 370|
 371|        out->partition_dn       = ldb_dn_new(out, ldb, partition_dn);
 372|        W_ERROR_HAVE_NO_MEMORY(out->partition_dn);
 373|
 374|        out->source_dsa         = source_dsa;
 375|        out->uptodateness_vector= uptodateness_vector;
 376|
 377|        out->num_objects        = object_count;
 378|        out->objects            = talloc_array(out,
 379|                                               struct dsdb_extended_replicated_object,
 380|                                               out->num_objects);
 381|        W_ERROR_HAVE_NO_MEMORY(out->objects);
 382|
 383|        for (i=0, cur = first_object; cur; cur = cur->next_object, i++) {
 384|                if (i == out->num_objects) {
 385|                        return WERR_FOOBAR;
 386|                }
 387|
 388|                status = dsdb_convert_object(ldb, schema, out, cur, gensec_skey,
->    out->objects, &out->objects[i]);
 389|                W_ERROR_NOT_OK_RETURN(status);
 390|        }
 391|        if (i != out->num_objects) {
 392|                return WERR_FOOBAR;
 393|        }
 394|
 395|        /* TODO: handle linked attributes */
 396|
 397|        ret = ldb_extended(ldb, DSDB_EXTENDED_REPLICATED_OBJECTS_OID, out, &ext_res);
 398|        if (ret != LDB_SUCCESS) {
 399|                DEBUG(0,("Failed to apply records: %d: %s\n",
 400|                        ret, ldb_strerror(ret)));
 401|                talloc_free(out);
 402|                return WERR_FOOBAR;
 403|        }
 404|        talloc_free(ext_res);
 405|
 406|        if (_out) {
 407|                *_out = out;
 408|        } else {
 409|                talloc_free(out);
 410|        }
 411|
 412|        return WERR_OK;
 413|}
```

A13. source/dsdb/samdb/ldb_modules/repl_meta_data.c

source/dsdb/samdb/ldb_modules/repl_meta_data.c:

```
   1|/*
   2|   ldb database library
   3|
   4|   Copyright (C) Simo Sorce  2004-2006
   5|   Copyright (C) Andrew Bartlett <abartlet@samba.org> 2005
   6|   Copyright (C) Andrew Tridgell 2005
   7|   Copyright (C) Stefan Metzmacher <metze@samba.org> 2007
   8|
   9|     ** NOTE! The following LGPL license applies to the ldb
  10|     ** library. This does NOT imply that all of Samba is released
  11|     ** under the LGPL
  12|
  13|   This library is free software; you can redistribute it and/or
  14|   modify it under the terms of the GNU Lesser General Public
  15|   License as published by the Free Software Foundation; either
  16|   version 2 of the License, or (at your option) any later version.
  17|
  18|   This library is distributed in the hope that it will be useful,
  19|   but WITHOUT ANY WARRANTY; without even the implied warranty of
  20|   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
  21|   Lesser General Public License for more details.
  22|
  23|   You should have received a copy of the GNU Lesser General Public
  24|   License along with this library; if not, write to the Free Software
  25|   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
  26|*/
  27|
  28|/*
  29| *  Name: ldb
  30| *
  31| *  Component: ldb repl_meta_data module
  32| *
  33| *  Description: - add a unique objectGUID onto every new record,
  34| *               - handle whenCreated, whenChanged timestamps
  35| *               - handle uSNCreated, uSNChanged numbers
  36| *               - handle replPropertyMetaData attribute
  37| *
  38| *  Author: Simo Sorce
  39| *  Author: Stefan Metzmacher
  40| */
  41|
  42|#include "includes.h"
  43|#include "lib/ldb/include/ldb.h"
  44|#include "lib/ldb/include/ldb_errors.h"
  45|#include "lib/ldb/include/ldb_private.h"
  46|#include "dsdb/samdb/samdb.h"
  47|#include "dsdb/common/flags.h"
  48|#include "librpc/gen_ndr/ndr_misc.h"
  49|#include "librpc/gen_ndr/ndr_drsuapi.h"
  50|#include "librpc/gen_ndr/ndr_drsblobs.h"
  51|
  52|struct replmd_replicated_request {
  53|        struct ldb_module *module;
  54|        struct ldb_handle *handle;
  55|        struct ldb_request *orig_req;
  56|
  57|        const struct dsdb_schema *schema;
  58|
  59|        struct dsdb_extended_replicated_objects *objs;
  60|
  61|        uint32_t index_current;
  62|
  63|        struct {
  64|                TALLOC_CTX *mem_ctx;
  65|                struct ldb_request *search_req;
  66|                struct ldb_message *search_msg;
  67|                int search_ret;
  68|                struct ldb_request *change_req;
  69|                int change_ret;
  70|        } sub;
  71|};
  72|
  73|static struct replmd_replicated_request *replmd_replicated_init_handle(struct ldb_module
->    *module,
  74|                                                                        struct ldb_request
->    *req,
  75|                                                                        struct
->    dsdb_extended_replicated_objects *objs)
  76|{
```

```
  77|        struct replmd_replicated_request *ar;
  78|        struct ldb_handle *h;
  79|        const struct dsdb_schema *schema;
  80|
  81|        schema = dsdb_get_schema(module->ldb);
  82|        if (!schema) {
  83|                ldb_debug_set(module->ldb, LDB_DEBUG_FATAL,
  84|                              "replmd_replicated_init_handle: no loaded schema
->    found\n");
  85|                return NULL;
  86|        }
  87|
  88|        h = talloc_zero(req, struct ldb_handle);
  89|        if (h == NULL) {
  90|                ldb_set_errstring(module->ldb, "Out of Memory");
  91|                return NULL;
  92|        }
  93|
  94|        h->module       = module;
  95|        h->state        = LDB_ASYNC_PENDING;
  96|        h->status       = LDB_SUCCESS;
  97|
  98|        ar = talloc_zero(h, struct replmd_replicated_request);
  99|        if (ar == NULL) {
 100|                ldb_set_errstring(module->ldb, "Out of Memory");
 101|                talloc_free(h);
 102|                return NULL;
 103|        }
 104|
 105|        h->private_data = ar;
 106|
 107|        ar->module      = module;
 108|        ar->handle      = h;
 109|        ar->orig_req    = req;
 110|        ar->schema      = schema;
 111|        ar->objs        = objs;
 112|
 113|        req->handle = h;
 114|
 115|        return ar;
 116|}
 117|
 118|/*
 119|  add a time element to a record
 120|*/
 121|static int add_time_element(struct ldb_message *msg, const char *attr, time_t t)
 122|{
 123|        struct ldb_message_element *el;
 124|        char *s;
 125|
 126|        if (ldb_msg_find_element(msg, attr) != NULL) {
 127|                return 0;
 128|        }
 129|
 130|        s = ldb_timestring(msg, t);
 131|        if (s == NULL) {
 132|                return -1;
 133|        }
 134|
 135|        if (ldb_msg_add_string(msg, attr, s) != 0) {
 136|                return -1;
 137|        }
 138|
 139|        el = ldb_msg_find_element(msg, attr);
 140|        /* always set as replace. This works because on add ops, the flag
 141|           is ignored */
 142|        el->flags = LDB_FLAG_MOD_REPLACE;
 143|
 144|        return 0;
 145|}
 146|
 147|/*
 148|  add a uint64_t element to a record
 149|*/
 150|static int add_uint64_element(struct ldb_message *msg, const char *attr, uint64_t v)
 151|{
 152|        struct ldb_message_element *el;
 153|
 154|        if (ldb_msg_find_element(msg, attr) != NULL) {
 155|                return 0;
 156|        }
 157|
 158|        if (ldb_msg_add_fmt(msg, attr, "%llu", (unsigned long long)v) != 0) {
 159|                return -1;
 160|        }
 161|
 162|        el = ldb_msg_find_element(msg, attr);
```

```
163|        /* always set as replace. This works because on add ops, the flag
164|           is ignored */
165|        el->flags = LDB_FLAG_MOD_REPLACE;
166|
167|        return 0;
168|}
169|
170|static int replmd_replPropertyMetaData1_attid_sort(const struct replPropertyMetaData1
->    *m1,
171|                                                   const struct replPropertyMetaData1
->    *m2,
172|                                                   const uint32_t *rdn_attid)
173|{
174|        if (m1->attid == m2->attid) {
175|                return 0;
176|        }
177|
178|        /*
179|         * the rdn attribute should be at the end!
180|         * so we need to return a value greater than zero
181|         * which means m1 is greater than m2
182|         */
183|        if (m1->attid == *rdn_attid) {
184|                return 1;
185|        }
186|
187|        /*
188|         * the rdn attribute should be at the end!
189|         * so we need to return a value less than zero
190|         * which means m2 is greater than m1
191|         */
192|        if (m2->attid == *rdn_attid) {
193|                return -1;
194|        }
195|
196|        return m1->attid - m2->attid;
197|}
198|
199|static void replmd_replPropertyMetaDataCtr1_sort(struct replPropertyMetaDataCtr1 *ctr1,
200|                                                 const uint32_t *rdn_attid)
201|{
202|        ldb_qsort(ctr1->array, ctr1->count, sizeof(struct replPropertyMetaData1),
203|                  discard_const_p(void, rdn_attid), (ldb_qsort_cmp_fn_t)replmd_replProper
->   tyMetaData1_attid_sort);
204|}
205|
206|static int replmd_ldb_message_element_attid_sort(const struct ldb_message_element *e1,
207|                                                 const struct ldb_message_element *e2,
208|                                                 const struct dsdb_schema *schema)
209|{
210|        const struct dsdb_attribute *a1;
211|        const struct dsdb_attribute *a2;
212|
213|        /*
214|         * TODO: make this faster by caching the dsdb_attribute pointer
215|         *       on the ldb_messag_element
216|         */
217|
218|        a1 = dsdb_attribute_by_lDAPDisplayName(schema, e1->name);
219|        a2 = dsdb_attribute_by_lDAPDisplayName(schema, e2->name);
220|
221|        /*
222|         * TODO: remove this check, we should rely on e1 and e2 having valid attribute
->    names
223|         *       in the schema
224|         */
225|        if (!a1 || !a2) {
226|                return strcasecmp(e1->name, e2->name);
227|        }
228|
229|        return a1->attributeID_id - a2->attributeID_id;
230|}
231|
232|static void replmd_ldb_message_sort(struct ldb_message *msg,
233|                                    const struct dsdb_schema *schema)
234|{
235|        ldb_qsort(msg->elements, msg->num_elements, sizeof(struct ldb_message_element),
236|                  discard_const_p(void, schema), (ldb_qsort_cmp_fn_t)replmd_ldb_message_e
->   lement_attid_sort);
237|}
238|
239|static int replmd_prepare_originating(struct ldb_module *module, struct ldb_request
->    *req,
240|                                      struct ldb_dn *dn, const char *fn_name,
241|                                      int (*fn)(struct ldb_module *,
242|                                                struct ldb_request *,
243|                                                const struct dsdb_schema *,
```

```
 244|                                                const struct
->    dsdb_control_current_partition *))
 245|{
 246|        const struct dsdb_schema *schema;
 247|        const struct ldb_control *partition_ctrl;
 248|        const struct dsdb_control_current_partition *partition;
 249|
 250|        /* do not manipulate our control entries */
 251|        if (ldb_dn_is_special(dn)) {
 252|                return ldb_next_request(module, req);
 253|        }
 254|
 255|        schema = dsdb_get_schema(module->ldb);
 256|        if (!schema) {
 257|                ldb_debug_set(module->ldb, LDB_DEBUG_FATAL,
 258|                              "%s: no dsdb_schema loaded",
 259|                              fn_name);
 260|                return LDB_ERR_CONSTRAINT_VIOLATION;
 261|        }
 262|
 263|        partition_ctrl = ldb_request_get_control(req, DSDB_CONTROL_CURRENT_PARTITION_OID)
->    ;
 264|        if (!partition_ctrl) {
 265|                ldb_debug_set(module->ldb, LDB_DEBUG_FATAL,
 266|                              "%s: no current partition control found",
 267|                              fn_name);
 268|                return LDB_ERR_CONSTRAINT_VIOLATION;
 269|        }
 270|
 271|        partition = talloc_get_type(partition_ctrl->data,
 272|                                     struct dsdb_control_current_partition);
 273|        if (!partition) {
 274|                ldb_debug_set(module->ldb, LDB_DEBUG_FATAL,
 275|                              "%s: current partition control contains invalid data",
 276|                              fn_name);
 277|                return LDB_ERR_CONSTRAINT_VIOLATION;
 278|        }
 279|
 280|        if (partition->version != DSDB_CONTROL_CURRENT_PARTITION_VERSION) {
 281|                ldb_debug_set(module->ldb, LDB_DEBUG_FATAL,
 282|                              "%s: current partition control contains invalid version [%u
->    != %u]\n",
 283|                              fn_name, partition->version,
->    DSDB_CONTROL_CURRENT_PARTITION_VERSION);
 284|                return LDB_ERR_CONSTRAINT_VIOLATION;
 285|        }
 286|
 287|        return fn(module, req, schema, partition);
 288|}
 289|
 290|static int replmd_add_originating(struct ldb_module *module,
 291|                                   struct ldb_request *req,
 292|                                   const struct dsdb_schema *schema,
 293|                                   const struct dsdb_control_current_partition
->    *partition)
 294|{
 295|        NTSTATUS nt_status;
 296|        struct ldb_request *down_req;
 297|        struct ldb_message *msg;
 298|        uint32_t instance_type;
 299|        struct ldb_dn *new_dn;
 300|        const char *rdn_name;
 301|        const char *rdn_name_upper;
 302|        const struct ldb_val *rdn_value = NULL;
 303|        const struct dsdb_attribute *rdn_attr = NULL;
 304|        struct GUID guid;
 305|        struct ldb_val guid_value;
 306|        struct replPropertyMetaDataBlob nmd;
 307|        struct ldb_val nmd_value;
 308|        uint64_t seq_num;
 309|        const struct GUID *our_invocation_id;
 310|        time_t t = time(NULL);
 311|        NTTIME now;
 312|        char *time_str;
 313|        int ret;
 314|        uint32_t i, ni=0;
 315|
 316|        ldb_debug(module->ldb, LDB_DEBUG_TRACE, "replmd_add_originating\n");
 317|
 318|        if (ldb_msg_find_element(req->op.add.message, "objectGUID")) {
 319|                ldb_debug_set(module->ldb, LDB_DEBUG_ERROR,
 320|                              "replmd_add_originating: it's not allowed to add an object
->    with objectGUID\n");
 321|                return LDB_ERR_UNWILLING_TO_PERFORM;
 322|        }
 323|
 324|        if (ldb_msg_find_element(req->op.add.message, "instanceType")) {
```

```
325|                ldb_debug_set(module->ldb, LDB_DEBUG_ERROR,
326|                              "replmd_add_originating: it's not allowed to add an object
->    with instanceType\n");
327|                return LDB_ERR_UNWILLING_TO_PERFORM;
328|        }
329|
330|        /* Get a sequence number from the backend */
331|        ret = ldb_sequence_number(module->ldb, LDB_SEQ_NEXT, &seq_num);
332|        if (ret != LDB_SUCCESS) {
333|                return ret;
334|        }
335|
336|        /* a new GUID */
337|        guid = GUID_random();
338|
339|        /* get our invicationId */
340|        our_invocation_id = samdb_ntds_invocation_id(module->ldb);
341|        if (!our_invocation_id) {
342|                ldb_debug_set(module->ldb, LDB_DEBUG_ERROR,
343|                              "replmd_add_originating: unable to find invocationId\n");
344|                return LDB_ERR_OPERATIONS_ERROR;
345|        }
346|
347|        /* create a copy of the request */
348|        down_req = talloc(req, struct ldb_request);
349|        if (down_req == NULL) {
350|                ldb_oom(module->ldb);
351|                return LDB_ERR_OPERATIONS_ERROR;
352|        }
353|        *down_req = *req;
354|
355|        /* we have to copy the message as the caller might have it as a const */
356|        down_req->op.add.message = msg = ldb_msg_copy_shallow(down_req,
->    req->op.add.message);
357|        if (msg == NULL) {
358|                talloc_free(down_req);
359|                ldb_oom(module->ldb);
360|                return LDB_ERR_OPERATIONS_ERROR;
361|        }
362|
363|        /* generated times */
364|        unix_to_nt_time(&now, t);
365|        time_str = ldb_timestring(msg, t);
366|        if (!time_str) {
367|                talloc_free(down_req);
368|                return LDB_ERR_OPERATIONS_ERROR;
369|        }
370|
371|        /*
372|         * get details of the rdn name
373|         */
374|        rdn_name        = ldb_dn_get_rdn_name(msg->dn);
375|        if (!rdn_name) {
376|                talloc_free(down_req);
377|                ldb_oom(module->ldb);
378|                return LDB_ERR_OPERATIONS_ERROR;
379|        }
380|        rdn_attr        = dsdb_attribute_by_lDAPDisplayName(schema, rdn_name);
381|        if (!rdn_attr) {
382|                talloc_free(down_req);
383|                return LDB_ERR_OPERATIONS_ERROR;
384|        }
385|        rdn_value       = ldb_dn_get_rdn_val(msg->dn);
386|        if (!rdn_value) {
387|                talloc_free(down_req);
388|                ldb_oom(module->ldb);
389|                return LDB_ERR_OPERATIONS_ERROR;
390|        }
391|
392|        /*
393|         * remove autogenerated attributes
394|         */
395|        ldb_msg_remove_attr(msg, rdn_name);
396|        ldb_msg_remove_attr(msg, "name");
397|        ldb_msg_remove_attr(msg, "whenCreated");
398|        ldb_msg_remove_attr(msg, "whenChanged");
399|        ldb_msg_remove_attr(msg, "uSNCreated");
400|        ldb_msg_remove_attr(msg, "uSNChanged");
401|        ldb_msg_remove_attr(msg, "replPropertyMetaData");
402|
403|        /*
404|         * TODO: construct a new DN out of:
405|         *       - the parent DN
406|         *       - the upper case of rdn_attr->LDAPDisplayName
407|         *       - rdn_value
408|         */
409|        new_dn = ldb_dn_copy(msg, msg->dn);
```

```
410|        if (!new_dn) {
411|                talloc_free(down_req);
412|                ldb_oom(module->ldb);
413|                return LDB_ERR_OPERATIONS_ERROR;
414|        }
415|        rdn_name_upper = strupper_talloc(msg, rdn_attr->lDAPDisplayName);
416|        if (!rdn_name_upper) {
417|                talloc_free(down_req);
418|                ldb_oom(module->ldb);
419|                return LDB_ERR_OPERATIONS_ERROR;
420|        }
421|        ret = ldb_dn_set_component(new_dn, 0, rdn_name_upper, *rdn_value);
422|        if (ret != LDB_SUCCESS) {
423|                talloc_free(down_req);
424|                ldb_oom(module->ldb);
425|                return LDB_ERR_OPERATIONS_ERROR;
426|        }
427|        msg->dn = new_dn;
428|
429|        /*
430|         * TODO: calculate correct instance type
431|         */
432|        instance_type = INSTANCE_TYPE_WRITE;
433|        if (ldb_dn_compare(partition->dn, msg->dn) == 0) {
434|                instance_type |= INSTANCE_TYPE_IS_NC_HEAD;
435|                if (ldb_dn_compare(msg->dn, samdb_base_dn(module->ldb)) != 0) {
436|                        instance_type |= INSTANCE_TYPE_NC_ABOVE;
437|                }
438|        }
439|
440|        /*
441|         * readd replicated attributes
442|         */
443|        ret = ldb_msg_add_value(msg, rdn_attr->lDAPDisplayName, rdn_value, NULL);
444|        if (ret != LDB_SUCCESS) {
445|                talloc_free(down_req);
446|                ldb_oom(module->ldb);
447|                return LDB_ERR_OPERATIONS_ERROR;
448|        }
449|        ret = ldb_msg_add_value(msg, "name", rdn_value, NULL);
450|        if (ret != LDB_SUCCESS) {
451|                talloc_free(down_req);
452|                ldb_oom(module->ldb);
453|                return LDB_ERR_OPERATIONS_ERROR;
454|        }
455|        ret = ldb_msg_add_string(msg, "whenCreated", time_str);
456|        if (ret != LDB_SUCCESS) {
457|                talloc_free(down_req);
458|                ldb_oom(module->ldb);
459|                return LDB_ERR_OPERATIONS_ERROR;
460|        }
461|        ret = ldb_msg_add_fmt(msg, "instanceType", "%u", instance_type);
462|        if (ret != LDB_SUCCESS) {
463|                talloc_free(down_req);
464|                ldb_oom(module->ldb);
465|                return LDB_ERR_OPERATIONS_ERROR;
466|        }
467|
468|        /* build the replication meta_data */
469|        ZERO_STRUCT(nmd);
470|        nmd.version             = 1;
471|        nmd.ctr.ctr1.count      = msg->num_elements;
472|        nmd.ctr.ctr1.array      = talloc_array(msg,
473|                                               struct replPropertyMetaData1,
474|                                               nmd.ctr.ctr1.count);
475|        if (!nmd.ctr.ctr1.array) {
476|                talloc_free(down_req);
477|                ldb_oom(module->ldb);
478|                return LDB_ERR_OPERATIONS_ERROR;
479|        }
480|
481|        for (i=0; i < msg->num_elements; i++) {
482|                struct ldb_message_element *e = &msg->elements[i];
483|                struct replPropertyMetaData1 *m = &nmd.ctr.ctr1.array[ni];
484|                const struct dsdb_attribute *sa;
485|
486|                sa = dsdb_attribute_by_lDAPDisplayName(schema, e->name);
487|                if (!sa) {
488|                        ldb_debug_set(module->ldb, LDB_DEBUG_ERROR,
489|                                      "replmd_add_originating: attribute '%s' not defined
->    in schema\n",
490|                                      e->name);
491|                        talloc_free(down_req);
492|                        return LDB_ERR_NO_SUCH_ATTRIBUTE;
493|                }
494|
495|                if ((sa->systemFlags & 0x00000001) || (sa->systemFlags & 0x00000004)) {
```

```
496|                       /* if the attribute is not replicated (0x00000001)
497|                        * or constructed (0x00000004) it has no metadata
498|                        */
499|                       continue;
500|               }
501|
502|               m->attid                       = sa->attributeID_id;
503|               m->version                      = 1;
504|               m->originating_change_time     = now;
505|               m->originating_invocation_id   = *our_invocation_id;
506|               m->originating_usn             = seq_num;
507|               m->local_usn                   = seq_num;
508|               ni++;
509|       }
510|
511|       /* fix meta data count */
512|       nmd.ctr.ctr1.count = ni;
513|
514|       /*
515|        * sort meta data array, and move the rdn attribute entry to the end
516|        */
517|       replmd_replPropertyMetaDataCtr1_sort(&nmd.ctr.ctr1, &rdn_attr->attributeID_id);
518|
519|       /* generated NDR encoded values */
520|       nt_status = ndr_push_struct_blob(&guid_value, msg, &guid,
521|                                        (ndr_push_flags_fn_t)ndr_push_GUID);
522|       if (!NT_STATUS_IS_OK(nt_status)) {
523|               talloc_free(down_req);
524|               ldb_oom(module->ldb);
525|               return LDB_ERR_OPERATIONS_ERROR;
526|       }
527|       nt_status = ndr_push_struct_blob(&nmd_value, msg, &nmd,
528|                                        (ndr_push_flags_fn_t)ndr_push_replPropertyMetaDa
->   taBlob);
529|       if (!NT_STATUS_IS_OK(nt_status)) {
530|               talloc_free(down_req);
531|               ldb_oom(module->ldb);
532|               return LDB_ERR_OPERATIONS_ERROR;
533|       }
534|
535|       /*
536|        * add the autogenerated values
537|        */
538|       ret = ldb_msg_add_value(msg, "objectGUID", &guid_value, NULL);
539|       if (ret != LDB_SUCCESS) {
540|               talloc_free(down_req);
541|               ldb_oom(module->ldb);
542|               return LDB_ERR_OPERATIONS_ERROR;
543|       }
544|       ret = ldb_msg_add_string(msg, "whenChanged", time_str);
545|       if (ret != LDB_SUCCESS) {
546|               talloc_free(down_req);
547|               ldb_oom(module->ldb);
548|               return LDB_ERR_OPERATIONS_ERROR;
549|       }
550|       ret = samdb_msg_add_uint64(module->ldb, msg, msg, "uSNCreated", seq_num);
551|       if (ret != LDB_SUCCESS) {
552|               talloc_free(down_req);
553|               ldb_oom(module->ldb);
554|               return LDB_ERR_OPERATIONS_ERROR;
555|       }
556|       ret = samdb_msg_add_uint64(module->ldb, msg, msg, "uSNChanged", seq_num);
557|       if (ret != LDB_SUCCESS) {
558|               talloc_free(down_req);
559|               ldb_oom(module->ldb);
560|               return LDB_ERR_OPERATIONS_ERROR;
561|       }
562|       ret = ldb_msg_add_value(msg, "replPropertyMetaData", &nmd_value, NULL);
563|       if (ret != LDB_SUCCESS) {
564|               talloc_free(down_req);
565|               ldb_oom(module->ldb);
566|               return LDB_ERR_OPERATIONS_ERROR;
567|       }
568|
569|       /*
570|        * sort the attributes by attid before storing the object
571|        */
572|       replmd_ldb_message_sort(msg, schema);
573|
574|       ldb_set_timeout_from_prev_req(module->ldb, req, down_req);
575|
576|       /* go on with the call chain */
577|       ret = ldb_next_request(module, down_req);
578|
579|       /* do not free down_req as the call results may be linked to it,
580|        * it will be freed when the upper level request get freed */
581|       if (ret == LDB_SUCCESS) {
```

```
 582|                req->handle = down_req->handle;
 583|        }
 584|
 585|        return ret;
 586|}
 587|
 588|static int replmd_add(struct ldb_module *module, struct ldb_request *req)
 589|{
 590|        return replmd_prepare_originating(module, req, req->op.add.message->dn,
 591|                                           "replmd_add", replmd_add_originating);
 592|}
 593|
 594|static int replmd_modify_originating(struct ldb_module *module,
 595|                                     struct ldb_request *req,
 596|                                     const struct dsdb_schema *schema,
 597|                                     const struct dsdb_control_current_partition
->    *partition)
 598|{
 599|        struct ldb_request *down_req;
 600|        struct ldb_message *msg;
 601|        int ret;
 602|        time_t t = time(NULL);
 603|        uint64_t seq_num;
 604|
 605|        ldb_debug(module->ldb, LDB_DEBUG_TRACE, "replmd_modify_originating\n");
 606|
 607|        down_req = talloc(req, struct ldb_request);
 608|        if (down_req == NULL) {
 609|                return LDB_ERR_OPERATIONS_ERROR;
 610|        }
 611|
 612|        *down_req = *req;
 613|
 614|        /* we have to copy the message as the caller might have it as a const */
 615|        down_req->op.mod.message = msg = ldb_msg_copy_shallow(down_req,
->    req->op.mod.message);
 616|        if (msg == NULL) {
 617|                talloc_free(down_req);
 618|                return LDB_ERR_OPERATIONS_ERROR;
 619|        }
 620|
 621|        if (add_time_element(msg, "whenChanged", t) != 0) {
 622|                talloc_free(down_req);
 623|                return LDB_ERR_OPERATIONS_ERROR;
 624|        }
 625|
 626|        /* Get a sequence number from the backend */
 627|        ret = ldb_sequence_number(module->ldb, LDB_SEQ_NEXT, &seq_num);
 628|        if (ret == LDB_SUCCESS) {
 629|                if (add_uint64_element(msg, "uSNChanged", seq_num) != 0) {
 630|                        talloc_free(down_req);
 631|                        return LDB_ERR_OPERATIONS_ERROR;
 632|                }
 633|        }
 634|
 635|        ldb_set_timeout_from_prev_req(module->ldb, req, down_req);
 636|
 637|        /* go on with the call chain */
 638|        ret = ldb_next_request(module, down_req);
 639|
 640|        /* do not free down_req as the call results may be linked to it,
 641|         * it will be freed when the upper level request get freed */
 642|        if (ret == LDB_SUCCESS) {
 643|                req->handle = down_req->handle;
 644|        }
 645|
 646|        return ret;
 647|}
 648|
 649|static int replmd_modify(struct ldb_module *module, struct ldb_request *req)
 650|{
 651|        return replmd_prepare_originating(module, req, req->op.mod.message->dn,
 652|                                           "replmd_modify", replmd_modify_originating);
 653|}
 654|
 655|static int replmd_replicated_request_reply_helper(struct replmd_replicated_request *ar,
->    int ret)
 656|{
 657|        struct ldb_reply *ares = NULL;
 658|
 659|        ar->handle->status = ret;
 660|        ar->handle->state = LDB_ASYNC_DONE;
 661|
 662|        if (!ar->orig_req->callback) {
 663|                return LDB_SUCCESS;
 664|        }
 665|
```

```
 666|        /* we're done and need to report the success to the caller */
 667|        ares = talloc_zero(ar, struct ldb_reply);
 668|        if (!ares) {
 669|                ar->handle->status = LDB_ERR_OPERATIONS_ERROR;
 670|                ar->handle->state = LDB_ASYNC_DONE;
 671|                return LDB_ERR_OPERATIONS_ERROR;
 672|        }
 673|
 674|        ares->type      = LDB_REPLY_EXTENDED;
 675|        ares->response  = NULL;
 676|
 677|        return ar->orig_req->callback(ar->module->ldb, ar->orig_req->context, ares);
 678|}
 679|
 680|static int replmd_replicated_request_done(struct replmd_replicated_request *ar)
 681|{
 682|        return replmd_replicated_request_reply_helper(ar, LDB_SUCCESS);
 683|}
 684|
 685|static int replmd_replicated_request_error(struct replmd_replicated_request *ar, int
->    ret)
 686|{
 687|        return replmd_replicated_request_reply_helper(ar, ret);
 688|}
 689|
 690|static int replmd_replicated_request_werror(struct replmd_replicated_request *ar, WERROR
->    status)
 691|{
 692|        int ret = LDB_ERR_OTHER;
 693|        /* TODO: do some error mapping */
 694|        return replmd_replicated_request_reply_helper(ar, ret);
 695|}
 696|
 697|static int replmd_replicated_apply_next(struct replmd_replicated_request *ar);
 698|
 699|static int replmd_replicated_apply_add_callback(struct ldb_context *ldb,
 700|                                                 void *private_data,
 701|                                                 struct ldb_reply *ares)
 702|{
 703|#ifdef REPLMD_FULL_ASYNC /* TODO: activate this code when ldb support full async code */
->
 704|        struct replmd_replicated_request *ar = talloc_get_type(private_data,
 705|                                                struct replmd_replicated_request);
 706|
 707|        ar->sub.change_ret = ldb_wait(ar->sub.search_req->handle, LDB_WAIT_ALL);
 708|        if (ar->sub.change_ret != LDB_SUCCESS) {
 709|                return replmd_replicated_request_error(ar, ar->sub.change_ret);
 710|        }
 711|
 712|        talloc_free(ar->sub.mem_ctx);
 713|        ZERO_STRUCT(ar->sub);
 714|
 715|        ar->index_current++;
 716|
 717|        return replmd_replicated_apply_next(ar);
 718|#else
 719|        return LDB_SUCCESS;
 720|#endif
 721|}
 722|
 723|static int replmd_replicated_apply_add(struct replmd_replicated_request *ar)
 724|{
 725|        NTSTATUS nt_status;
 726|        struct ldb_message *msg;
 727|        struct replPropertyMetaDataBlob *md;
 728|        struct ldb_val md_value;
 729|        uint32_t i;
 730|        uint64_t seq_num;
 731|        int ret;
 732|
 733|        /*
 734|         * TODO: check if the parent object exist
 735|         */
 736|
 737|        /*
 738|         * TODO: handle the conflict case where an object with the
 739|         *       same name exist
 740|         */
 741|
 742|        msg = ar->objs->objects[ar->index_current].msg;
 743|        md = ar->objs->objects[ar->index_current].meta_data;
 744|
 745|        ret = ldb_sequence_number(ar->module->ldb, LDB_SEQ_NEXT, &seq_num);
 746|        if (ret != LDB_SUCCESS) {
 747|                return replmd_replicated_request_error(ar, ret);
 748|        }
 749|
```

```
 750|        ret = ldb_msg_add_value(msg, "objectGUID", &ar->objs->objects[ar->index_current].
->   guid_value, NULL);
 751|        if (ret != LDB_SUCCESS) {
 752|                return replmd_replicated_request_error(ar, ret);
 753|        }
 754|
 755|        ret = ldb_msg_add_string(msg, "whenChanged", ar->objs->objects[ar->index_current]
->   .when_changed);
 756|        if (ret != LDB_SUCCESS) {
 757|                return replmd_replicated_request_error(ar, ret);
 758|        }
 759|
 760|        ret = samdb_msg_add_uint64(ar->module->ldb, msg, msg, "uSNCreated", seq_num);
 761|        if (ret != LDB_SUCCESS) {
 762|                return replmd_replicated_request_error(ar, ret);
 763|        }
 764|
 765|        ret = samdb_msg_add_uint64(ar->module->ldb, msg, msg, "uSNChanged", seq_num);
 766|        if (ret != LDB_SUCCESS) {
 767|                return replmd_replicated_request_error(ar, ret);
 768|        }
 769|
 770|        /*
 771|         * the meta data array is already sorted by the caller
 772|         */
 773|        for (i=0; i < md->ctr.ctr1.count; i++) {
 774|                md->ctr.ctr1.array[i].local_usn = seq_num;
 775|        }
 776|        nt_status = ndr_push_struct_blob(&md_value, msg, md,
 777|                                          (ndr_push_flags_fn_t)ndr_push_replPropertyMetaDa
->   taBlob);
 778|        if (!NT_STATUS_IS_OK(nt_status)) {
 779|                return replmd_replicated_request_werror(ar,
->    ntstatus_to_werror(nt_status));
 780|        }
 781|        ret = ldb_msg_add_value(msg, "replPropertyMetaData", &md_value, NULL);
 782|        if (ret != LDB_SUCCESS) {
 783|                return replmd_replicated_request_error(ar, ret);
 784|        }
 785|
 786|        replmd_ldb_message_sort(msg, ar->schema);
 787|
 788|        ret = ldb_build_add_req(&ar->sub.change_req,
 789|                                 ar->module->ldb,
 790|                                 ar->sub.mem_ctx,
 791|                                 msg,
 792|                                 NULL,
 793|                                 ar,
 794|                                 replmd_replicated_apply_add_callback);
 795|        if (ret != LDB_SUCCESS) return replmd_replicated_request_error(ar, ret);
 796|
 797|#ifdef REPLMD_FULL_ASYNC /* TODO: activate this code when ldb support full async code */
->
 798|        return ldb_next_request(ar->module, ar->sub.change_req);
 799|#else
 800|        ret = ldb_next_request(ar->module, ar->sub.change_req);
 801|        if (ret != LDB_SUCCESS) return replmd_replicated_request_error(ar, ret);
 802|
 803|        ar->sub.change_ret = ldb_wait(ar->sub.search_req->handle, LDB_WAIT_ALL);
 804|        if (ar->sub.change_ret != LDB_SUCCESS) {
 805|                return replmd_replicated_request_error(ar, ar->sub.change_ret);
 806|        }
 807|
 808|        talloc_free(ar->sub.mem_ctx);
 809|        ZERO_STRUCT(ar->sub);
 810|
 811|        ar->index_current++;
 812|
 813|        return LDB_SUCCESS;
 814|#endif
 815|}
 816|
 817|static int replmd_replPropertyMetaData1_conflict_compare(struct replPropertyMetaData1
->    *m1,
 818|                                                         struct replPropertyMetaData1
->    *m2)
 819|{
 820|        int ret;
 821|
 822|        if (m1->version != m2->version) {
 823|                return m1->version - m2->version;
 824|        }
 825|
 826|        if (m1->originating_change_time != m2->originating_change_time) {
 827|                return m1->originating_change_time - m2->originating_change_time;
 828|        }
 829|
```

```
 830|        ret = GUID_compare(&m1->originating_invocation_id,
->    &m2->originating_invocation_id);
 831|        if (ret != 0) {
 832|                return ret;
 833|        }
 834|
 835|        return m1->originating_usn - m2->originating_usn;
 836|}
 837|
 838|static int replmd_replicated_apply_merge_callback(struct ldb_context *ldb,
 839|                                                  void *private_data,
 840|                                                  struct ldb_reply *ares)
 841|{
 842|#ifdef REPLMD_FULL_ASYNC /* TODO: activate this code when ldb support full async code */
->
 843|        struct replmd_replicated_request *ar = talloc_get_type(private_data,
 844|                                                 struct replmd_replicated_request);
 845|
 846|        ret = ldb_next_request(ar->module, ar->sub.change_req);
 847|        if (ret != LDB_SUCCESS) return replmd_replicated_request_error(ar, ret);
 848|
 849|        ar->sub.change_ret = ldb_wait(ar->sub.search_req->handle, LDB_WAIT_ALL);
 850|        if (ar->sub.change_ret != LDB_SUCCESS) {
 851|                return replmd_replicated_request_error(ar, ar->sub.change_ret);
 852|        }
 853|
 854|        talloc_free(ar->sub.mem_ctx);
 855|        ZERO_STRUCT(ar->sub);
 856|
 857|        ar->index_current++;
 858|
 859|        return LDB_SUCCESS;
 860|#else
 861|        return LDB_SUCCESS;
 862|#endif
 863|}
 864|
 865|static int replmd_replicated_apply_merge(struct replmd_replicated_request *ar)
 866|{
 867|        NTSTATUS nt_status;
 868|        struct ldb_message *msg;
 869|        struct replPropertyMetaDataBlob *rmd;
 870|        struct replPropertyMetaDataBlob omd;
 871|        const struct ldb_val *omd_value;
 872|        struct replPropertyMetaDataBlob nmd;
 873|        struct ldb_val nmd_value;
 874|        uint32_t i,j,ni=0;
 875|        uint32_t removed_attrs = 0;
 876|        uint64_t seq_num;
 877|        int ret;
 878|
 879|        msg = ar->objs->objects[ar->index_current].msg;
 880|        rmd = ar->objs->objects[ar->index_current].meta_data;
 881|        ZERO_STRUCT(omd);
 882|        omd.version = 1;
 883|
 884|        /*
 885|         * TODO: add rename conflict handling
 886|         */
 887|        if (ldb_dn_compare(msg->dn, ar->sub.search_msg->dn) != 0) {
 888|                ldb_debug_set(ar->module->ldb, LDB_DEBUG_FATAL,
->    "replmd_replicated_apply_merge[%u]: rename not supported",
 889|                              ar->index_current);
 890|                ldb_debug(ar->module->ldb, LDB_DEBUG_FATAL, "%s => %s\n",
 891|                          ldb_dn_get_linearized(ar->sub.search_msg->dn),
 892|                          ldb_dn_get_linearized(msg->dn));
 893|                return replmd_replicated_request_werror(ar, WERR_NOT_SUPPORTED);
 894|        }
 895|
 896|        ret = ldb_sequence_number(ar->module->ldb, LDB_SEQ_NEXT, &seq_num);
 897|        if (ret != LDB_SUCCESS) {
 898|                return replmd_replicated_request_error(ar, ret);
 899|        }
 900|
 901|        /* find existing meta data */
 902|        omd_value = ldb_msg_find_ldb_val(ar->sub.search_msg, "replPropertyMetaData");
 903|        if (omd_value) {
 904|                nt_status = ndr_pull_struct_blob(omd_value, ar->sub.mem_ctx, &omd,
 905|                                                 (ndr_pull_flags_fn_t)ndr_pull_replProper
->   tyMetaDataBlob);
 906|                if (!NT_STATUS_IS_OK(nt_status)) {
 907|                        return replmd_replicated_request_werror(ar,
->    ntstatus_to_werror(nt_status));
 908|                }
 909|
 910|                if (omd.version != 1) {
 911|                        return replmd_replicated_request_werror(ar,
```

```
->    WERR_DS_DRA_INTERNAL_ERROR);
 912|                }
 913|        }
 914|
 915|        ZERO_STRUCT(nmd);
 916|        nmd.version = 1;
 917|        nmd.ctr.ctr1.count = omd.ctr.ctr1.count + rmd->ctr.ctr1.count;
 918|        nmd.ctr.ctr1.array = talloc_array(ar->sub.mem_ctx,
 919|                                         struct replPropertyMetaData1,
 920|                                         nmd.ctr.ctr1.count);
 921|        if (!nmd.ctr.ctr1.array) return replmd_replicated_request_werror(ar,
->    WERR_NOMEM);
 922|
 923|        /* first copy the old meta data */
 924|        for (i=0; i < omd.ctr.ctr1.count; i++) {
 925|                nmd.ctr.ctr1.array[ni]  = omd.ctr.ctr1.array[i];
 926|                ni++;
 927|        }
 928|
 929|        /* now merge in the new meta data */
 930|        for (i=0; i < rmd->ctr.ctr1.count; i++) {
 931|                bool found = false;
 932|
 933|                rmd->ctr.ctr1.array[i].local_usn = seq_num;
 934|
 935|                for (j=0; j < ni; j++) {
 936|                        int cmp;
 937|
 938|                        if (rmd->ctr.ctr1.array[i].attid != nmd.ctr.ctr1.array[j].attid)
->    {
 939|                                continue;
 940|                        }
 941|
 942|                        cmp = replmd_replPropertyMetaData1_conflict_compare(&rmd->ctr.ctr
->   1.array[i],
 943|
->    &nmd.ctr.ctr1.array[j]);
 944|                        if (cmp > 0) {
 945|                                /* replace the entry */
 946|                                nmd.ctr.ctr1.array[j] = rmd->ctr.ctr1.array[i];
 947|                                found = true;
 948|                                break;
 949|                        }
 950|
 951|                        /* we don't want to apply this change so remove the attribute */
 952|                        ldb_msg_remove_element(msg, &msg->elements[i-removed_attrs]);
 953|                        removed_attrs++;
 954|
 955|                        found = true;
 956|                        break;
 957|                }
 958|
 959|                if (found) continue;
 960|
 961|                nmd.ctr.ctr1.array[ni] = rmd->ctr.ctr1.array[i];
 962|                ni++;
 963|        }
 964|
 965|        /*
 966|         * finally correct the size of the meta_data array
 967|         */
 968|        nmd.ctr.ctr1.count = ni;
 969|
 970|        /*
 971|         * the rdn attribute (the alias for the name attribute),
 972|         * 'cn' for most objects is the last entry in the meta data array
 973|         * we have stored
 974|         *
 975|         * sort the new meta data array
 976|         */
 977|        {
 978|                struct replPropertyMetaData1 *rdn_p;
 979|                uint32_t rdn_idx = omd.ctr.ctr1.count - 1;
 980|
 981|                rdn_p = &nmd.ctr.ctr1.array[rdn_idx];
 982|                replmd_replPropertyMetaDataCtr1_sort(&nmd.ctr.ctr1, &rdn_p->attid);
 983|        }
 984|
 985|        /* create the meta data value */
 986|        nt_status = ndr_push_struct_blob(&nmd_value, msg, &nmd,
 987|                                         (ndr_push_flags_fn_t)ndr_push_replPropertyMetaDa
->   taBlob);
 988|        if (!NT_STATUS_IS_OK(nt_status)) {
 989|                return replmd_replicated_request_werror(ar,
->    ntstatus_to_werror(nt_status));
 990|        }
 991|
```

```
 992|        /*
 993|         * check if some replicated attributes left, otherwise skip the ldb_modify()
->    call
 994|         */
 995|        if (msg->num_elements == 0) {
 996|                ldb_debug(ar->module->ldb, LDB_DEBUG_TRACE,
->    "replmd_replicated_apply_merge[%u]: skip replace\n",
 997|                          ar->index_current);
 998|                goto next_object;
 999|        }
1000|
1001|        ldb_debug(ar->module->ldb, LDB_DEBUG_TRACE, "replmd_replicated_apply_merge[%u]:
->    replace %u attributes\n",
1002|                  ar->index_current, msg->num_elements);
1003|
1004|        /*
1005|         * when we now that we'll modify the record, add the whenChanged, uSNChanged
1006|         * and replPopertyMetaData attributes
1007|         */
1008|        ret = ldb_msg_add_string(msg, "whenChanged", ar->objs->objects[ar->index_current]
->   .when_changed);
1009|        if (ret != LDB_SUCCESS) {
1010|                return replmd_replicated_request_error(ar, ret);
1011|        }
1012|        ret = samdb_msg_add_uint64(ar->module->ldb, msg, msg, "uSNChanged", seq_num);
1013|        if (ret != LDB_SUCCESS) {
1014|                return replmd_replicated_request_error(ar, ret);
1015|        }
1016|        ret = ldb_msg_add_value(msg, "replPropertyMetaData", &nmd_value, NULL);
1017|        if (ret != LDB_SUCCESS) {
1018|                return replmd_replicated_request_error(ar, ret);
1019|        }
1020|
1021|        replmd_ldb_message_sort(msg, ar->schema);
1022|
1023|        /* we want to replace the old values */
1024|        for (i=0; i < msg->num_elements; i++) {
1025|                msg->elements[i].flags = LDB_FLAG_MOD_REPLACE;
1026|        }
1027|
1028|        ret = ldb_build_mod_req(&ar->sub.change_req,
1029|                                 ar->module->ldb,
1030|                                 ar->sub.mem_ctx,
1031|                                 msg,
1032|                                 NULL,
1033|                                 ar,
1034|                                 replmd_replicated_apply_merge_callback);
1035|        if (ret != LDB_SUCCESS) return replmd_replicated_request_error(ar, ret);
1036|
1037|#ifdef REPLMD_FULL_ASYNC /* TODO: activate this code when ldb support full async code */
->
1038|        return ldb_next_request(ar->module, ar->sub.change_req);
1039|#else
1040|        ret = ldb_next_request(ar->module, ar->sub.change_req);
1041|        if (ret != LDB_SUCCESS) return replmd_replicated_request_error(ar, ret);
1042|
1043|        ar->sub.change_ret = ldb_wait(ar->sub.search_req->handle, LDB_WAIT_ALL);
1044|        if (ar->sub.change_ret != LDB_SUCCESS) {
1045|                return replmd_replicated_request_error(ar, ar->sub.change_ret);
1046|        }
1047|
1048|next_object:
1049|        talloc_free(ar->sub.mem_ctx);
1050|        ZERO_STRUCT(ar->sub);
1051|
1052|        ar->index_current++;
1053|
1054|        return LDB_SUCCESS;
1055|#endif
1056|}
1057|
1058|static int replmd_replicated_apply_search_callback(struct ldb_context *ldb,
1059|                                                   void *private_data,
1060|                                                   struct ldb_reply *ares)
1061|{
1062|        struct replmd_replicated_request *ar = talloc_get_type(private_data,
1063|                                                 struct replmd_replicated_request);
1064|        bool is_done = false;
1065|
1066|        switch (ares->type) {
1067|        case LDB_REPLY_ENTRY:
1068|                ar->sub.search_msg = talloc_steal(ar->sub.mem_ctx, ares->message);
1069|                break;
1070|        case LDB_REPLY_REFERRAL:
1071|                /* we ignore referrals */
1072|                break;
1073|        case LDB_REPLY_EXTENDED:
```

```
1074|        case LDB_REPLY_DONE:
1075|                is_done = true;
1076|        }
1077|
1078|        talloc_free(ares);
1079|
1080|#ifdef REPLMD_FULL_ASYNC /* TODO: activate this code when ldb support full async code */
->
1081|        if (is_done) {
1082|                ar->sub.search_ret = ldb_wait(ar->sub.search_req->handle, LDB_WAIT_ALL);
1083|                if (ar->sub.search_ret != LDB_SUCCESS) {
1084|                        return replmd_replicated_request_error(ar, ar->sub.search_ret);
1085|                }
1086|                if (ar->sub.search_msg) {
1087|                        return replmd_replicated_apply_merge(ar);
1088|                }
1089|                return replmd_replicated_apply_add(ar);
1090|        }
1091|#endif
1092|        return LDB_SUCCESS;
1093|}
1094|
1095|static int replmd_replicated_apply_search(struct replmd_replicated_request *ar)
1096|{
1097|        int ret;
1098|        char *tmp_str;
1099|        char *filter;
1100|
1101|        tmp_str = ldb_binary_encode(ar->sub.mem_ctx, ar->objs->objects[ar->index_current]
->   .guid_value);
1102|        if (!tmp_str) return replmd_replicated_request_werror(ar, WERR_NOMEM);
1103|
1104|        filter = talloc_asprintf(ar->sub.mem_ctx, "(objectGUID=%s)", tmp_str);
1105|        if (!filter) return replmd_replicated_request_werror(ar, WERR_NOMEM);
1106|        talloc_free(tmp_str);
1107|
1108|        ret = ldb_build_search_req(&ar->sub.search_req,
1109|                                    ar->module->ldb,
1110|                                    ar->sub.mem_ctx,
1111|                                    ar->objs->partition_dn,
1112|                                    LDB_SCOPE_SUBTREE,
1113|                                    filter,
1114|                                    NULL,
1115|                                    NULL,
1116|                                    ar,
1117|                                    replmd_replicated_apply_search_callback);
1118|        if (ret != LDB_SUCCESS) return replmd_replicated_request_error(ar, ret);
1119|
1120|#ifdef REPLMD_FULL_ASYNC /* TODO: activate this code when ldb support full async code */
->
1121|        return ldb_next_request(ar->module, ar->sub.search_req);
1122|#else
1123|        ret = ldb_next_request(ar->module, ar->sub.search_req);
1124|        if (ret != LDB_SUCCESS) return replmd_replicated_request_error(ar, ret);
1125|
1126|        ar->sub.search_ret = ldb_wait(ar->sub.search_req->handle, LDB_WAIT_ALL);
1127|        if (ar->sub.search_ret != LDB_SUCCESS) {
1128|                return replmd_replicated_request_error(ar, ar->sub.search_ret);
1129|        }
1130|        if (ar->sub.search_msg) {
1131|                return replmd_replicated_apply_merge(ar);
1132|        }
1133|
1134|        return replmd_replicated_apply_add(ar);
1135|#endif
1136|}
1137|
1138|static int replmd_replicated_apply_next(struct replmd_replicated_request *ar)
1139|{
1140|#ifdef REPLMD_FULL_ASYNC /* TODO: activate this code when ldb support full async code */
->
1141|        if (ar->index_current >= ar->objs->num_objects) {
1142|                return replmd_replicated_uptodate_vector(ar);
1143|        }
1144|#endif
1145|
1146|        ar->sub.mem_ctx = talloc_new(ar);
1147|        if (!ar->sub.mem_ctx) return replmd_replicated_request_werror(ar, WERR_NOMEM);
1148|
1149|        return replmd_replicated_apply_search(ar);
1150|}
1151|
1152|static int replmd_replicated_uptodate_modify_callback(struct ldb_context *ldb,
1153|                                                      void *private_data,
1154|                                                      struct ldb_reply *ares)
1155|{
1156|#ifdef REPLMD_FULL_ASYNC /* TODO: activate this code when ldb support full async code */
```

```
->
1157|        struct replmd_replicated_request *ar = talloc_get_type(private_data,
1158|                                                struct replmd_replicated_request);
1159|
1160|        ar->sub.change_ret = ldb_wait(ar->sub.search_req->handle, LDB_WAIT_ALL);
1161|        if (ar->sub.change_ret != LDB_SUCCESS) {
1162|                return replmd_replicated_request_error(ar, ar->sub.change_ret);
1163|        }
1164|
1165|        talloc_free(ar->sub.mem_ctx);
1166|        ZERO_STRUCT(ar->sub);
1167|
1168|        return replmd_replicated_request_done(ar);
1169|#else
1170|        return LDB_SUCCESS;
1171|#endif
1172|}
1173|
1174|static int replmd_drsuapi_DsReplicaCursor2_compare(const struct drsuapi_DsReplicaCursor2
->    *c1,
1175|                                                   const struct drsuapi_DsReplicaCursor2
->    *c2)
1176|{
1177|        return GUID_compare(&c1->source_dsa_invocation_id,
->    &c2->source_dsa_invocation_id);
1178|}
1179|
1180|static int replmd_replicated_uptodate_modify(struct replmd_replicated_request *ar)
1181|{
1182|        NTSTATUS nt_status;
1183|        struct ldb_message *msg;
1184|        struct replUpToDateVectorBlob ouv;
1185|        const struct ldb_val *ouv_value;
1186|        const struct drsuapi_DsReplicaCursor2CtrEx *ruv;
1187|        struct replUpToDateVectorBlob nuv;
1188|        struct ldb_val nuv_value;
1189|        struct ldb_message_element *nuv_el = NULL;
1190|        const struct GUID *our_invocation_id;
1191|        struct ldb_message_element *orf_el = NULL;
1192|        struct repsFromToBlob nrf;
1193|        struct ldb_val *nrf_value = NULL;
1194|        struct ldb_message_element *nrf_el = NULL;
1195|        uint32_t i,j,ni=0;
1196|        uint64_t seq_num;
1197|        bool found = false;
1198|        time_t t = time(NULL);
1199|        NTTIME now;
1200|        int ret;
1201|
1202|        ruv = ar->objs->uptodateness_vector;
1203|        ZERO_STRUCT(ouv);
1204|        ouv.version = 2;
1205|        ZERO_STRUCT(nuv);
1206|        nuv.version = 2;
1207|
1208|        unix_to_nt_time(&now, t);
1209|
1210|        /*
1211|         * we use the next sequence number for our own highest_usn
1212|         * because we will do a modify request and this will increment
1213|         * our highest_usn
1214|         */
1215|        ret = ldb_sequence_number(ar->module->ldb, LDB_SEQ_NEXT, &seq_num);
1216|        if (ret != LDB_SUCCESS) {
1217|                return replmd_replicated_request_error(ar, ret);
1218|        }
1219|
1220|        /*
1221|         * first create the new replUpToDateVector
1222|         */
1223|        ouv_value = ldb_msg_find_ldb_val(ar->sub.search_msg, "replUpToDateVector");
1224|        if (ouv_value) {
1225|                nt_status = ndr_pull_struct_blob(ouv_value, ar->sub.mem_ctx, &ouv,
1226|                                                 (ndr_pull_flags_fn_t)ndr_pull_replUpToDa
->   teVectorBlob);
1227|                if (!NT_STATUS_IS_OK(nt_status)) {
1228|                        return replmd_replicated_request_werror(ar,
->    ntstatus_to_werror(nt_status));
1229|                }
1230|
1231|                if (ouv.version != 2) {
1232|                        return replmd_replicated_request_werror(ar,
->    WERR_DS_DRA_INTERNAL_ERROR);
1233|                }
1234|        }
1235|
1236|        /*
```

```
1237|         * the new uptodateness vector will at least
1238|         * contain 1 entry, one for the source_dsa
1239|         *
1240|         * plus optional values from our old vector and the one from the source_dsa
1241|         */
1242|        nuv.ctr.ctr2.count = 1 + ouv.ctr.ctr2.count;
1243|        if (ruv) nuv.ctr.ctr2.count += ruv->count;
1244|        nuv.ctr.ctr2.cursors = talloc_array(ar->sub.mem_ctx,
1245|                                            struct drsuapi_DsReplicaCursor2,
1246|                                            nuv.ctr.ctr2.count);
1247|        if (!nuv.ctr.ctr2.cursors) return replmd_replicated_request_werror(ar,
->    WERR_NOMEM);
1248|
1249|        /* first copy the old vector */
1250|        for (i=0; i < ouv.ctr.ctr2.count; i++) {
1251|                nuv.ctr.ctr2.cursors[ni] = ouv.ctr.ctr2.cursors[i];
1252|                ni++;
1253|        }
1254|
1255|        /* get our invocation_id if we have one already attached to the ldb */
1256|        our_invocation_id = samdb_ntds_invocation_id(ar->module->ldb);
1257|
1258|        /* merge in the source_dsa vector is available */
1259|        for (i=0; (ruv && i < ruv->count); i++) {
1260|                found = false;
1261|
1262|                if (our_invocation_id &&
1263|                    GUID_equal(&ruv->cursors[i].source_dsa_invocation_id,
1264|                               our_invocation_id)) {
1265|                        continue;
1266|                }
1267|
1268|                for (j=0; j < ni; j++) {
1269|                        if (!GUID_equal(&ruv->cursors[i].source_dsa_invocation_id,
1270|                                        &nuv.ctr.ctr2.cursors[j].source_dsa_invocation_id
->   )) {
1271|                                continue;
1272|                        }
1273|
1274|                        found = true;
1275|
1276|                        /*
1277|                         * we update only the highest_usn and not the latest_sync_success
->    time,
1278|                         * because the last success stands for direct replication
1279|                         */
1280|                        if (ruv->cursors[i].highest_usn > nuv.ctr.ctr2.cursors[j].highest
->   _usn) {
1281|                                nuv.ctr.ctr2.cursors[j].highest_usn =
->    ruv->cursors[i].highest_usn;
1282|                        }
1283|                        break;
1284|                }
1285|
1286|                if (found) continue;
1287|
1288|                /* if it's not there yet, add it */
1289|                nuv.ctr.ctr2.cursors[ni] = ruv->cursors[i];
1290|                ni++;
1291|        }
1292|
1293|        /*
1294|         * merge in the current highwatermark for the source_dsa
1295|         */
1296|        found = false;
1297|        for (j=0; j < ni; j++) {
1298|                if (!GUID_equal(&ar->objs->source_dsa->source_dsa_invocation_id,
1299|                                &nuv.ctr.ctr2.cursors[j].source_dsa_invocation_id)) {
1300|                        continue;
1301|                }
1302|
1303|                found = true;
1304|
1305|                /*
1306|                 * here we update the highest_usn and last_sync_success time
1307|                 * because we're directly replicating from the source_dsa
1308|                 *
1309|                 * and use the tmp_highest_usn because this is what we have just applied
1310|                 * to our ldb
1311|                 */
1312|                nuv.ctr.ctr2.cursors[j].highest_usn            =
->    ar->objs->source_dsa->highwatermark.tmp_highest_usn;
1313|                nuv.ctr.ctr2.cursors[j].last_sync_success      = now;
1314|                break;
1315|        }
1316|        if (!found) {
1317|                /*
```

```
1318|                 * here we update the highest_usn and last_sync_success time
1319|                 * because we're directly replicating from the source_dsa
1320|                 *
1321|                 * and use the tmp_highest_usn because this is what we have just applied
1322|                 * to our ldb
1323|                 */
1324|                nuv.ctr.ctr2.cursors[ni].source_dsa_invocation_id=
->    ar->objs->source_dsa->source_dsa_invocation_id;
1325|                nuv.ctr.ctr2.cursors[ni].highest_usn            =
->    ar->objs->source_dsa->highwatermark.tmp_highest_usn;
1326|                nuv.ctr.ctr2.cursors[ni].last_sync_success      = now;
1327|                ni++;
1328|        }
1329|
1330|        /*
1331|         * finally correct the size of the cursors array
1332|         */
1333|        nuv.ctr.ctr2.count = ni;
1334|
1335|        /*
1336|         * sort the cursors
1337|         */
1338|        qsort(nuv.ctr.ctr2.cursors, nuv.ctr.ctr2.count,
1339|              sizeof(struct drsuapi_DsReplicaCursor2),
1340|              (comparison_fn_t)replmd_drsuapi_DsReplicaCursor2_compare);
1341|
1342|        /*
1343|         * create the change ldb_message
1344|         */
1345|        msg = ldb_msg_new(ar->sub.mem_ctx);
1346|        if (!msg) return replmd_replicated_request_werror(ar, WERR_NOMEM);
1347|        msg->dn = ar->sub.search_msg->dn;
1348|
1349|        nt_status = ndr_push_struct_blob(&nuv_value, msg, &nuv,
1350|                                          (ndr_push_flags_fn_t)ndr_push_replUpToDateVector
->   Blob);
1351|        if (!NT_STATUS_IS_OK(nt_status)) {
1352|                return replmd_replicated_request_werror(ar,
->    ntstatus_to_werror(nt_status));
1353|        }
1354|        ret = ldb_msg_add_value(msg, "replUpToDateVector", &nuv_value, &nuv_el);
1355|        if (ret != LDB_SUCCESS) {
1356|                return replmd_replicated_request_error(ar, ret);
1357|        }
1358|        nuv_el->flags = LDB_FLAG_MOD_REPLACE;
1359|
1360|        /*
1361|         * now create the new repsFrom value from the given repsFromTo1 structure
1362|         */
1363|        ZERO_STRUCT(nrf);
1364|        nrf.version                                     = 1;
1365|        nrf.ctr.ctr1                                   = *ar->objs->source_dsa;
1366|        /* and fix some values... */
1367|        nrf.ctr.ctr1.consecutive_sync_failures         = 0;
1368|        nrf.ctr.ctr1.last_success                      = now;
1369|        nrf.ctr.ctr1.last_attempt                      = now;
1370|        nrf.ctr.ctr1.result_last_attempt               = WERR_OK;
1371|        nrf.ctr.ctr1.highwatermark.highest_usn         = nrf.ctr.ctr1.highwatermark.tmp_
->   highest_usn;
1372|
1373|        /*
1374|         * first see if we already have a repsFrom value for the current source dsa
1375|         * if so we'll later replace this value
1376|         */
1377|        orf_el = ldb_msg_find_element(ar->sub.search_msg, "repsFrom");
1378|        if (orf_el) {
1379|                for (i=0; i < orf_el->num_values; i++) {
1380|                        struct repsFromToBlob *trf;
1381|
1382|                        trf = talloc(ar->sub.mem_ctx, struct repsFromToBlob);
1383|                        if (!trf) return replmd_replicated_request_werror(ar,
->    WERR_NOMEM);
1384|
1385|                        nt_status = ndr_pull_struct_blob(&orf_el->values[i], trf, trf,
1386|                                                         (ndr_pull_flags_fn_t)ndr_pull_re
->   psFromToBlob);
1387|                        if (!NT_STATUS_IS_OK(nt_status)) {
1388|                                return replmd_replicated_request_werror(ar,
->    ntstatus_to_werror(nt_status));
1389|                        }
1390|
1391|                        if (trf->version != 1) {
1392|                                return replmd_replicated_request_werror(ar,
->    WERR_DS_DRA_INTERNAL_ERROR);
1393|                        }
1394|
1395|                        /*
```

```
1396|                          * we compare the source dsa objectGUID not the invocation_id
1397|                          * because we want only one repsFrom value per source dsa
1398|                          * and when the invocation_id of the source dsa has changed we
->    don't need
1399|                          * the old repsFrom with the old invocation_id
1400|                          */
1401|                         if (!GUID_equal(&trf->ctr.ctr1.source_dsa_obj_guid,
1402|                                         &ar->objs->source_dsa->source_dsa_obj_guid)) {
1403|                                 talloc_free(trf);
1404|                                 continue;
1405|                         }
1406|
1407|                         talloc_free(trf);
1408|                         nrf_value = &orf_el->values[i];
1409|                         break;
1410|                 }
1411|
1412|                 /*
1413|                  * copy over all old values to the new ldb_message
1414|                  */
1415|                 ret = ldb_msg_add_empty(msg, "repsFrom", 0, &nrf_el);
1416|                 if (ret != LDB_SUCCESS) return replmd_replicated_request_error(ar, ret);
1417|                 *nrf_el = *orf_el;
1418|         }
1419|
1420|         /*
1421|          * if we haven't found an old repsFrom value for the current source dsa
1422|          * we'll add a new value
1423|          */
1424|         if (!nrf_value) {
1425|                 struct ldb_val zero_value;
1426|                 ZERO_STRUCT(zero_value);
1427|                 ret = ldb_msg_add_value(msg, "repsFrom", &zero_value, &nrf_el);
1428|                 if (ret != LDB_SUCCESS) return replmd_replicated_request_error(ar, ret);
1429|
1430|                 nrf_value = &nrf_el->values[nrf_el->num_values - 1];
1431|         }
1432|
1433|         /* we now fill the value which is already attached to ldb_message */
1434|         nt_status = ndr_push_struct_blob(nrf_value, msg, &nrf,
1435|                                          (ndr_push_flags_fn_t)ndr_push_repsFromToBlob);
1436|         if (!NT_STATUS_IS_OK(nt_status)) {
1437|                 return replmd_replicated_request_werror(ar,
->    ntstatus_to_werror(nt_status));
1438|         }
1439|
1440|         /*
1441|          * the ldb_message_element for the attribute, has all the old values and the new
->    one
1442|          * so we'll replace the whole attribute with all values
1443|          */
1444|         nrf_el->flags = LDB_FLAG_MOD_REPLACE;
1445|
1446|         /* prepare the ldb_modify() request */
1447|         ret = ldb_build_mod_req(&ar->sub.change_req,
1448|                                 ar->module->ldb,
1449|                                 ar->sub.mem_ctx,
1450|                                 msg,
1451|                                 NULL,
1452|                                 ar,
1453|                                 replmd_replicated_uptodate_modify_callback);
1454|         if (ret != LDB_SUCCESS) return replmd_replicated_request_error(ar, ret);
1455|
1456|#ifdef REPLMD_FULL_ASYNC /* TODO: activate this code when ldb support full async code */
->
1457|         return ldb_next_request(ar->module, ar->sub.change_req);
1458|#else
1459|         ret = ldb_next_request(ar->module, ar->sub.change_req);
1460|         if (ret != LDB_SUCCESS) return replmd_replicated_request_error(ar, ret);
1461|
1462|         ar->sub.change_ret = ldb_wait(ar->sub.search_req->handle, LDB_WAIT_ALL);
1463|         if (ar->sub.change_ret != LDB_SUCCESS) {
1464|                 return replmd_replicated_request_error(ar, ar->sub.change_ret);
1465|         }
1466|
1467|         talloc_free(ar->sub.mem_ctx);
1468|         ZERO_STRUCT(ar->sub);
1469|
1470|         return replmd_replicated_request_done(ar);
1471|#endif
1472|}
1473|
1474|static int replmd_replicated_uptodate_search_callback(struct ldb_context *ldb,
1475|                                                       void *private_data,
1476|                                                       struct ldb_reply *ares)
1477|{
1478|         struct replmd_replicated_request *ar = talloc_get_type(private_data,
```

```
1479|                                                 struct replmd_replicated_request);
1480|        bool is_done = false;
1481|
1482|        switch (ares->type) {
1483|        case LDB_REPLY_ENTRY:
1484|                ar->sub.search_msg = talloc_steal(ar->sub.mem_ctx, ares->message);
1485|                break;
1486|        case LDB_REPLY_REFERRAL:
1487|                /* we ignore referrals */
1488|                break;
1489|        case LDB_REPLY_EXTENDED:
1490|        case LDB_REPLY_DONE:
1491|                is_done = true;
1492|        }
1493|
1494|        talloc_free(ares);
1495|
1496|#ifdef REPLMD_FULL_ASYNC /* TODO: activate this code when ldb support full async code */
->
1497|        if (is_done) {
1498|                ar->sub.search_ret = ldb_wait(ar->sub.search_req->handle, LDB_WAIT_ALL);
1499|                if (ar->sub.search_ret != LDB_SUCCESS) {
1500|                        return replmd_replicated_request_error(ar, ar->sub.search_ret);
1501|                }
1502|                if (!ar->sub.search_msg) {
1503|                        return replmd_replicated_request_werror(ar,
->    WERR_DS_DRA_INTERNAL_ERROR);
1504|                }
1505|
1506|                return replmd_replicated_uptodate_modify(ar);
1507|        }
1508|#endif
1509|        return LDB_SUCCESS;
1510|}
1511|
1512|static int replmd_replicated_uptodate_search(struct replmd_replicated_request *ar)
1513|{
1514|        int ret;
1515|        static const char *attrs[] = {
1516|                "replUpToDateVector",
1517|                "repsFrom",
1518|                NULL
1519|        };
1520|
1521|        ret = ldb_build_search_req(&ar->sub.search_req,
1522|                                   ar->module->ldb,
1523|                                   ar->sub.mem_ctx,
1524|                                   ar->objs->partition_dn,
1525|                                   LDB_SCOPE_BASE,
1526|                                   "(objectClass=*)",
1527|                                   attrs,
1528|                                   NULL,
1529|                                   ar,
1530|                                   replmd_replicated_uptodate_search_callback);
1531|        if (ret != LDB_SUCCESS) return replmd_replicated_request_error(ar, ret);
1532|
1533|#ifdef REPLMD_FULL_ASYNC /* TODO: activate this code when ldb support full async code */
->
1534|        return ldb_next_request(ar->module, ar->sub.search_req);
1535|#else
1536|        ret = ldb_next_request(ar->module, ar->sub.search_req);
1537|        if (ret != LDB_SUCCESS) return replmd_replicated_request_error(ar, ret);
1538|
1539|        ar->sub.search_ret = ldb_wait(ar->sub.search_req->handle, LDB_WAIT_ALL);
1540|        if (ar->sub.search_ret != LDB_SUCCESS) {
1541|                return replmd_replicated_request_error(ar, ar->sub.search_ret);
1542|        }
1543|        if (!ar->sub.search_msg) {
1544|                return replmd_replicated_request_werror(ar, WERR_DS_DRA_INTERNAL_ERROR);
1545|        }
1546|
1547|        return replmd_replicated_uptodate_modify(ar);
1548|#endif
1549|}
1550|
1551|static int replmd_replicated_uptodate_vector(struct replmd_replicated_request *ar)
1552|{
1553|        ar->sub.mem_ctx = talloc_new(ar);
1554|        if (!ar->sub.mem_ctx) return replmd_replicated_request_werror(ar, WERR_NOMEM);
1555|
1556|        return replmd_replicated_uptodate_search(ar);
1557|}
1558|
1559|static int replmd_extended_replicated_objects(struct ldb_module *module, struct
->    ldb_request *req)
1560|{
1561|        struct dsdb_extended_replicated_objects *objs;
```

```
1562|        struct replmd_replicated_request *ar;
1563|
1564|        ldb_debug(module->ldb, LDB_DEBUG_TRACE, "replmd_extended_replicated_objects\n");
1565|
1566|        objs = talloc_get_type(req->op.extended.data, struct
->    dsdb_extended_replicated_objects);
1567|        if (!objs) {
1568|                ldb_debug(module->ldb, LDB_DEBUG_FATAL, "replmd_extended_replicated_objec
->   ts: invalid extended data\n");
1569|                return LDB_ERR_PROTOCOL_ERROR;
1570|        }
1571|
1572|        if (objs->version != DSDB_EXTENDED_REPLICATED_OBJECTS_VERSION) {
1573|                ldb_debug(module->ldb, LDB_DEBUG_FATAL, "replmd_extended_replicated_objec
->   ts: extended data invalid version [%u != %u]\n",
1574|                          objs->version, DSDB_EXTENDED_REPLICATED_OBJECTS_VERSION);
1575|                return LDB_ERR_PROTOCOL_ERROR;
1576|        }
1577|
1578|        ar = replmd_replicated_init_handle(module, req, objs);
1579|        if (!ar) {
1580|                return LDB_ERR_OPERATIONS_ERROR;
1581|        }
1582|
1583|#ifdef REPLMD_FULL_ASYNC /* TODO: activate this code when ldb support full async code */
->
1584|        return replmd_replicated_apply_next(ar);
1585|#else
1586|        while (ar->index_current < ar->objs->num_objects &&
1587|               req->handle->state != LDB_ASYNC_DONE) {
1588|                replmd_replicated_apply_next(ar);
1589|        }
1590|
1591|        if (req->handle->state != LDB_ASYNC_DONE) {
1592|                replmd_replicated_uptodate_vector(ar);
1593|        }
1594|
1595|        return LDB_SUCCESS;
1596|#endif
1597|}
1598|
1599|static int replmd_extended(struct ldb_module *module, struct ldb_request *req)
1600|{
1601|        if (strcmp(req->op.extended.oid, DSDB_EXTENDED_REPLICATED_OBJECTS_OID) == 0) {
1602|                return replmd_extended_replicated_objects(module, req);
1603|        }
1604|
1605|        return ldb_next_request(module, req);
1606|}
1607|
1608|static int replmd_wait_none(struct ldb_handle *handle) {
1609|        struct replmd_replicated_request *ar;
1610|
1611|        if (!handle || !handle->private_data) {
1612|                return LDB_ERR_OPERATIONS_ERROR;
1613|        }
1614|
1615|        ar = talloc_get_type(handle->private_data, struct replmd_replicated_request);
1616|        if (!ar) {
1617|                return LDB_ERR_OPERATIONS_ERROR;
1618|        }
1619|
1620|        /* we do only sync calls */
1621|        if (handle->state != LDB_ASYNC_DONE) {
1622|                return LDB_ERR_OPERATIONS_ERROR;
1623|        }
1624|
1625|        return handle->status;
1626|}
1627|
1628|static int replmd_wait_all(struct ldb_handle *handle) {
1629|
1630|        int ret;
1631|
1632|        while (handle->state != LDB_ASYNC_DONE) {
1633|                ret = replmd_wait_none(handle);
1634|                if (ret != LDB_SUCCESS) {
1635|                        return ret;
1636|                }
1637|        }
1638|
1639|        return handle->status;
1640|}
1641|
1642|static int replmd_wait(struct ldb_handle *handle, enum ldb_wait_type type)
1643|{
1644|        if (type == LDB_WAIT_ALL) {
```

```
1645|                return replmd_wait_all(handle);
1646|        } else {
1647|                return replmd_wait_none(handle);
1648|        }
1649|}
1650|
1651|static const struct ldb_module_ops replmd_ops = {
1652|        .name          = "repl_meta_data",
1653|        .add           = replmd_add,
1654|        .modify        = replmd_modify,
1655|        .extended      = replmd_extended,
1656|        .wait          = replmd_wait
1657|};
1658|
1659|int repl_meta_data_module_init(void)
1660|{
1661|        return ldb_register_module(&replmd_ops);
1662|}
```

A14. source/dsdb/samdb/ldb_modules/schema_fsmo.c

source/dsdb/samdb/ldb_modules/schema_fsmo.c:

```
   1|/*
   2|   Unix SMB/CIFS mplementation.
   3|
   4|   The module that handles the Schema FSMO Role Owner
   5|   checkings, it also loads the dsdb_schema.
   6|
   7|   Copyright (C) Stefan Metzmacher <metze@samba.org> 2007
   8|
   9|   This program is free software; you can redistribute it and/or modify
  10|   it under the terms of the GNU General Public License as published by
  11|   the Free Software Foundation; either version 2 of the License, or
  12|   (at your option) any later version.
  13|
  14|   This program is distributed in the hope that it will be useful,
  15|   but WITHOUT ANY WARRANTY; without even the implied warranty of
  16|   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  17|   GNU General Public License for more details.
  18|
  19|   You should have received a copy of the GNU General Public License
  20|   along with this program; if not, write to the Free Software
  21|   Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
  22|
  23|*/
  24|
  25|#include "includes.h"
  26|#include "lib/ldb/include/ldb.h"
  27|#include "lib/ldb/include/ldb_errors.h"
  28|#include "lib/ldb/include/ldb_private.h"
  29|#include "dsdb/samdb/samdb.h"
  30|#include "librpc/gen_ndr/ndr_misc.h"
  31|#include "librpc/gen_ndr/ndr_drsuapi.h"
  32|#include "librpc/gen_ndr/ndr_drsblobs.h"
  33|#include "lib/util/dlinklist.h"
  34|
  35|static int schema_fsmo_init(struct ldb_module *module)
  36|{
  37|        WERROR status;
  38|        TALLOC_CTX *mem_ctx;
  39|        struct ldb_dn *schema_dn;
  40|        struct dsdb_schema *schema;
  41|        struct dsdb_schema_fsmo *schema_fsmo;
  42|        struct ldb_result *schema_res;
  43|        const struct ldb_val *prefix_val;
  44|        const struct ldb_val *info_val;
  45|        struct ldb_val info_val_default;
  46|        struct ldb_result *a_res;
  47|        struct ldb_result *c_res;
  48|        uint32_t i;
  49|        int ret;
  50|        static const char *schema_attrs[] = {
  51|                "prefixMap",
  52|                "schemaInfo",
  53|                "fSMORoleOwner",
  54|                NULL
  55|        };
  56|
  57|        schema_dn = samdb_schema_dn(module->ldb);
  58|        if (!schema_dn) {
  59|                ldb_debug(module->ldb, LDB_DEBUG_WARNING,
  60|                          "schema_fsmo_init: no schema dn present: (skip schema
->    loading)\n");
  61|                return ldb_next_init(module);
  62|        }
  63|
  64|        mem_ctx = talloc_new(module);
  65|        if (!mem_ctx) {
  66|                ldb_oom(module->ldb);
  67|                return LDB_ERR_OPERATIONS_ERROR;
  68|        }
  69|
  70|        schema_fsmo = talloc_zero(mem_ctx, struct dsdb_schema_fsmo);
  71|        if (!schema_fsmo) {
  72|                ldb_oom(module->ldb);
  73|                return LDB_ERR_OPERATIONS_ERROR;
  74|        }
  75|        module->private_data = schema_fsmo;
  76|
  77|        schema = talloc_zero(mem_ctx, struct dsdb_schema);
  78|        if (!schema) {
```

```
  79|                ldb_oom(module->ldb);
  80|                return LDB_ERR_OPERATIONS_ERROR;
  81|        }
  82|
  83|        /*
  84|         * setup the prefix mappings and schema info
  85|         */
  86|        ret = ldb_search(module->ldb, schema_dn,
  87|                         LDB_SCOPE_BASE,
  88|                         NULL, schema_attrs,
  89|                         &schema_res);
  90|        if (ret != LDB_SUCCESS) {
  91|                ldb_debug_set(module->ldb, LDB_DEBUG_FATAL,
  92|                              "schema_fsmo_init: failed to search the schema head:
->    %d:%s\n",
  93|                              ret, ldb_strerror(ret));
  94|                talloc_free(mem_ctx);
  95|                return ret;
  96|        }
  97|        talloc_steal(mem_ctx, schema_res);
  98|        if (schema_res->count == 0) {
  99|                ldb_debug(module->ldb, LDB_DEBUG_WARNING,
 100|                          "schema_fsmo_init: no schema head present: (skip schema
->    loading)\n");
 101|                talloc_free(mem_ctx);
 102|                return ldb_next_init(module);
 103|        } else if (schema_res->count > 1) {
 104|                ldb_debug_set(module->ldb, LDB_DEBUG_FATAL,
 105|                              "schema_fsmo_init: [%u] schema heads found on a base
->    search\n",
 106|                              schema_res->count);
 107|                talloc_free(mem_ctx);
 108|                return LDB_ERR_CONSTRAINT_VIOLATION;
 109|        }
 110|
 111|        prefix_val = ldb_msg_find_ldb_val(schema_res->msgs[0], "prefixMap");
 112|        if (!prefix_val) {
 113|                ldb_debug_set(module->ldb, LDB_DEBUG_FATAL,
 114|                              "schema_fsmo_init: no prefixMap attribute found\n");
 115|                talloc_free(mem_ctx);
 116|                return LDB_ERR_CONSTRAINT_VIOLATION;
 117|        }
 118|        info_val = ldb_msg_find_ldb_val(schema_res->msgs[0], "schemaInfo");
 119|        if (!info_val) {
 120|                info_val_default = strhex_to_data_blob("FF0000000000000000000000000000000000
->   000000000");
 121|                if (!info_val_default.data) {
 122|                        ldb_oom(module->ldb);
 123|                        return LDB_ERR_OPERATIONS_ERROR;
 124|                }
 125|                talloc_steal(mem_ctx, info_val_default.data);
 126|                info_val = &info_val_default;
 127|        }
 128|
 129|        status = dsdb_load_oid_mappings_ldb(schema, prefix_val, info_val);
 130|        if (!W_ERROR_IS_OK(status)) {
 131|                ldb_debug_set(module->ldb, LDB_DEBUG_FATAL,
 132|                              "schema_fsmo_init: failed to load oid mappings: %s\n",
 133|                              win_errstr(status));
 134|                talloc_free(mem_ctx);
 135|                return LDB_ERR_CONSTRAINT_VIOLATION;
 136|        }
 137|
 138|        /*
 139|         * load the attribute definitions
 140|         */
 141|        ret = ldb_search(module->ldb, schema_dn,
 142|                         LDB_SCOPE_ONELEVEL,
 143|                         "(objectClass=attributeSchema)", NULL,
 144|                         &a_res);
 145|        if (ret != LDB_SUCCESS) {
 146|                ldb_debug_set(module->ldb, LDB_DEBUG_FATAL,
 147|                              "schema_fsmo_init: failed to search attributeSchema
->    objects: %d:%s\n",
 148|                              ret, ldb_strerror(ret));
 149|                talloc_free(mem_ctx);
 150|                return ret;
 151|        }
 152|        talloc_steal(mem_ctx, a_res);
 153|
 154|        for (i=0; i < a_res->count; i++) {
 155|                struct dsdb_attribute *sa;
 156|
 157|                sa = talloc_zero(schema, struct dsdb_attribute);
 158|                if (!sa) {
 159|                        ldb_oom(module->ldb);
 160|                        return LDB_ERR_OPERATIONS_ERROR;
```

```
 161|                }
 162|
 163|                status = dsdb_attribute_from_ldb(schema, a_res->msgs[i], sa, sa);
 164|                if (!W_ERROR_IS_OK(status)) {
 165|                        ldb_debug_set(module->ldb, LDB_DEBUG_FATAL,
 166|                                      "schema_fsmo_init: failed to load attriute
->    definition: %s:%s\n",
 167|                                      ldb_dn_get_linearized(a_res->msgs[i]->dn),
 168|                                      win_errstr(status));
 169|                        talloc_free(mem_ctx);
 170|                        return LDB_ERR_CONSTRAINT_VIOLATION;
 171|                }
 172|
 173|                DLIST_ADD_END(schema->attributes, sa, struct dsdb_attribute *);
 174|        }
 175|        talloc_free(a_res);
 176|
 177|        /*
 178|         * load the objectClass definitions
 179|         */
 180|        ret = ldb_search(module->ldb, schema_dn,
 181|                          LDB_SCOPE_ONELEVEL,
 182|                          "(objectClass=classSchema)", NULL,
 183|                          &c_res);
 184|        if (ret != LDB_SUCCESS) {
 185|                ldb_debug_set(module->ldb, LDB_DEBUG_FATAL,
 186|                              "schema_fsmo_init: failed to search classSchema objects:
->    %d:%s\n",
 187|                              ret, ldb_strerror(ret));
 188|                talloc_free(mem_ctx);
 189|                return ret;
 190|        }
 191|        talloc_steal(mem_ctx, c_res);
 192|
 193|        for (i=0; i < c_res->count; i++) {
 194|                struct dsdb_class *sc;
 195|
 196|                sc = talloc_zero(schema, struct dsdb_class);
 197|                if (!sc) {
 198|                        ldb_oom(module->ldb);
 199|                        return LDB_ERR_OPERATIONS_ERROR;
 200|                }
 201|
 202|                status = dsdb_class_from_ldb(schema, c_res->msgs[i], sc, sc);
 203|                if (!W_ERROR_IS_OK(status)) {
 204|                        ldb_debug_set(module->ldb, LDB_DEBUG_FATAL,
 205|                                      "schema_fsmo_init: failed to load class definition:
->    %s:%s\n",
 206|                                      ldb_dn_get_linearized(c_res->msgs[i]->dn),
 207|                                      win_errstr(status));
 208|                        talloc_free(mem_ctx);
 209|                        return LDB_ERR_CONSTRAINT_VIOLATION;
 210|                }
 211|
 212|                DLIST_ADD_END(schema->classes, sc, struct dsdb_class *);
 213|        }
 214|        talloc_free(c_res);
 215|
 216|        /* dsdb_set_schema() steal schema into the ldb_context */
 217|        ret = dsdb_set_schema(module->ldb, schema);
 218|        if (ret != LDB_SUCCESS) {
 219|                ldb_debug_set(module->ldb, LDB_DEBUG_FATAL,
 220|                              "schema_fsmo_init: dsdb_set_schema() failed: %d:%s\n",
 221|                              ret, ldb_strerror(ret));
 222|                talloc_free(mem_ctx);
 223|                return ret;
 224|        }
 225|
 226|        schema_fsmo->master_dn = ldb_msg_find_attr_as_dn(module->ldb, schema_fsmo,
->    schema_res->msgs[0], "fSMORoleOwner");
 227|        if (ldb_dn_compare(samdb_ntds_settings_dn(module->ldb), schema_fsmo->master_dn)
->    == 0) {
 228|                schema_fsmo->we_are_master = true;
 229|        } else {
 230|                schema_fsmo->we_are_master = false;
 231|        }
 232|
 233|        if (ldb_set_opaque(module->ldb, "dsdb_schema_fsmo", schema_fsmo) != LDB_SUCCESS)
->    {
 234|                ldb_oom(module->ldb);
 235|                return LDB_ERR_OPERATIONS_ERROR;
 236|        }
 237|
 238|        talloc_steal(module, schema_fsmo);
 239|
 240|        ldb_debug(module->ldb, LDB_DEBUG_TRACE,
 241|                          "schema_fsmo_init: we are master: %s\n",
```

```
242|                          (schema_fsmo->we_are_master?"yes":"no"));
243|
244|        talloc_free(mem_ctx);
245|        return ldb_next_init(module);
246|}
247|
248|static const struct ldb_module_ops schema_fsmo_ops = {
249|        .name           = "schema_fsmo",
250|        .init_context   = schema_fsmo_init
251|};
252|
253|int schema_fsmo_module_init(void)
254|{
255|        return ldb_register_module(&schema_fsmo_ops);
256|}
```

A15. source/dsdb/samdb/ldb_modules/show_deleted.c

source/dsdb/samdb/ldb_modules/show_deleted.c:

```
   1|/*
   2|   ldb database library
   3|
   4|   Copyright (C) Simo Sorce  2005
   5|   Copyright (C) Stefa Metzmacher <metze@samba.org> 2007
   6|
   7|     ** NOTE! The following LGPL license applies to the ldb
   8|     ** library. This does NOT imply that all of Samba is released
   9|     ** under the LGPL
  10|
  11|   This library is free software; you can redistribute it and/or
  12|   modify it under the terms of the GNU Lesser General Public
  13|   License as published by the Free Software Foundation; either
  14|   version 2 of the License, or (at your option) any later version.
  15|
  16|   This library is distributed in the hope that it will be useful,
  17|   but WITHOUT ANY WARRANTY; without even the implied warranty of
  18|   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
  19|   Lesser General Public License for more details.
  20|
  21|   You should have received a copy of the GNU Lesser General Public
  22|   License along with this library; if not, write to the Free Software
  23|   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
  24|*/
  25|
  26|/*
  27| *  Name: ldb
  28| *
  29| *  Component: ldb deleted objects control module
  30| *
  31| *  Description: this module hides deleted objects, and returns them if the control is
->    there
  32| *
  33| *  Author: Stefan Metzmacher
  34| */
  35|
  36|#include "includes.h"
  37|#include "ldb/include/ldb.h"
  38|#include "ldb/include/ldb_errors.h"
  39|#include "ldb/include/ldb_private.h"
  40|#include "dsdb/samdb/samdb.h"
  41|
  42|/* search */
  43|struct show_deleted_search_request {
  44|
  45|        struct ldb_module *module;
  46|        void *up_context;
  47|        int (*up_callback)(struct ldb_context *, void *, struct ldb_reply *);
  48|
  49|        bool remove_from_msg;
  50|};
  51|
  52|static int show_deleted_search_callback(struct ldb_context *ldb, void *context, struct
->    ldb_reply *ares)
  53|{
  54|        struct show_deleted_search_request *ar;
  55|
  56|        if (!context || !ares) {
  57|                ldb_set_errstring(ldb, "NULL Context or Result in callback");
  58|                goto error;
  59|        }
  60|
  61|        ar = talloc_get_type(context, struct show_deleted_search_request);
  62|
  63|        if (ares->type == LDB_REPLY_ENTRY) {
  64|                bool isDeleted;
  65|
  66|                isDeleted = ldb_msg_find_attr_as_bool(ares->message, "isDeleted",
->    false);
  67|
  68|                if (isDeleted) {
  69|                        goto skip_deleted;
  70|                }
  71|
  72|                if (ar->remove_from_msg) {
  73|                        ldb_msg_remove_attr(ares->message, "isDeleted");
  74|                }
  75|        }
  76|
```

```
  77|        return ar->up_callback(ldb, ar->up_context, ares);
  78|
  79|skip_deleted:
  80|        talloc_free(ares);
  81|        return LDB_SUCCESS;
  82|error:
  83|        talloc_free(ares);
  84|        return LDB_ERR_OPERATIONS_ERROR;
  85|}
  86|
  87|static int show_deleted_search(struct ldb_module *module, struct ldb_request *req)
  88|{
  89|        struct ldb_control *control;
  90|        struct ldb_control **saved_controls;
  91|        struct show_deleted_search_request *ar;
  92|        struct ldb_request *down_req;
  93|        char **new_attrs;
  94|        uint32_t num_attrs = 0;
  95|        uint32_t i;
  96|        int ret;
  97|
  98|        /* check if there's a show deleted control */
  99|        control = ldb_request_get_control(req, LDB_CONTROL_SHOW_DELETED_OID);
 100|
 101|        /* copy the request for modification */
 102|        down_req = talloc(req, struct ldb_request);
 103|        if (down_req == NULL) {
 104|                ldb_oom(module->ldb);
 105|                return LDB_ERR_OPERATIONS_ERROR;
 106|        }
 107|
 108|        /* copy the request */
 109|        *down_req = *req;
 110|
 111|        /* if a control is there remove if from the modified request */
 112|        if (control && !save_controls(control, down_req, &saved_controls)) {
 113|                return LDB_ERR_OPERATIONS_ERROR;
 114|        }
 115|
 116|        /* if we had a control, then just go on to the next request as we have nothing to
->    hide */
 117|        if (control) {
 118|                goto next_request;
 119|        }
 120|
 121|        ar = talloc(down_req, struct show_deleted_search_request);
 122|        if (ar == NULL) {
 123|                ldb_oom(module->ldb);
 124|                return LDB_ERR_OPERATIONS_ERROR;
 125|        }
 126|
 127|        ar->module              = module;
 128|        ar->up_context          = req->context;
 129|        ar->up_callback         = req->callback;
 130|        ar->remove_from_msg     = true;
 131|
 132|        /* check if attrs only is specified, in that case check wether we need to modify
->    them */
 133|        if (down_req->op.search.attrs) {
 134|                for (i=0; (down_req->op.search.attrs && down_req->op.search.attrs[i]);
->    i++) {
 135|                        num_attrs++;
 136|                        if (strcasecmp(down_req->op.search.attrs[i], "*") == 0) {
 137|                                ar->remove_from_msg = false;
 138|                        } else if (strcasecmp(down_req->op.search.attrs[i], "isDeleted")
->    == 0) {
 139|                                ar->remove_from_msg = false;
 140|                        }
 141|                }
 142|        } else {
 143|                ar->remove_from_msg = false;
 144|        }
 145|
 146|        if (ar->remove_from_msg) {
 147|                new_attrs = talloc_array(down_req, char *, num_attrs + 2);
 148|                if (!new_attrs) {
 149|                        ldb_oom(module->ldb);
 150|                        return LDB_ERR_OPERATIONS_ERROR;
 151|                }
 152|                for (i=0; i < num_attrs; i++) {
 153|                        new_attrs[i] = discard_const_p(char,
->    down_req->op.search.attrs[i]);
 154|                }
 155|                new_attrs[i] = talloc_strdup(new_attrs, "isDeleted");
 156|                if (!new_attrs[i]) {
 157|                        ldb_oom(module->ldb);
 158|                        return LDB_ERR_OPERATIONS_ERROR;
```

```
159|                }
160|                new_attrs[i+1] = NULL;
161|                down_req->op.search.attrs = (const char * const *)new_attrs;
162|        }
163|
164|        down_req->context = ar;
165|        down_req->callback = show_deleted_search_callback;
166|        ldb_set_timeout_from_prev_req(module->ldb, req, down_req);
167|
168|next_request:
169|        /* perform the search */
170|        ret = ldb_next_request(module, down_req);
171|
172|        /* do not free down_req as the call results may be linked to it,
173|         * it will be freed when the upper level request get freed */
174|        if (ret == LDB_SUCCESS) {
175|                req->handle = down_req->handle;
176|        }
177|
178|        return ret;
179|}
180|
181|static int show_deleted_init(struct ldb_module *module)
182|{
183|        struct ldb_request *req;
184|        int ret;
185|
186|        req = talloc(module, struct ldb_request);
187|        if (req == NULL) {
188|                return LDB_ERR_OPERATIONS_ERROR;
189|        }
190|
191|        req->operation = LDB_REQ_REGISTER_CONTROL;
192|        req->op.reg_control.oid = LDB_CONTROL_SHOW_DELETED_OID;
193|        req->controls = NULL;
194|
195|        ret = ldb_request(module->ldb, req);
196|        if (ret != LDB_SUCCESS) {
197|                ldb_debug(module->ldb, LDB_DEBUG_ERROR, "show_deleted: Unable to register
->    control with rootdse!\n");
198|                talloc_free(req);
199|                return LDB_ERR_OPERATIONS_ERROR;
200|        }
201|
202|        talloc_free(req);
203|        return ldb_next_init(module);
204|}
205|
206|static const struct ldb_module_ops show_deleted_ops = {
207|        .name              = "show_deleted",
208|        .search            = show_deleted_search,
209|        .init_context      = show_deleted_init
210|};
211|
212|int ldb_show_deleted_init(void)
213|{
214|        return ldb_register_module(&show_deleted_ops);
215|}
```

A16. Inhalt der beigelegten CD

Der gedruckten Form dieser Bachelorarbeit liegt eine CD mit folgendem Inhalt bei:

- Eine PDF-Version dieser Bachelorarbeit.
- Der Quelltext von Samba 4.0 in Revision 21859.

Wo die Daten auf der CD zu finden sind, ist in der Datei `INDEX.txt` beschrieben, die sich im Hauptverzeichnis befindet.

A17. Anleitung zu Samba 4.0

```
Anleitung zu Samba 4.0 :

   1|Die Folgenden Schritte zeigen wie Samba 4.0 auf einem
   2|modernen Linux-System kompiliert, installiert und
   3|getestet wird.
   4|
   5|
   6|Auf dem Linux-System müssen alle Tools zur
   7|C-, Python- und Perl-Entwicklung installiert sein.
   8|Außerdem wird das 'subversion' Paket benötigt, um
   9|die, zum Abgabetermin dieser Bachelorarbeit, aktuelle
  10|Revision 21859 auszuchecken.
  11|
  12|Auf der beigelegten CD (siehe Anhang A16) ist diese Version auch gespeichert.
  13|(Hier kann der './autogen.sh' Schritt entfallen)
  14|
  15|!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
  16|Achtung: Generell sollte nur in einer reinen Test-Umgebung
  17|         getestet werden!
  18|!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
  19|
  20|thesis@tempo4:~> svn co -r 21859 svn://svnanon.samba.org/samba/branches/SAMBA_4_0/
->    SAMBA_4_0_rev_21859
  21|thesis@tempo4:~> cd SAMBA_4_0_rev_21859/source/
  22|thesis@tempo4:~/SAMBA_4_0_rev_21859/source>./autogen.sh
  23|thesis@tempo4:~/SAMBA_4_0_rev_21859/source>./configure.developer --prefix=$HOME/samba4
  24|thesis@tempo4:~/SAMBA_4_0_rev_21859/source> make && make install
  25|thesis@tempo4:~/SAMBA_4_0_rev_21859/source> cd ~/samba4/
  26|thesis@tempo4:~/samba4> /sbin/ifconfig vmnet24
  27|vmnet24   Protokoll:Ethernet  Hardware Adresse 00:50:56:C0:00:18
  28|          inet Adresse:172.31.24.1  Bcast:172.31.24.255  Maske:255.255.255.0
  29|          inet6 Adresse: fe80::250:56ff:fec0:18/64 Gültigkeitsbereich:Verbindung
  30|          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
  31|          RX packets:127758 errors:0 dropped:0 overruns:0 frame:0
  32|          TX packets:32501 errors:0 dropped:0 overruns:0 carrier:0
  33|          collisions:0 Sendewarteschlangenlänge:1000
  34|          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
  35|
  36|
  37|Die folgenden Konfigurationen müssen angepasst werden!
  38|Achtung: Die krb5.conf und hosts sind sehr wichtig!
  39|
  40|
  41|thesis@tempo4:~/samba4> cat ~/samba4/etc/smb.conf
  42|[globals]
  43|        netbios name            = smbtorturedc
  44|        workgroup               = W2K3-THESIS
  45|        realm                   = W2K3-THESIS.VMNET24.VM.BASE
  46|        sam database            = smbtorturedc_samdb.ldb
  47|        secrets database        = smbtorturedc_secrets.ldb
  48|
  49|        server role             = domain controller
  50|
  51|        bind interfaces only    = yes
  52|        interfaces              = vmnet24
  53|
  54|[c$]
  55|        path                    = /home/thesis/prefix/samba4/test
  56|        read only               = no
  57|        ntvfs handler           = posix
  58|
  59|thesis@tempo4:~/samba4> cat /etc/krb5.conf
  60|[libdefaults]
  61|        default_realm = W2K3-THESIS.VMNET24.VM.BASE
  62|[realms]
  63|        W2K3-THESIS = {
  64|                kdc = w2k3-21.w2k3-thesis.vmnet24.vm.base
  65|        }
  66|        W2K3-THESIS.VMNET24.VM.BASE = {
  67|                kdc = w2k3-21.w2k3-thesis.vmnet24.vm.base
  68|        }
  69|[domain_realms]
  70|        . = W2K3-THESIS.VMNET24.VM.BASE
  71|        .w2k3-thesis.vmnet24.vm.base = W2K3-THESIS.VMNET24.VM.BASE
  72|thesis@tempo4:~/samba4> cat /etc/hosts
  73|127.0.0.1       localhost
  74|172.31.24.1     tempo4
  75|172.31.24.21    w2k3-21.w2k3-thesis.vmnet24.vm.base
  76|
  77|
  78|
```

```
  79|
  80|
  81|
  82|
  83|Replizieren der Daten
  84|Achtung: vor jedem Lauf sollten die alten Dateien gelöscht werden!
  85|
  86|!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
  87|Achtung: Keines Falls in einer Produktiv-Umgebung testen!!!
  88|!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
  89|
  90|thesis@tempo4:~/samba4> rm private/smbtorturedc_*
  91|thesis@tempo4:~/samba4> bin/smbtorture ncacn_np:w2k3-21.w2k3-thesis.vmnet24.vm.base -U
->    administrator%test -W W2K3-THESIS --realm w2k3-thesis.vmnet24.vm.base NET-API-BECOME-DC
  92|Using seed 1174111058
  93|Become DC [smbtorturedc] of Domain[W2K3-THESIS]/[w2k3-thesis.vmnet24.vm.base]
  94|Promotion Partner is Server[w2k3-21.w2k3-thesis.vmnet24.vm.base] from
->    Site[Default-First-Site-Name]
  95|Options:crossRef behavior_version[0]
  96|        schema object_version[30]
  97|        domain behavior_version[0]
  98|        domain w2k3_update_revision[8]
  99|New Server[smbtorturedc.w2k3-thesis.vmnet24.vm.base] in Site[Default-First-Site-Name]
 100|DSA Instance [CN=NTDS Settings,CN=smbtorturedc,CN=Servers,CN=Default-First-Site-Name,CN=S
->   ites,CN=Configuration,DC=w2k3-thesis,DC=vmnet24,DC=vm,DC=base]
 101|        objectGUID[f1bb4021-522f-4267-adcd-7326a86009b9]
 102|        invocationId[26c94114-cfc2-429b-b8f7-c45ae89938f0]
 103|Pathes under PRIVATEDIR[/home/thesis/samba4/private]
 104|SAMDB[smbtorturedc_samdb.ldb] SECRETS[smbtorturedc_secrets.ldb]
->    KEYTAB[smbtorturedc_secrets.keytab]
 105|Schema Partition[CN=Schema,CN=Configuration,DC=w2k3-thesis,DC=vmnet24,DC=vm,DC=base =>
->    smbtorturedc_schema.ldb]
 106|Config Partition[CN=Configuration,DC=w2k3-thesis,DC=vmnet24,DC=vm,DC=base =>
->    smbtorturedc_config.ldb]
 107|Domain Partition[DC=w2k3-thesis,DC=vmnet24,DC=vm,DC=base => smbtorturedc_domain.ldb]
 108|Setting up smbtorturedc_samdb.ldb partitions
 109|schema_fsmo_init: no schema dn present: (skip schema loading)
 110|naming_fsmo_init: no partitions dn present: (skip loading of naming contexts details)
 111|pdc_fsmo_init: no domain dn present: (skip loading of domain details)
 112|Setting up smbtorturedc_samdb.ldb attributes
 113|Setting up smbtorturedc_samdb.ldb rootDSE
 114|Erasing data from partitions
 115|Setting up smbtorturedc_samdb.ldb indexes
 116|Setting up smbtorturedc_samdb.ldb templates
 117|Setting up smbtorturedc_secrets.ldb
 118|WARNING: probable memory leak in ldb /home/thesis/samba4/private/smbtorturedc_samdb.ldb -
->    837 blocks (startup 247) 14819 bytes
 119|Open the SAM LDB with system credentials: smbtorturedc_samdb.ldb
 120|schema_fsmo_init: no schema head present: (skip schema loading)
 121|naming_fsmo_init: no cross-ref container present: (skip loading of naming contexts
->    details)
 122|pdc_fsmo_init: no domain object present: (skip loading of domain details)
 123|Schema-DN[CN=Schema,CN=Configuration,DC=w2k3-thesis,DC=vmnet24,DC=vm,DC=base]
->    objects[133]
 124|Schema-DN[CN=Schema,CN=Configuration,DC=w2k3-thesis,DC=vmnet24,DC=vm,DC=base]
->    objects[133]
 125|Schema-DN[CN=Schema,CN=Configuration,DC=w2k3-thesis,DC=vmnet24,DC=vm,DC=base]
->    objects[133]
 126|Schema-DN[CN=Schema,CN=Configuration,DC=w2k3-thesis,DC=vmnet24,DC=vm,DC=base]
->    objects[133]
 127|Schema-DN[CN=Schema,CN=Configuration,DC=w2k3-thesis,DC=vmnet24,DC=vm,DC=base]
->    objects[133]
 128|Schema-DN[CN=Schema,CN=Configuration,DC=w2k3-thesis,DC=vmnet24,DC=vm,DC=base]
->    objects[133]
 129|Schema-DN[CN=Schema,CN=Configuration,DC=w2k3-thesis,DC=vmnet24,DC=vm,DC=base]
->    objects[133]
 130|Schema-DN[CN=Schema,CN=Configuration,DC=w2k3-thesis,DC=vmnet24,DC=vm,DC=base]
->    objects[133]
 131|Schema-DN[CN=Schema,CN=Configuration,DC=w2k3-thesis,DC=vmnet24,DC=vm,DC=base]
->    objects[133]
 132|Schema-DN[CN=Schema,CN=Configuration,DC=w2k3-thesis,DC=vmnet24,DC=vm,DC=base]
->    objects[75]
 133|Analyze and apply schema objects
 134|WARNING: probable memory leak in ldb /home/thesis/samba4/private/smbtorturedc_samdb.ldb -
->    2243 blocks (startup 908) 200331 bytes
 135|Reopen the SAM LDB with system credentials and a already stored schema:
->    smbtorturedc_samdb.ldb
 136|naming_fsmo_init: no cross-ref container present: (skip loading of naming contexts
->    details)
 137|pdc_fsmo_init: no domain object present: (skip loading of domain details)
 138|Partition[CN=Configuration,DC=w2k3-thesis,DC=vmnet24,DC=vm,DC=base] objects[133]
 139|Partition[CN=Configuration,DC=w2k3-thesis,DC=vmnet24,DC=vm,DC=base] objects[133]
 140|Partition[CN=Configuration,DC=w2k3-thesis,DC=vmnet24,DC=vm,DC=base] objects[133]
 141|Partition[CN=Configuration,DC=w2k3-thesis,DC=vmnet24,DC=vm,DC=base] objects[133]
 142|Partition[CN=Configuration,DC=w2k3-thesis,DC=vmnet24,DC=vm,DC=base] objects[133]
 143|Partition[CN=Configuration,DC=w2k3-thesis,DC=vmnet24,DC=vm,DC=base] objects[133]
 144|Partition[CN=Configuration,DC=w2k3-thesis,DC=vmnet24,DC=vm,DC=base] objects[133]
```

```
Partition[CN=Configuration,DC=w2k3-thesis,DC=vmnet24,DC=vm,DC=base] objects[133]
Partition[CN=Configuration,DC=w2k3-thesis,DC=vmnet24,DC=vm,DC=base] objects[133]
Partition[CN=Configuration,DC=w2k3-thesis,DC=vmnet24,DC=vm,DC=base] objects[133]
Partition[CN=Configuration,DC=w2k3-thesis,DC=vmnet24,DC=vm,DC=base] objects[133]
Partition[CN=Configuration,DC=w2k3-thesis,DC=vmnet24,DC=vm,DC=base] objects[84]
Partition[DC=w2k3-thesis,DC=vmnet24,DC=vm,DC=base] objects[133]
Partition[DC=w2k3-thesis,DC=vmnet24,DC=vm,DC=base] objects[82]
mark ROOTDSE with isSynchronized=TRUE
Reopen the SAM LDB with system credentials and all replicated data:
->    smbtorturedc_samdb.ldb
Delete of machine account was successful.


thesis@tempo4:~/samba4> ls -l ~/samba4/private/
insgesamt 10736
-rw------- 1 thesis thesis 5070848 2007-03-17 02:14 smbtorturedc_config.ldb
-rw------- 1 thesis thesis 1204224 2007-03-17 02:14 smbtorturedc_domain.ldb
-rw------- 1 thesis thesis  147456 2007-03-17 02:14 smbtorturedc_samdb.ldb
-rw------- 1 thesis thesis 4435968 2007-03-17 02:14 smbtorturedc_schema.ldb
-rw------- 1 thesis thesis     674 2007-03-17 02:14 smbtorturedc_secrets.keytab
-rw------- 1 thesis thesis  131072 2007-03-17 02:14 smbtorturedc_secrets.ldb
drwxr-sr-x 2 thesis thesis       6 2007-03-15 15:27 tls


Die Daten lokal überprüfen:


thesis@tempo4:~/samba4> bin/ldbsearch -H ~/samba4/private/smbtorturedc_samdb.ldb
->    "(name=Administrator)"
# record 1
dn: CN=Administrator,CN=Users,DC=w2k3-thesis,DC=vmnet24,DC=vm,DC=base
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: user
cn: Administrator
description: Built-in account for administering the computer/domain
instanceType: 4
whenCreated: 20061217150534.0Z
whenChanged: 20061217153251.0Z
uSNCreated: 9065
uSNChanged: 9065
nTSecurityDescriptor: O:S-1-5-21-2283420010-3047391439-292721624-512G:S-1-5-21
 -2283420010-3047391439-292721624-512D:PAI(A;;RPLCLORC;;;AU)(A;;RPWPCRCCDCLCLO
 RCWOWDSDSW;;;BA)(A;;RPWPCRCCDCLCLORCWOWDSW;;;S-1-5-21-2283420010-3047391439-2
 92721624-519)(A;;RPWPCRCCDCLCLORCWOWDSW;;;S-1-5-21-2283420010-3047391439-2927
 21624-512)(A;;RPWPCRCCDCLCLORCWOWDSDDTSW;;;SY)(OA;;RP;037088f8-0ae1-11d2-b422
 -00a0c968f939;bf967aba-0de6-11d0-a285-00aa003049e2;RU)(OA;;RP;59ba2f42-79a2-1
 1d0-9020-00c04fc2d3cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)(OA;;RP;bc0ac24
 0-79a9-11d0-9020-00c04fc2d4cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)(OA;;RP
 ;4c164200-20c0-11d0-a768-00aa006e0529;bf967aba-0de6-11d0-a285-00aa003049e2;RU
 )(OA;;RP;5f202010-79a5-11d0-9020-00c04fc2d4cf;bf967aba-0de6-11d0-a285-00aa003
 049e2;RU)(OA;;RPLCLORC;;bf967aba-0de6-11d0-a285-00aa003049e2;RU)(OA;;CR;ab721
 a53-1e2f-11d0-9819-00aa0040529b;;WD)(OA;;CR;ab721a53-1e2f-11d0-9819-00aa00405
 29b;;PS)(OA;;RPWP;bf967a7f-0de6-11d0-a285-00aa003049e2;;S-1-5-21-2283420010-3
 047391439-292721624-517)(OA;;RP;037088f8-0ae1-11d2-b422-00a0c968f939;4828cc14
 -1437-45bc-9b07-ad6f015e5f28;RU)(OA;;RP;59ba2f42-79a2-11d0-9020-00c04fc2d3cf;
 4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)(OA;;RP;bc0ac240-79a9-11d0-9020-00c04
 fc2d4cf;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)(OA;;RP;4c164200-20c0-11d0-a7
 68-00aa006e0529;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)(OA;;RP;5f202010-79a5
 -11d0-9020-00c04fc2d4cf;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)(OA;;RPLCLORC
 ;;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)(OA;;RP;46a9b11d-60ae-405a-b7e8-ff8
 a58d456d2;;S-1-5-32-560)(OA;;RPWP;6db69a1c-9422-11d1-aebd-0000f80367c1;;S-1-5
 -32-561)S:AI(AU;SA;WPWOWD;;;WD)(OU;CIIOIDSA;WP;f30e3bbe-9ff0-11d1-b603-0000f8
 0367c1;bf967aa5-0de6-11d0-a285-00aa003049e2;WD)(OU;CIIOIDSA;WP;f30e3bbf-9ff0-
 11d1-b603-0000f80367c1;bf967aa5-0de6-11d0-a285-00aa003049e2;WD)
name: Administrator
objectGUID: 299cb4b3-cd36-419d-8f15-1b030bc02b80
replPropertyMetaData:: AQAAAAAAAAAgAAAAAAAAAAAAAABAAAAvu2V+wIAAAAuK0NCfVjRR6E
 ga9/Rb0HWAiAAAAAAAABpIwAAAAAAAA0AAAABAAAAvu2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAA
 AAAAAABpIwAAAAAAAAEAAgABAAAAvu2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAAAAAAAABpIwAAA
 AAAAAIAAgABAAAAvu2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAAAAAAAABpIwAAAAAAAA0AAgABAA
 AAvu2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAAAAAAAABpIwAAAAAAABkBAgACAAAAI/SV+wIAAAA
 uK0NCfVjRR6Ega9/Rb0HW4TAAAAAAAABpIwAAAAAAAAEACQABAAAAvu2V+wIAAAAuK0NCfVjRR6Eg
 a9/Rb0HWAiAAAAAAAABpIwAAAAAAAAgACQABAAAAvu2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAAA
 AAAAABpIwAAAAAAABAACQABAAAAvu2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAAAAAAAABpIwAAAA
 AAABkACQABAAAAvu2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAAAAAAAABpIwAAAAAAACwACQABAAA
 Avu2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAAAAAAAABpIwAAAAAAAC0ACQABAAAAvu2V+wIAAAAu
 K0NCfVjRR6Ega9/Rb0HWAiAAAAAAAABpIwAAAAAAADcACQABAAAAvu2V+wIAAAAuK0NCfVjRR6Ega
 9/Rb0HWAiAAAAAAAABpIwAAAAAAAD4ACQABAAAAvu2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAAAA
 AAAABpIwAAAAAAAEAACQABAAAAvu2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAAAAAAAABpIwAAAAA
 AAFYACQABAAAAvu2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAAAAAAAABpIwAAAAAAAFoACQABAAAA
 vu2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAAAAAAAABpIwAAAAAAAF4ACQABAAAAvu2V+wIAAAAuK
 0NCfVjRR6Ega9/Rb0HWAiAAAAAAAABpIwAAAAAAAGAACQABAAAAvu2V+wIAAAAuK0NCfVjRR6Ega9
 /Rb0HWAiAAAAAAAABpIwAAAAAAAGIACQABAAAAvu2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAAAAA
 AAABpIwAAAAAAAIoACQABAAAAvu2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAAAAAAAABpIwAAAAAA
 AIsACQABAAAAvu2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAAAAAAAABpIwAAAAAAAJIACQABAAAAv
```

```
 230| u2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAAAAAAAABpIwAAAAAAAJYACQABAAAAI/SV+wIAAAAuK0
 231| NCfVjRR6Ega9/Rb0HW4TAAAAAAAABpIwAAAAAAAJwACQABAAAAvu2V+wIAAAAuK0NCfVjRR6Ega9/
 232| Rb0HWAiAAAAAAAABpIwAAAAAAAJ8ACQABAAAAvu2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAAAAAA
 233| AABpIwAAAAAAAKAACQABAAAAvu2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAAAAAAAABpIwAAAAAA
 234| N0ACQABAAAAvu2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAAAAAAAABpIwAAAAAAC4BCQABAAAAvu
 235| 2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAAAAAAAABpIwAAAAAAA4DCQABAAAAvu2V+wIAAAAuK0N
 236| CfVjRR6Ega9/Rb0HWAiAAAAAAAABpIwAAAAAAAGQDCQABAAAAvu2V+wIAAAAuK0NCfVjRR6Ega9/R
 237| b0HWAiAAAAAAAABpIwAAAAAAAAMAAAABAAAAvu2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAAAAAA
 238| ABpIwAAAAAAAA==
 239|userAccountControl: 66048
 240|codePage: 0
 241|countryCode: 0
 242|dBCSPwd:: Afxaa+e8aSmq07Q1tRQE7g==
 243|unicodePwd:: DLaUiAX3l78qgoB5c7iVNw==
 244|pwdLastSet: 128085090543125000
 245|primaryGroupID: 513
 246|objectSid: S-1-5-21-2283420010-3047391439-292721624-500
 247|adminCount: 1
 248|accountExpires: 9223372036854775807
 249|sAMAccountName: Administrator
 250|sAMAccountType: 805306368
 251|objectCategory: CN=Person,CN=Schema,CN=Configuration,DC=w2k3-thesis,DC=vmnet24
 252| ,DC=vm,DC=base
 253|isCriticalSystemObject: TRUE
 254|distinguishedName: CN=Administrator,CN=Users,DC=w2k3-thesis,DC=vmnet24,DC=vm,D
 255| C=base
 256|
 257|# returned 1 records
 258|# 1 entries
 259|# 0 referrals
 260|
 261|
 262|
 263|
 264|Den Server starten (muss mit 'kill' später gestoppt werden):
 265|
 266|thesis@tempo4:~/samba4> su -c "sbin/smbd -M single"
 267|Passwort:
 268|thesis@tempo4:~/samba4> ps ax |grep smbd
 269|18085 ?        Ss     0:00 SCREEN -S smbd
 270|26873 ?        Rs     0:05 sbin/smbd -M single
 271|26875 pts/0    S+     0:00 grep smbd
 272|
 273|
 274|
 275|Die Daten und die Authentifizierung über LDAP testen:
 276|
 277|
 278|
 279|thesis@tempo4:~/samba4> bin/ldbsearch -H ldap://172.31.24.1/ -U administrator%test
->    "(name=Administrator)"
 280|# record 1
 281|dn: CN=Administrator,CN=Users,DC=w2k3-thesis,DC=vmnet24,DC=vm,DC=base
 282|objectClass: top
 283|objectClass: person
 284|objectClass: organizationalPerson
 285|objectClass: user
 286|cn: Administrator
 287|description: Built-in account for administering the computer/domain
 288|instanceType: 4
 289|whenCreated: 20061217150534.0Z
 290|whenChanged: 20061217153251.0Z
 291|uSNCreated: 9065
 292|uSNChanged: 9065
 293|nTSecurityDescriptor: O:S-1-5-21-2283420010-3047391439-292721624-512G:S-1-5-21
 294| -2283420010-3047391439-292721624-512D:PAI(A;;RPLCLORC;;;AU)(A;;RPWPCRCCDCLCLO
 295| RCWOWDSDSW;;;BA)(A;;RPWPCRCCDCLCLORCWOWDSW;;;S-1-5-21-2283420010-3047391439-2
 296| 92721624-519)(A;;RPWPCRCCDCLCLORCWOWDSW;;;S-1-5-21-2283420010-3047391439-2927
 297| 21624-512)(A;;RPWPCRCCDCLCLORCWOWDSDDTSW;;;SY)(OA;;RP;037088f8-0ae1-11d2-b422
 298| -00a0c968f939;bf967aba-0de6-11d0-a285-00aa003049e2;RU)(OA;;RP;59ba2f42-79a2-1
 299| 1d0-9020-00c04fc2d3cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)(OA;;RP;bc0ac24
 300| 0-79a9-11d0-9020-00c04fc2d4cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)(OA;;RP
 301| ;4c164200-20c0-11d0-a768-00aa006e0529;bf967aba-0de6-11d0-a285-00aa003049e2;RU
 302| )(OA;;RP;5f202010-79a5-11d0-9020-00c04fc2d4cf;bf967aba-0de6-11d0-a285-00aa003
 303| 049e2;RU)(OA;;RPLCLORC;;bf967aba-0de6-11d0-a285-00aa003049e2;RU)(OA;;CR;ab721
 304| a53-1e2f-11d0-9819-00aa0040529b;;WD)(OA;;CR;ab721a53-1e2f-11d0-9819-00aa00405
 305| 29b;;PS)(OA;;RPWP;bf967a7f-0de6-11d0-a285-00aa003049e2;;S-1-5-21-2283420010-3
 306| 047391439-292721624-517)(OA;;RP;037088f8-0ae1-11d2-b422-00a0c968f939;4828cc14
 307| -1437-45bc-9b07-ad6f015e5f28;RU)(OA;;RP;59ba2f42-79a2-11d0-9020-00c04fc2d3cf;
 308| 4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)(OA;;RP;bc0ac240-79a9-11d0-9020-00c04
 309| fc2d4cf;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)(OA;;RP;4c164200-20c0-11d0-a7
 310| 68-00aa006e0529;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)(OA;;RP;5f202010-79a5
 311| -11d0-9020-00c04fc2d4cf;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)(OA;;RPLCLORC
 312| ;;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)(OA;;RP;46a9b11d-60ae-405a-b7e8-ff8
 313| a58d456d2;;S-1-5-32-560)(OA;;RPWP;6db69a1c-9422-11d1-aebd-0000f80367c1;;S-1-5
 314| -32-561)S:AI(AU;SA;WPWOWD;;;WD)(OU;CIIOIDSA;WP;f30e3bbe-9ff0-11d1-b603-0000f8
 315| 0367c1;bf967aa5-0de6-11d0-a285-00aa003049e2;WD)(OU;CIIOIDSA;WP;f30e3bbf-9ff0-
```

```
 11d1-b603-0000f80367c1;bf967aa5-0de6-11d0-a285-00aa003049e2;WD)
name: Administrator
objectGUID: 299cb4b3-cd36-419d-8f15-1b030bc02b80
replPropertyMetaData:: AQAAAAAAAAAgAAAAAAAAAAAAAABAAAAvu2V+wIAAAAuK0NCfVjRR6E
 ga9/Rb0HWAiAAAAAAAABpIwAAAAAAAA0AAABAAAAvu2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAA
 AAAAABpIwAAAAAAAAEAAgABAAAAvu2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAAAAAAAABpIwAAA
 AAAAAIAAgABAAAAvu2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAAAAAAAABpIwAAAAAAA0AAgABAA
 AAvu2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAAAAAAAABpIwAAAAAAABkBAgACAAAAI/SV+wIAAAA
 uK0NCfVjRR6Ega9/Rb0HW4TAAAAAAAABpIwAAAAAAAAEACQABAAAAvu2V+wIAAAAuK0NCfVjRR6Eg
 a9/Rb0HWAiAAAAAAAABpIwAAAAAAAAgACQABAAAAvu2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAAA
 AAAAABpIwAAAAAAABAACQABAAAAvu2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAAAAAAAABpIwAAAA
 AAABkACQABAAAAvu2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAAAAAAAABpIwAAAAAAACwACQABAAA
 Avu2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAAAAAAAABpIwAAAAAAC0ACQABAAAAvu2V+wIAAAAu
 K0NCfVjRR6Ega9/Rb0HWAiAAAAAAAABpIwAAAAAADcACQABAAAAvu2V+wIAAAAuK0NCfVjRR6Ega
 9/Rb0HWAiAAAAAAAABpIwAAAAAAD4ACQABAAAAvu2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAAAA
 AAAABpIwAAAAAAAEAACQABAAAAvu2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAAAAAAAABpIwAAAA
 AAFYACQABAAAAvu2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAAAAAAAABpIwAAAAAAFoACQABAAAA
 vu2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAAAAAAAABpIwAAAAAAF4ACQABAAAAvu2V+wIAAAAuK
 0NCfVjRR6Ega9/Rb0HWAiAAAAAAAABpIwAAAAAAGAACQABAAAAvu2V+wIAAAAuK0NCfVjRR6Ega9
 /Rb0HWAiAAAAAAAABpIwAAAAAAGIACQABAAAAvu2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAAAAA
 AAABpIwAAAAAAIoACQABAAAAvu2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAAAAAAAABpIwAAAAAA
 AIsACQABAAAAvu2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAAAAAAAABpIwAAAAAAJIACQABAAAAv
 u2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAAAAAAAABpIwAAAAAAJYACQABAAAAI/SV+wIAAAAuK0
 NCfVjRR6Ega9/Rb0HW4TAAAAAAAABpIwAAAAAAJwACQABAAAAvu2V+wIAAAAuK0NCfVjRR6Ega9/
 Rb0HWAiAAAAAAAABpIwAAAAAAJ8ACQABAAAAvu2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAAAAAA
 AABpIwAAAAAAKAACQABAAAAvu2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAAAAAAAABpIwAAAAAA
 N0ACQABAAAAvu2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAAAAAAAABpIwAAAAAAC4BCQABAAAAvu
 2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAAAAAAAABpIwAAAAAA4DCQABAAAAvu2V+wIAAAAuK0N
 CfVjRR6Ega9/Rb0HWAiAAAAAAAABpIwAAAAAAGQDCQABAAAAvu2V+wIAAAAuK0NCfVjRR6Ega9/R
 b0HWAiAAAAAAAABpIwAAAAAAAMAAAABAAAAvu2V+wIAAAAuK0NCfVjRR6Ega9/Rb0HWAiAAAAAA
 ABpIwAAAAAAAA==
userAccountControl: 66048
codePage: 0
countryCode: 0
dBCSPwd:: Afxaa+e8aSmq07Q1tRQE7g==
unicodePwd:: DLaUiAX3l78qgoB5c7iVNw==
pwdLastSet: 128085090543125000
primaryGroupID: 513
objectSid: S-1-5-21-2283420010-3047391439-292721624-500
adminCount: 1
accountExpires: 9223372036854775807
sAMAccountName: Administrator
sAMAccountType: 805306368
objectCategory: CN=Person,CN=Schema,CN=Configuration,DC=w2k3-thesis,DC=vmnet24
 ,DC=vm,DC=base
isCriticalSystemObject: TRUE
distinguishedName: CN=Administrator,CN=Users,DC=w2k3-thesis,DC=vmnet24,DC=vm,D
 C=base

# returned 1 records
# 1 entries
# 0 referrals

```