

Fachhochschule Köln

University of Applied Sciences Cologne

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

Institut für Nachrichtentechnik
Labor für Informatik

Bachelorarbeit

Thema: Active Directory Replikation

Untertitel: Dokumentation und Analyse des DRSUAPI Replikationsprotokolls,
sowie einer Teilimplementierung der Replikation von Windows-
2003 nach Samba 4.0.

Student: Stefan Metzmacher

Matrikelnummer: 11027678

Referent: Prof. Dr. phil. Gregor Büchel

Korreferent: Prof. Dr.-Ing. Andreas Grebe

Abgabedatum: 18.03.2007

Hiermit versichere ich, dass ich die Bachelorarbeit selbständig angefertigt und keine anderen als die angegebenen und bei Zitaten kenntlich gemachten Quellen und Hilfsmittel benutzt habe.

Stefan Metzmacher

Inhaltsverzeichnis

1 Einführung	1
1.1 Problemstellung	1
1.1.1 Was ist Active-Directory?	1
1.1.2 Was ist Samba?	2
1.1.3 Warum ist das Thema wichtig?	2
1.1.4 Samba soll ein Domänen-Controller für Active-Directory sein	3
1.1.5 Was sind die Schwierigkeiten	3
1.1.6 Warum Teilimplementierung	3
1.2 Vorarbeiten	4
1.2.1 Studienarbeit „Active Directory Replikation“	4
1.2.2 Entwicklung im Rahmen des Samba-Projekts	4
2 Zielsetzung	5
2.1 Implementierung der Funktion libnet_BecomeDC()	5
2.2 Implementierung einer Schema-API	5
2.3 Implementierung eines LDB-Modules zum Speichern der Replikations-Metadaten	6
2.4 Voraussetzungen und Abgrenzungen	6
3 Beschreibung des Replikationssystems	7
3.1 Was ist ein Verzeichnisdienst	7
3.1.1 Übersicht	7
3.1.2 RootDSE	8
3.1.3 Verzeichnis-Objekte	10
3.1.4 Die Partitionen	11
3.2 Grundlegende Protokolle	12
3.2.1 Authentifizierungsprotokolle	12
3.2.2 LDAP	13
3.2.3 CLDAP	14
3.2.4 SMB	14
3.2.5 DCERPC	14
3.2.6 DCERPC basierte Schnittstellen/Protokolle	14
3.2.7 SAMR	14
3.2.8 NETLOGON	15
3.2.9 DRSUAPI	15
3.3 Grundlegende Konzepte des Replikationssystems	16
3.3.1 Source-DSA vs. Destination-DSA	16
3.3.2 Singlemaster-Replikation vs. Multimaster-Replikation	16
3.3.3 Push-Replikation vs. Pull-Replikation	16
3.3.4 Log-basierte vs. Status-basierte Replikation	17
3.3.5 Store-and-Forward-Replikation	17
3.4 Grundlegende Attribute und Strukturen	18
3.4.1 Das „objectGUID“ Attribut	18
3.4.2 Das „objectSid“ Attribut	18
3.4.3 Die Struktur „DsReplicaObjectIdentifier“	18
3.4.4 Das Objekt „NTDS Settings“	19
3.4.5 Das Attribut „highestCommittedUSN“	20
3.4.6 Die Strukturen „DsReplicaCursor“ und „DsReplicaCursorCtr“	21
3.4.7 Die Struktur „DsReplicaHighWaterMark“	21
3.4.8 Die Attribute „repsFrom“ und „repsTo“	22

3.4.9 ASN.1 „Object Identifier“	23
3.4.10 Das „Prefix-Mapping“	23
3.4.11 Die Strukturen „DsReplicaOIDMapping“ und „DsReplicaOIDMapping_Ctr“ ..	25
3.4.12 Originating-Updates vs. Replicating-Updates	25
3.4.13 Das Attribut „replPropertyMetaData“	25
3.4.14 Die Struktur „DsReplicaObject“	26
3.4.15 Die Listen-Elemente „DsReplicaObjectListItem“ und „DsReplicaObjectListItemEx“	27
3.4.16 Die Attribute „uSNCreate“ und „uSNChanged“	28
3.4.17 Das Attribut „instanceType“	28
3.4.18 Das Attribut „systemFlags“	28
3.4.19 Das Attribut „msDs-Behavior-Version“	29
3.5 Originating-Updates	30
3.5.1 Metadaten­speicherung beim Originating-Add	30
3.5.2 Metadaten­speicherung beim Originating-Modify	30
3.5.3 Metadaten­speicherung beim Originating-Move	31
3.5.4 Metadaten­speicherung beim Originating-Delete	31
3.6 Replikation von Verzeichnisdaten	32
3.6.1 Die Funktion DsGetNCChanges ()	32
3.6.2 Interpretation der replizierten Verzeichnis-Objekte	34
3.6.3 Methoden zum Filtern der zu replizierenden Daten	36
3.6.4 Algorithmus zur Konfliktauflösung bei Multimaster-Replikation	37
3.6.5 Change-Notify via DsReplicaSync()	38
3.6.6 Periodische Pull-Replikation	38
4 Implementierung von Teilaspekten basierend auf Samba 4.0	39
4.1 Implementierungskonzepte in Samba 4.0	39
4.1.1 Programmiersprachen	39
4.1.2 Betriebssysteme	39
4.1.3 TALLOC der „Tree-Allocator“	40
4.1.4 Das Subsystem „EVENTS“	40
4.1.5 Asynchrone Funktionsaufrufe	42
4.1.6 TDB, die „Triviale Datenbank“	44
4.1.7 LDB, die „Leichtgewichtige Datenbank“	44
4.2 Das Subsystem „DSDB“ (oder „SAMDB“)	45
4.2.1 DSA-Operational-Layer	45
4.2.2 Database-Layer	45
4.2.3 Storage-Layer	45
4.2.4 DSDB LDB-Module	46
4.3 Änderung der Server-Rolle in die Domänen-Controller-Rolle	50
4.3.1 Die Vorbedingungen für libnet_BecomeDC ()	50
4.3.2 Die Eingabe-Parameter von libnet_BecomeDC ()	50
4.3.3 Die logischen Schritte von libnet_BecomeDC ()	50
4.4 Zugriff auf Schema-Informationen	52
4.4.1 Schema-Cache	52
4.4.2 Die Schema-API-Funktionen zum Erstellen des Schema-Caches	53
4.4.3 Die Schema-API-Funktionen zum Benutzen des Schema-Caches	54
4.4.4 LDB-Modul zum Laden des Schema-Caches	55
4.5 Speicherung der replizierten Daten inklusive von Metadaten	56
4.5.1 Erste Phase: Interpretation und Aufbereitung	56
4.5.2 Zweite Phase: Konfliktauflösung und Speicherung	56
4.5.3 Originating-Add	57
4.6 Der „NET-API-BECOME-DC“ Test	58

4.6.1 Erstellen eines Test-Computer-Kontos	58
4.6.2 Heraufstufen zum Domänen-Controller	58
4.6.3 Herunterstufen und Löschen des Computer-Kontos	59
4.6.4 Ausführen von „NET-API-BECOME-DC“ via smbtorure	59
5 Fazit	60
5.1 Was erreicht wurde	60
5.2 Was noch zu tun ist	60
6 Literaturverzeichnis	61
7 Abkürzungsverzeichnis	65
8 Stichwortverzeichnis	67
9 Anhang	71
A1 drsblobs.idl	71
A2 drsuapi.idl	76
A3 samdb.h	94
A4 schema.h	96
A5 libnet_become_dc.h	98
A6 libnet_unbecome_dc.h	100
A7 torture_libnet_BecomeDC.c	101
A8 libnet_become_dc.c	112
A9 libnet_unbecome_dc.c	148
A10 schema_init.c	157
A11 schema_syntax.c	170
A12 replicated_objects.c	187
A13 repl_meta_data.c	192
A14 schema_fsmo.c	213
A15 show_deleted.c	217
A16 Inhalt der beigelegten CD	220
A17 Anleitung zu Samba 4.0	221

Abbildungsverzeichnis

3.1 Ein typischer Verzeichnis-Baum mit Verzeichnis-Objekten.	8
3.2 Ein typisches RootDSE-Objekt (ein paar Attribut-Werte wurden zur Verbesserung der Übersicht entfernt).	9
3.3 Ein typischer Domänen-Wald unter Windows-2003.	10
3.4 Ein typisches Verzeichnis-Objekt mit Attributen.	11
3.5 Übersicht der Netzwerk-Protokolle (vgl. „Replication and LDAP Client-Server Architecture“ in [MSTN01]).	13
3.6 Die IDL-Beschreibung der <i>Verzeichnis-Objekt-Identifizier</i> Grundstrukturen.	19
3.7 Ein typisches „NTDS Settings“ Objekt.	20
3.8 Die IDL-Beschreibung der <i>Up-To-Date-Vector</i> Grundstrukturen.	21
3.9 Die IDL-Beschreibung der <i>High-Water-Mark</i> Struktur.	21
3.10 Die IDL-Beschreibung der „repsFromToBlob“ Struktur.	22
3.11 Die Bit-Flags „DsReplicaNeighbourFlags“.	23
3.12 Beispiel für das <i>Prefix-Mapping</i> inklusive der in Active-Directory verwendeten Basis-Tabelle.	24
3.13 Die IDL-Beschreibung der <i>Prefix-Mapping</i> Grundstrukturen.	25
3.14 Die IDL-Beschreibung der „replPropertyMetaData“ und „DsReplicaMetaDataCtr“ Grundstrukturen.	26
3.15 Die IDL-Beschreibung der „DsReplicaObject“ Grundstrukturen.	27
3.16 Die IDL-Beschreibung der „DsReplicaObjectListItem“ und „DsReplicaObjectListItemEx“ Listenelemente.	27
3.17 Die Bit-Flags für das „instanceType“ Attribut.	28
3.18 Die Bit-Flags für das „systemFlags“ Attribut.	28
3.19 Die möglichen Werte für das „msDs-Behavior-Version“ Attribut.	29
3.20 Ein Beispiel für ein <i>Tombstone-Objekt</i>	31
3.21 Die IDL-Beschreibung der „DsGetNCChangesRequest8“ Struktur.	32
3.22 Die IDL-Beschreibung der „DsGetNCChangesCtr6“ Struktur.	33
3.23 Die sitzungsspezifische Verschlüsselung der Passwort-Informationen.	36
3.24 Die IDL-Beschreibung der „DsReplicaSyncRequest1“ Struktur.	38
4.1 Ein beispielhafte Nutzung von „TALLOC“.	40
4.2 Ein beispielhafte Nutzung der Funktionen des Subsystem „EVENTS“.	41
4.3 Ein Beispiel einer asynchron aufgerufenen Funktion.	43
4.4 Schichtenvergleich zwischen Windows-2003 und Samba 4.0 (vgl. „Replication Subsystem Components“ in [MSTN01]).	46
4.5 Übersicht der LDB-Module, die im Subsystem „DSDB“ verwendet werden.	47

1 Einführung

In der heutigen IT-Welt spielen Verzeichnisdienste eine zentrale Rolle. Sie bieten einen zentralen Speicherplatz für Benutzerkonten¹ inkl. Gruppenzugehörigkeiten sowie Softwarekonfigurationen jeglicher Art. Außerdem bieten Verzeichnisdienste die Authentifizierung der Benutzer sowie kontrolliert konfigurierbare Richtlinien zum Zugriff auf die im Verzeichnis gespeicherten Objekte und deren Attribute an.

Aus Sicht der IT-Verantwortlichen in Betrieben ist es wünschenswert, frei zwischen alternativen Produkten auf dem Markt der Verzeichnisdienste wählen zu können. Der wichtigste Aspekt ist hierbei, dass alle im Verzeichnis abgelegten Informationen auch wieder exportiert werden können, um auf ein anderes Produkt zu wechseln. Nach dem Motto: „Meine Daten gehören mir und nicht der Software, die sie verwaltet!“.

Aus Sicht der Produkthersteller ist es wünschenswert, den Kunden an das jeweils eigene Produkt zu binden. Bei allen Produkten sind nahezu alle Daten über das *Lightweight Directory Access Protocol* (LDAP) [RFC4510], ein standardisiertes Netzwerkprotokoll zum Zugriff auf Verzeichnisdienste, zugänglich. Doch ein Zugriff auf Benutzerpasswörter ist bei den meisten proprietären Produkten, selbst über verschlüsselte Datenverbindungen mit Administratorprivilegien, nicht möglich. Da aber in den meisten Umgebungen aus Redundanzgründen mehrere Server Kopien der Verzeichnisdaten vorhalten, werden in der Regel nicht standardisierte Netzwerkprotokolle, zum Abgleich der Datenbestände inklusive der Benutzerpasswörter, verwendet. Dieser Umstand sorgt dafür, dass ein Kunde an ein bestimmtes Produkt gebunden ist, da es in den meisten Fällen praktisch unmöglich ist, jedem Benutzer ein neues Passwort zu vergeben.

Der Marktführer Microsoft benutzt in seinem Verzeichnisdienst „Active Directory“ auch ein proprietäres Replikationsprotokoll namens *Directory Replication Service Update API* (DRSUAPI), welches in dieser Bachelorarbeit analysiert wurde.

1.1 Problemstellung

1.1.1 Was ist Active-Directory?

Der Verzeichnisdienst der Firma Microsoft trägt den Namen „Active-Directory“ und ist Bestandteil der Produkte „Windows-2000-Server“ und „Windows-2003-Server“. Active-Directory übernimmt in *Windows-Domänen* die Rolle des sogenannten *Domänen-Controllers* (DC), welcher für die Benutzerverwaltung und Benutzerauthentifizierung verantwortlich ist. Teilweise wird Windows-2000 mit W2K und Windows-2003 mit W2K3 abgekürzt.

Bei Windows-NT4-Domänen wurden ausschließlich proprietäre Netzwerkprotokolle, wie z. B. SAMR und LSA zur Benutzerverwaltung benutzt. Als Authentifizierungsprotokolle kamen NTLMSSP, sowie NETLOGON/SCHANNEL zum Einsatz. Außerdem gab es keinen Verzeichnisdienst, sondern einen *Security Account Manager* (SAM), welcher die Benutzer und Gruppen in der „Windows Registry“ speicherte. Unter Windows-NT4 wurde noch zwischen *Primary Domain Controller* (PDC) und *Backup Domain Controller* (BDC) unterschieden, wobei Schreibzugriffe nur auf dem PDC stattfinden konnten.

Active-Directory integriert eine Reihe moderner Netzwerk - und Authentifizierungsprotokolle, wie z. B. LDAP, DNS, Kerberos² und DIGEST-MD5. Zusätzlich werden aber immer

¹Mit Benutzerkonten sind „User-Accounts“ gemeint.

²Auch als Kerberos-V oder Krb5 bekannt

noch die „alten“ Protokolle von NT4 unterstützt. Der *NT Directory Service* (NTDS) bildet nun die Datengrundlage für alle unterstützten Protokolle und bildet die Hauptkomponente des Active-Directory.

Typischerweise werden die Authentifizierungsprotokolle von Active-Directory zur Authentifizierung von Datei- und Druckerzugriffen mittels des SMB-Protokolls³, aber auch bei E-Mail oder Web Diensten verwendet.

1.1.2 Was ist Samba?

Samba ist ein Softwarepaket, welches Datei- und Druckerdienste kompatibel zu Microsoft-Produkten unter Unix/Linux Betriebssystemen zur Verfügung stellt. Samba ist „Freie Software“ und unter der „GNU General Public License“ verfügbar (siehe [SaTe04], [FSF01], [FSF02] und [WikiPe02]).

Samba wird seit 1992 von Andrew Tridgell und seit 1996 vom Samba-Team entwickelt. Tridgell wollte damals von seinem PC mit „Pathworks for DOS“ auf die Festplatte seiner „Sun Sparcstation“ zugreifen, und schrieb dafür ein Serverprogramm, ohne zu wissen, dass es sich bei dem verwendeten Protokoll um „Server Message Block“ (SMB) handelte. Zu dieser Zeit war Microsoft noch sehr klein am Markt, dies änderte sich jedoch mit der Einführung von Windows-NT3.5 und später Windows-NT4. Microsoft baute seine proprietären Authentifizierungs- und Benutzerverwaltungsprotokolle auf den Protokollen SMB und DCERPC auf. Dabei wurden SMB und DCERPC unter anderem mit nicht dokumentierten Erweiterungen versehen. Das Samba-Team verfolgt das Ziel, möglichst kompatibel zu Microsoft-Produkten zu sein. Dazu wurden einige Methoden zur Netzwerkanalyse entwickelt (siehe [Tridge01] und [Tridge02]). Heute ist Samba in der Version 3.0 als Datei-, Druck- und Anmeldeserver zum größten Teil kompatibel mit Microsoft-Produkten.

Seit 2004 arbeiten einige Mitglieder des Samba-Teams, darunter Andrew Tridgell, Andrew Bartlett, Simo Sorce, Jelmer Vernooij und der Autor dieser Bachelorarbeit an einer Grundüberholung des in die Jahre gekommenen Samba-Quelltextes. Ein Ziel dieser Restrukturierung ist eine Active-Directory-kompatible Implementierung eines Domainen-Controllers mit Unterstützung aller relevanten Protokolle. Die neue Version trägt den Namen Samba 4.0, wobei momentan nur „Technology Previews“, bei denen es sich um reine Entwicklerversionen handelt, veröffentlicht werden. Ein „Alpha-Status“ ist noch nicht erreicht.

1.1.3 Warum ist das Thema wichtig?

Samba wird am häufigsten in Kombination mit dem „Freien Betriebssystem“ Linux (siehe [FSF01], [FSF02] und [WikiPe02]) benutzt. Immer mehr IT-Verantwortliche schätzen die Tatsache, dass jeder den Quelltext von Linux und Samba einsehen und erweitern darf. In proprietärer Software werden häufig Hintertüren und Spionageprogramme vermutet, was „Freie Software“, wie Linux und Samba, gerade für Regierungen interessant macht, unter anderen benutzt der Deutsche Bundestag auch Samba in Kombination mit Linux. In vielen Fällen spielen auch nicht anfallende Lizenzkosten eine große Rolle bei der Entscheidung für Samba. Außerdem können Linux und Samba in den Punkten Stabilität, Performance und Sicherheit sehr gut mit Windows-Servern mithalten.

Das Samba-Team bekommt immer wieder Anfragen von Nutzern, ob oder ab wann Samba einen Domänen-Controller für Active-Directory ersetzen kann. Viele würden gerne von Windows-Servern zu Samba-Servern wechseln, benötigen allerdings zwingend einige spezielle Eigenschaften von Active-Directory.

³Auch CIFS-Protokoll genannt.

1.1.4 Samba soll ein Domänen-Controller für Active-Directory sein

Um aus Samba 4.0 einen zu Active-Directory kompatibelen Domänen-Controller zu machen, wurden unter anderem ein LDAP-Server und ein Kerberos-Server⁴ in das Serverprogramm `smbd` eingebaut.

Zum heutigen Stand kann Samba 4.0 bereits als Active-Directory-Domänen-Controller gegenüber Windows-Domänen-Mitgliedern⁵ auftreten. Im Detail bedeutet dies, dass ein Windows-Mitgliedsrechner bereitwillig einer von Samba 4.0 kontrollierten Domäne beitrifft, im Glauben es handele sich um eine von Windows-2003-Servern kontrollierte Domäne. Anschließend kann man sich mit einem Benutzerkonto der Domäne auf dem Mitgliedsrechner mit Kerberos-Authentifizierung einloggen.

Um als gleichwertiger Domänen-Controller neben Windows-Servern eine Domäne zu verwalten, fehlen Samba 4.0 noch einige Fähigkeiten. Momentan wird der Zugriffsschutz für Verzeichnisobjekte nur sehr trivial geregelt, d. h. ein Administrator darf alle Daten lesen und schreiben, und alle anderen dürfen alles außer den Passwörtern lesen. Außerdem werden geschriebene Daten nicht auf syntaktische Korrektheit geprüft, d. h. es gibt kein sogenanntes Verzeichnis-Schema, welches den Aufbau der Objekte und deren Hierarchie sowie die Syntax der Objektattribute definiert. Die wichtigste fehlende Funktion ist jedoch das Abgleichen der Verzeichnisse mit den „Windows-Kollegen“, um immer ein aktuelles Replikat des Verzeichnisses zu haben. Der Vorgang des Abgleichens wird im allgemeinen „Replikation“ genannt und ist Hauptthema dieser Bachelorarbeit.

1.1.5 Was sind die Schwierigkeiten

Es gibt mittlerweile gute Dokumentationen über die bei Active-Directory benutzten Technologien und Konzepte. Diese werden von Microsoft selbst auf den Internetportalen „Microsoft TechNet“ und „Microsoft MSDN“ angeboten (siehe [MSTN01], [MSTN02], [MSTN03], [MSTN04], [MSTN05], [MSTN06] und [MSDN01]). Diese richten sich aber ausschließlich an Administratoren und enthalten deshalb keine Dokumentation der eingesetzten Netzwerkprotokolle. Entsprechend liegt die Hauptschwierigkeit bei der Implementierung des DRSUAPI-Replikations-Protokolls in der Tatsache, dass es nicht wie z. B. bei LDAP oder Kerberos 5 eine veröffentlichte Spezifikation gibt.

Auf Grund der fehlenden Spezifikation wird die Anwendung der von Tridgell beschriebenen Methoden zur Erarbeitung einer Netzprotokoll-Spezifikation benötigt (siehe [Tridge01] und [Tridge02]).

1.1.6 Warum Teilimplementierung

Eine komplette Implementierung von DRSUAPI-Server und DRSUAPI-Client sowie allen Administrationswerkzeugen würde den Rahmen dieser Bachelorarbeit bei weitem übertreffen. Deshalb werden in dieser Bachelorarbeit nur die im Kapitel „Zielsetzung“ beschriebenen Punkte bearbeitet.

⁴Ein Kerberos *Key Distribution Center* (KDC)

⁵So werden Arbeitsplatzrechner und Server genannt, welche in der Regel über keine lokalen Benutzerkonten verfügen und die Authentifizierungsdienste der Domänen-Controller in Anspruch nehmen.

1.2 Vorarbeiten

1.2.1 Studienarbeit „Active Directory Replikation“

Diese Bachelorarbeit baut auf der Studienarbeit „Active Directory Replikation“ im Fach Datenetze, von Michael Kohlgraf sowie dem Autor dieser Bachelorarbeit, auf (siehe [KoMe2005]). In der Studienarbeit wurden die Schichten 5 (Sitzungsschicht) und 6 (Darstellungsschicht) des OSI-Modells⁶ der DRSUAPI-Protokolls analysiert. Dabei wurde im Besonderen auf die NDR-Kodierung⁷ und die Schnittstellendefinition mit Hilfe von IDL⁸ eingegangen. Allerdings wurde hier noch nicht die zur Replikation benutzte Funktion `DsGetNCChanges()` erkundet.

1.2.2 Entwicklung im Rahmen des Samba-Projekts

Im Rahmen der Entwicklung von Samba 4.0 habe ich das DRSUAPI-Protokoll weiter erforscht. Dabei wurden auch die IDL-Beschreibungen für die zur Replikation notwendigen Funktionen `DsGetNCChanges()`, `DsAddEntry()` und `DsReplicaUpdateRefs()` erarbeitet. Samba 4.0 besitzt ein Testprogramm `smbtorture`, womit Netzwerkprotokolle und interne Schnittstellen getestet werden können. Hier wurde ein neuer Test namens „RPC-DSSYNC“ hinzugefügt, welcher die Funktionsweise von `DsGetNCChanges()` demonstriert. Damit konnte nachvollzogen werden, wie Verzeichnisobjekte samt ihrer Attribute übertragen werden. Allerdings waren noch einige Details unklar, wie z. B. die verbindungspezifische Verschlüsselung diverser Passwort-Attribute.

Samba 4.0 stellt alle zur Replikation benötigten Technologien, wie z.B. Kerberos-5, LDAP und DCERPC zur Verfügung. Diese Bachelorarbeit baut deshalb auf Samba 4.0 auf, wobei die Entwicklungen wieder in den Samba-Quelltext einfließen.

Innerhalb dieser Bachelorarbeit wird nur auf die Entwicklerversion Samba 4.0 Bezug genommen. Deshalb wird in den folgenden Kapiteln auf die explizite Versionsangabe 4.0 verzichtet und mit „Samba“ implizit Samba 4.0 gemeint.

⁶Auch 7-Schichten-Modell genannt (siehe [WikiPe01])

⁷„Network Data Representation“ (siehe [OpGr02])

⁸„Interface Definition Language“ (siehe [OpGr03])

2 Zielsetzung

Im diesem Kapitel werden die Ziele dieser Bachelorarbeit definiert.

2.1 Implementierung der Funktion `libnet_BecomeDC()`

Im Rahmen dieser Bachelorarbeit soll die Funktion `libnet_BecomeDC()` implementiert werden, welche die Server-Rolle¹ eines Samba-Servers in einer von Windows-2003-Servern kontrollierten Active-Directory-Domäne in Mixed-Mode² von Domänen-Mitglied nach Domänen-Controller ändert.

Dazu wird Sambas Management-API *LIBNET* um die Funktion `libnet_BecomeDC()` erweitert. Zur Vollständigkeit wird ebenfalls die Funktion `libnet_UnbecomeDC()` implementiert, welche die Server-Rolle¹ wieder zurück in Domänen-Mitglied ändert. Allerdings wird aus Umfangsgründen nur die Funktion `libnet_BecomeDC()` im Detail erläutert.

2.2 Implementierung einer Schema-API

Innerhalb Samba's DSDB-Komponente³ soll eine Schema-API inklusive Schema-Cache entwickelt werden, um den Zugriff auf das Verzeichnis-Schema⁴ zu ermöglichen. Der Schema-Cache wird durch die C-Strukturen `struct dsdb_schema`, `struct dsdb_class` und `struct dsdb_attribute` realisiert. Wobei `struct dsdb_schema` einen Container für `struct dsdb_class`, bzw. `struct dsdb_attribute` Instanzen bildet. Die Schema-API bietet daraufhin C-Funktionen zum Zugriff auf den Schema-Cache. Für eine erfolgreiche Replikation ist ein Schema-Cache, mit einer Schema-API, zwingend notwendig, da ansonsten einige Teile der Replikations-Nachrichten nicht interpretiert werden können. Im Rahmen dieser Bachelorarbeit wird die Schema-API nur für diesen Zweck implementiert. Eine Nutzung der Schema-API zur Validierung von Schreibzugriffen auf Verzeichnis-Objekte wird hierbei nicht berücksichtigt.

Außerdem wird ein LDB-Modul⁵ mit den Namen „`schema_fsmo`“ entwickelt. Dieses Modul soll später unter anderem Schreibzugriffe auf das Verzeichnis-Schema⁴ validieren, da die Schreibzugriffe unter Active-Directory nur auf einen ausgewählten Server, dem sogenannten „Schema-Master“⁶, möglich sind. Im Rahmen dieser Bachelorarbeit wird allerdings nur das Laden des Schema-Caches zur Nutzung in der Schema-API entwickelt.

¹Folgende Server-Rollen sind definiert: „Alleinstehend“, „Domänen-Mitglied“ sowie „Domänen-Controller“

²Active-Directory-Domänen können in zwei Modi betrieben werden. In „Mixed-Mode“ werden Windows-NT4 *Backup-Domänen-Controller* unterstützt, aus diesem Grund sind einige Funktionen von Active-Directory, wie z.B. geschachtelte Gruppenzugehörigkeiten, deaktiviert. In „Native-Mode“ werden nur Active-Directory-Domänen-Controller unterstützt. Der Modus wird bei der Installation von Active-Directory festgelegt und kann später nur von Mixed-Mode in Native-Mode geändert werden, nicht aber umgekehrt.

³DSDB steht für „Directory Service Database“

⁴Das Verzeichnis-Schema beinhaltet Attribut- und Klassendefinitionen, die den Aufbau und die Hierarchie der im Verzeichnis gespeicherten Objekte regelt.

⁵LDB (Lightweight Database) ist Samba's LDAP-artige Schnittstelle zum Speichern von LDAP-artigen Objekten. LDB ist sehr stark modularisiert und kann mit LDB-Modulen erweitert werden, um die Funktionen hinsichtlich der öffentlichen LDB-API zu erweitern.

2.3 Implementierung eines LDB-Modules zum Speichern der Replikations-Metadaten

Damit Samba die zur Replikation benötigten Metadaten speichert, wird ein LDB-Modul⁵ mit dem Namen „repl_meta_data“ entwickelt. Bei direkten Schreibzugriffen auf Verzeichnis-Objekte soll dieses Modul die zur Replikation benötigten Metadaten erstellen und im „replPropertyMetaData“-Attribute des jeweiligen Objekts speichern. Diese Aufgabe wird im Rahmen dieser Bachelorarbeit exemplarisch nur für die `ldb_add()` Funktion implementiert.

Innerhalb der DSDB-Komponente³ wird außerdem die Funktion `dsdb_extended_replicated_objects_commit()` implementiert, welche die in einer Replikations-Nachricht enthaltenen Änderungen auf die lokalen Verzeichnis-Daten anwendet. Hierbei werden die Änderungen erst unter Nutzung der Schema-API aufbereitet und dann mittels einer sogenannten *LDB-Extended-Operation*⁷ namens `DSDB_EXTENDED_REPLICATED_OBJECTS_OID` über die Funktion `ldb_extended()` an die LDB-Schicht übergeben. Das LDB-Modul „repl_meta_data“ nimmt diese Anfrage entgegen und wendet die Änderungen an.

2.4 Voraussetzungen und Abgrenzungen

Im Laufe der Bearbeitungszeit wird Samba täglich weiterentwickelt und wöchentlich fließen neue Erkenntnisse ein. Deshalb kann in dieser Bachelorarbeit auf manche Entdeckungen nur teilweise eingegangen werden, um den zeitlichen Rahmen nicht zu sprengen.

Da es sich bei dem Thema der Bachelorarbeit um ein sehr spezielles Thema handelt, müssen Grundkenntnisse in den Fachgebieten Informatik, Datenbanken, Datennetze, Betriebssystemen, Verteilten-Systemen sowie Kenntnisse im Bereich von LDAP und Verzeichnisdiensten vorausgesetzt werden.

Außerdem können die vorhandenen Samba C-Funktionen nicht erläutert werden, dazu sind die gesamten Samba 4.0 Quelltexte unter [SaTe01] verfügbar (siehe auch [SaTe02]).

Im Rahmen dieser Bachelorarbeit wird die Behandlung von sogenannten *Linked-Attributes*⁸ für die Implementierung völlig außeracht gelassen.

⁶ In Active-Directory sind folgende 5 „Flexible Service Operation Master“-Rollen definiert: „Schema-Master“, „Domain-Naming-Master“, „RID-Master“, „PDC-Emulator-Master“ und „Infrastructure-Master“ (Siehe auch [MSTN03])

⁷ Mit Hilfe von *LDB-Extended-Operations* kann man dem LDAP-Protokoll, bzw. der LDB-API neue Funktionen hinzufügen.

⁸ *Linked-Attributes* sind Attribute, die Verweise auf andere Verzeichnis-Objekte speichern. Diese Attribute treten immer als Paar auf, wobei es einen *Forward-Link* und einen *Backward-Link* gibt, die jeweils auf das andere Objekt verweisen. Es kann jeweils immer nur der *Forward-Link* geändert werden, der *Backward-Link* wird immer automatisch angepasst. Bei der Replikation wird auch immer nur der *Forward-Link* übertragen, wobei der *Backward-Link* automatisch erzeugt werden muss. In einer Active-Directory-Domäne in Native-Mode werden *Linked-Attributes* separat von den Objekten repliziert.

3 Beschreibung des Replikationssystems

Im folgenden Kapitel wird das grundlegende Replikationsmodell vorgestellt. Außerdem werden die verwendeten Technologien kurz erläutert.

Die Beschreibungen beziehen sich auf das Microsoft Technet Dokument „How the Active Directory Model Works“ [MSTN01]. Zusätzlich liefern vom Samba-Team erstellte Testprogramme bewiesene Fakten, welche zusätzlich in die Beschreibungen einfließen, da in [MSTN01] keinerlei Spezifikationen zu finden sind.

3.1 Was ist ein Verzeichnisdienst

3.1.1 Übersicht

Ein Verzeichnisdienst speichert Verzeichnis-Objekte in einer Baumstruktur, welche *Directory Information Tree* (DIT) genannt wird. Abbildung 3.1 zeigt die Ansicht eines DIT, in einem LDAP-Client mit grafischer Oberfläche. Jedes Objekt hat dabei einen *Distinguished Name* (DN), z. B. „OU=newtop, DC=sub1, DC=w2k3, DC=vmnet1, DC=vm, DC=base“, wobei die einzelnen Komponenten des DN's als *Relative Distinguished Name* (RDN) bezeichnet werden. Im Beispiel ist „OU=newtop“ der RDN des Objekts und „DC=sub1, DC=w2k3, DC=vmnet1, DC=vm, DC=base“ das Vater-Objekt. „OU=newtop, DC=sub1, DC=w2k3, DC=vmnet1, DC=vm, DC=base“ ist also ein Sohn-Objekt von „DC=sub1, DC=w2k3, DC=vmnet1, DC=vm, DC=base“. Im allgemeinen ist der Namensraum des Verzeichnisses global, bezogen auf das Internet. Jeder *Directory Service Agent* (DSA)¹ speichert aber nur einen Teil des gesamten Namensraums. Jeder DSA bietet allerdings das Wurzel-Objekt, welches einen leeren DN besitzt und als *Root DSA-specific Entry* (RootDSE) bezeichnet wird.

¹Die Serverinstanz des Verzeichnisdienst wird im englischen „Directory Service Agent“ genannt.

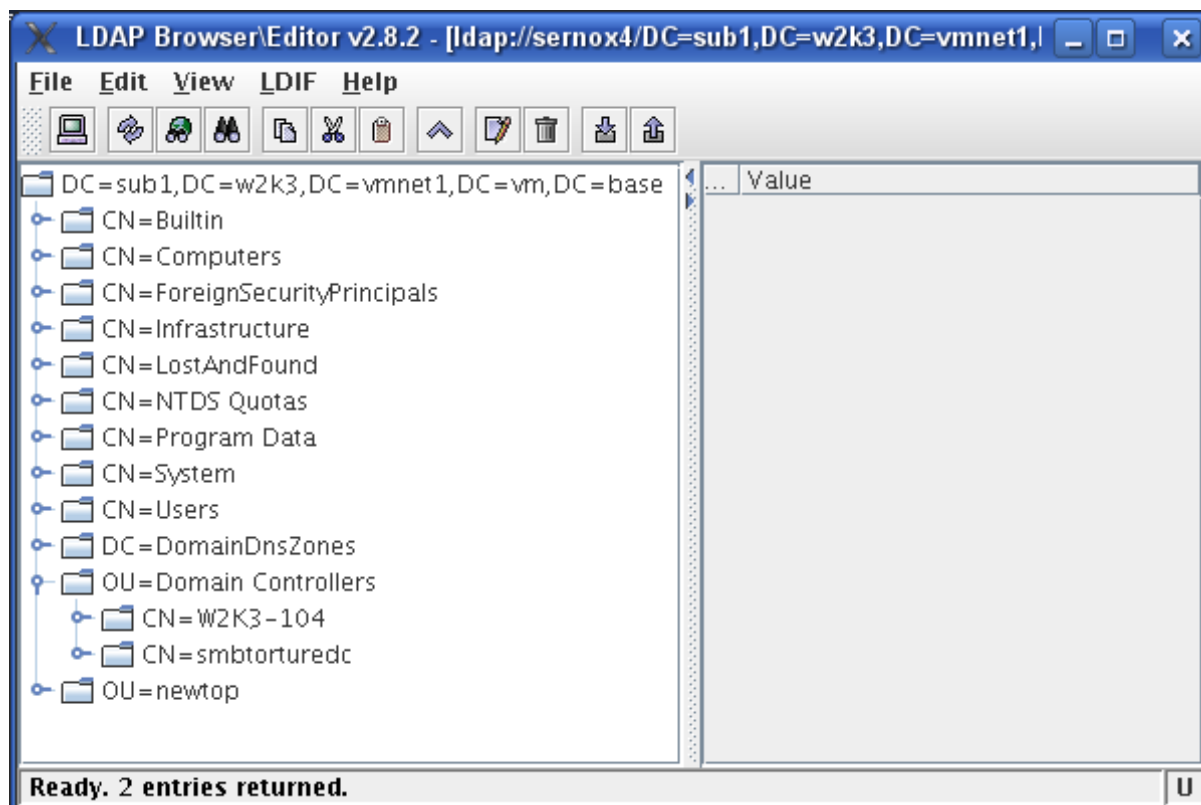


Abbildung 3.1 Ein typischer Verzeichnis-Baum mit Verzeichnis-Objekten.

3.1.2 RootDSE

Abbildung 3.2 zeigt das RootDSE-Objekt eines Windows-2003-Servers im *LDAP Data Interchange Format* (LDIF) [RFC2849]². Mit dem Attribut „namingContexts“ gibt der DSA an, welche Teile des globalen Namensraums er anbietet. Windows-2003-Server bieten außerdem noch die Attribut „defaultNamingContext“, „configurationNamingContext“, „schemaNamingContext“ und „rootDomainNamingContext“ an (vgl. Abbildung 3.2). Jeder Domänen-Controller speichert jeweils mindestens drei schreibbare *Verzeichnis-Partitionen*³: die *Domänen-Partition*, die *Konfigurations-Partition* und die *Schema-Partition*. Abbildung 3.3 zeigt einen sogenannten Domänen-Wald (engl. *Domain Forest*), in dem mehrere Domänen-Bäume (engl. *Domain Trees*) logisch zusammen gefasst werden. Dabei gibt es pro Wald nur eine Konfigurations-Partition und nur eine Schema-Partition. Das Attribut „rootDomainNamingContext“ enthält immer den DN der Wurzel-Domäne im Domänen-Wald, während das Attribut „defaultNamingContext“ immer den DN der lokalen Domäne, für die der DSA verantwortlich ist, enthält. D. h. dass jeder DSA immer nur eine Domäne verwalten kann. Die DN's der Domänen basieren auf ihren DNS-Namen. Im Beispiel ist „sub1.w2k3.vmnet1.vm.base“ der DNS-Domänen-Name. Es gibt außerdem noch *Applikations-Partitionen* und nicht schreibbare Partitions-Kopien. Auf diese beiden Sonder-

²Die erste Zeile enthält immer den DN des Objekts mit vorangestelltem „dn: “. Objekte bestehen immer aus ihrem DN und einer Reihe von Objekt-Attributen. Alle folgenden Zeilen bis zur nächsten Leerzeile sind Paare aus Attribut-Name und Attribut-Wert. Beginnt eine Zeile mit einem Leerzeichen, so gehören die folgenden Zeichen noch zum Attribut-Wert der vorherigen Zeile. Bei Attribut-Werten, die als ASCII-Zeichenkette dargestellt werden können, werden Attribut-Name und Attribut-Wert durch „: “ getrennt. Ist dies nicht möglich, so wird der Attribut-Wert mittels *Base64* [RFC4648] kodiert und „: “ wird zur Trennung von Attribut-Name und Attribut-Wert verwendet.

³ Verzeichnis-Partition (engl. „Directory Partition“) ist ein Synonym für Namens-Kontext (engl. „Naming Context“ (NC)). In dieser Bachelorarbeit wird der Begriff Verzeichnis-Partition verwendet.

fälle wird in dieser Bachelorarbeit nicht eingegangen.

```
dn:
currentTime: 20070304212854.0Z
subschemaSubentry: CN=Aggregate,CN=Schema,CN=Configuration,DC=w2k3,DC=vmnet1,DC=vm,DC=base
dsServiceName: 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
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=base
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=Sites,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
```

Abbildung 3.2 Ein typisches RootDSE-Objekt (ein paar Attribut-Werte wurden zur Verbesserung der Übersicht entfernt).

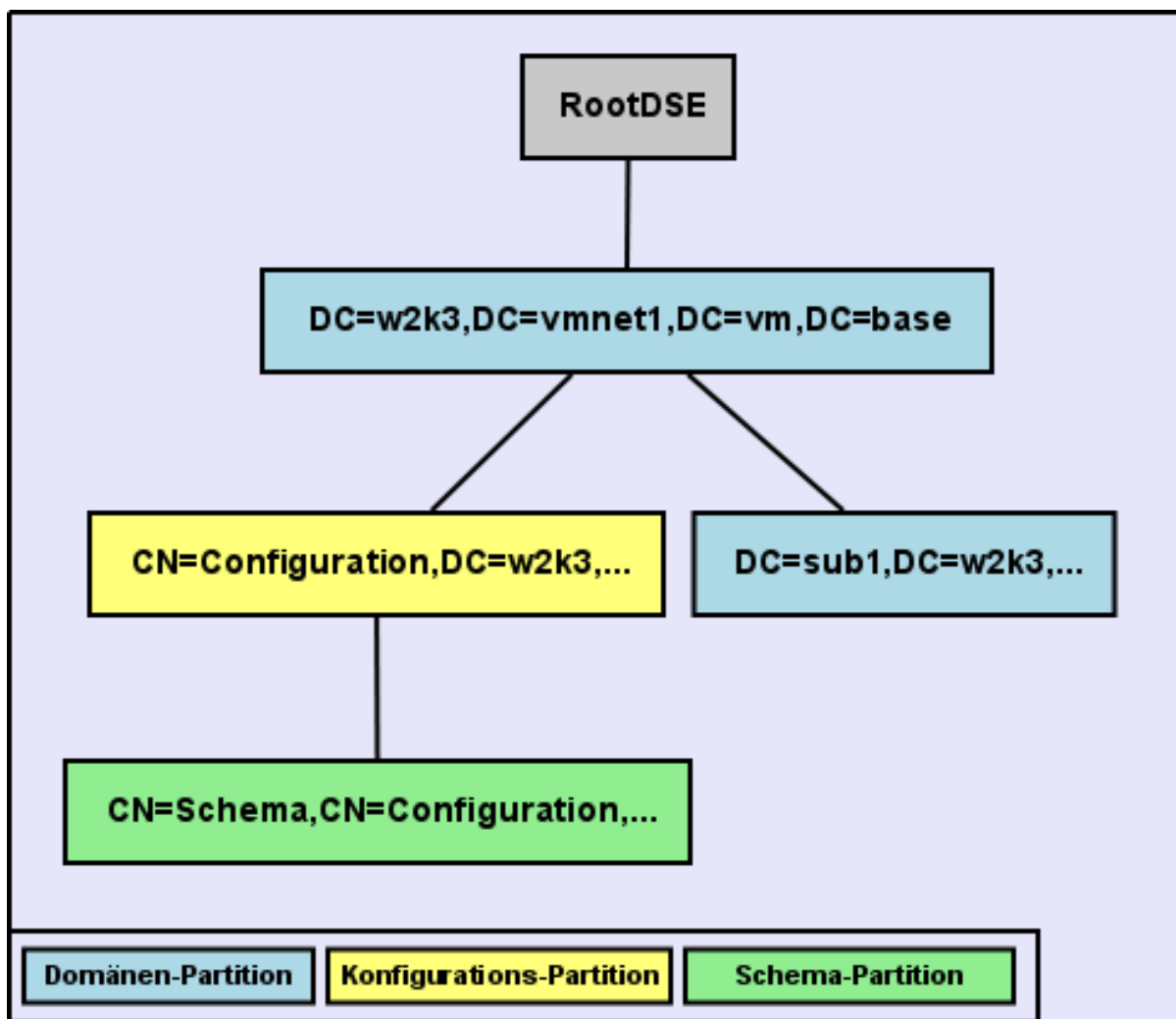


Abbildung 3.3 Ein typischer Domänen-Wald unter Windows-2003.

3.1.3 Verzeichnis-Objekte

Abbildung 3.4 zeigt ein Beispiel für ein Verzeichnis-Objekt. Das Attribut „objectClass“ hat eine besondere Rolle, da Objekte instanziierte Objekt-Klassen sind. Das Verzeichnis-Schema enthält die Definitionen von Attributen und Objekt-Klassen, welche ihrerseits Instanzen der Objekt-Klassen „attributeSchema“ bzw. „classSchema“ sind. Objekt-Klassen definieren, welche Attribute für ein Objekt zwingend notwendig oder optional sind. Außerdem können Objekt-Klassen Eigenschaften von anderen Klassen erben, z. B. ist jede Klasse von der Superklasse „top“ abgeleitet. Für Attribute wird unter anderem definiert, ob mehrere Attribut-Werte für ein Objekt zugelassen sind oder welchen Syntax-Anforderungen Attribut-Werte genügen müssen. Wer mehr über Vererbung und andere Details des in Active-Directory verwendeten Verzeichnis-Schemas wissen möchte, findet unter [MSTN04] eine gute Beschreibung (siehe auch source/setup/schema.ldif unter [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
```

Abbildung 3.4 Ein typisches Verzeichnis-Objekt mit Attributen.

3.1.4 Die Partitionen

Die Schema-Partition

In der Schema-Partition werden alle Informationen zum Verzeichnis-Schema gespeichert (siehe auch [MSTN05]).

Die Konfigurations-Partition

In der Konfigurations-Partition werden alle Konfigurations-Informationen gespeichert, welche den ganzen Domänen-Wald betreffen (siehe auch [MSTN05]). Dazu gehören z.B. Informationen zu allen Domänen-Controllern und deren Standorte (engl. *Site*). Außerdem werden hier Informationen über alle im Domänen-Wald vorhandenen Verzeichnis-Partitionen gespeichert.

Die Domänen-Partition

In der Domänen-Partition werden alle für die jeweilige Domäne relevanten Daten gespeichert (siehe auch [MSTN05]). Dazu gehören Benutzer- und Computer-Konten sowie Benutzer-Gruppen. Außerdem werden hier auch domänen-bezogene Konfigurations-Daten abgelegt.

3.2 Grundlegende Protokolle

In Active-Directory kommen eine Reihe grundlegender Protokolle zum Einsatz. Die Basis bilden dabei das *Internet Protokoll* (IP) [RFC791] sowie darauf aufbauend das *Transmission Control Protocol* (TCP) [RFC793] und das *User Datagram Protocol* (UDP) [RFC768]. Zur Auflösung von Namen zu IP-Adressen werden die Protokolle *Domain Name Service* (DNS) [RFC1034,RFC1035] sowie *Netbios Name Service* [RFC1001,RFC1002] benutzt.

3.2.1 Authentifizierungsprotokolle

Das Zusammenspiel der Netzwerkprotokolle ist in Abbildung 3.5 dargestellt. Alle Protokolle oberhalb von TCP verwenden authentifizierte Sitzungen, wobei als Authentifizierungsprotokolle *Kerberos 5* (KRB5) [RFC4120] und *New Technology Lan Manager Secure Service Provider* (NTLMSSP) [Gla2006] zum Einsatz kommen. Teilweise werden zur Aushandlung des jeweils benutzten Protokolls Zusatzprotokolle wie *Generic Security Service Application Program Interface* (GSSAPI) [RFC2743], *Simple and Protected Negotiation* (SPNEGO) [RFC4178] und/oder *Simple Authentication and Security Layer* (SASL) [RFC2222] benutzt.

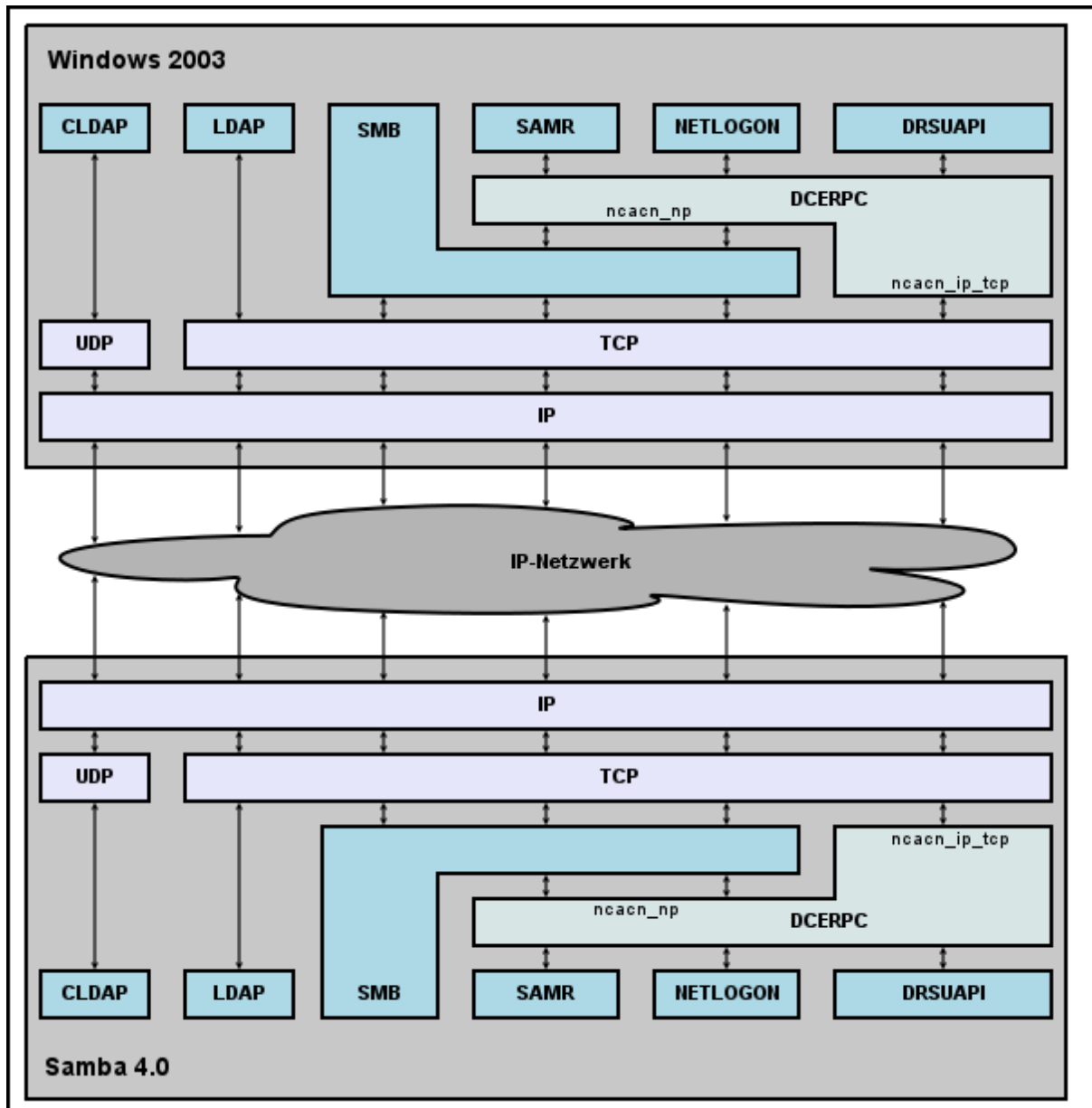


Abbildung 3.5 Übersicht der Netzwerk-Protokolle (vgl. „Replication and LDAP Client-Server Architecture“ in [MSTN01]).

3.2.2 LDAP

Das *Lightweight Directory Access Protocol* (LDAP) [RFC4510] ist ein Protokoll, das den Zugriff auf Verzeichnis-Objekte und deren Attribute ermöglicht. Das Protokoll basiert auf TCP und verwendet den TCP-Port 389. Typischerweise muss sich ein Client mit der `ldap_bind()` beim Server authentifizieren. Dann stehen die Funktionen `ldap_add()`, `ldap_modify()`, `ldap_modify_rdn()` und `ldap_delete()` zur Bearbeitung eines bestimmten Verzeichnis-Objekts zur Verfügung. Außerdem dient die Funktion `ldap_search()` zum Auflisten von Verzeichnis-Objekten. Durch den Base-DN wird die Suche auf einen bestimmten Unterbaum der Verzeichnis-Hierarchie beschränkt. Zur weiteren Eingrenzung der Suche wird ein Suchbereich durch `LDAP_SCOPE_BASE` (nur Base-DN), `LDAP_SCOPE_ONELEVEL` (nur direkte Kinder-Objekte von Base-DN) oder

LDAP_SCOPE_SUBTREE (alle Objekte unterhalb von Base-DN und Base-DN selber) angegeben. Um nur Objekte mit bestimmten Attribut-Werten aufzulisten, bestimmt der Suchfilter, z. B. „(&(objectClass=user)(name=Administrator))“, welche Objekte der Server an den Client senden soll. Schließlich gibt eine Attribut-Liste an, welche Attribute der gefundenen Objekte geliefert werden sollen. Beim Suchfilter und der Attribut-Liste darf „*“ als Wildcard benutzt werden.

3.2.3 CLDAP

Das *Connection-less Lightweight Directory Access Protocol* (CLDAP) [RFC3352] ist die verbindungslose Variante von LDAP. Sie basiert auf UDP und benutzt keine Authentifizierung. Demzufolge werden nur Suchanfragen unterstützt.

3.2.4 SMB

Das Protokoll *Server Message Block* (SMB)⁴ [Her2003] ist verbindungsorientiert und basiert auf TCP. Dabei werden sowohl TCP-Port 139 als auch TCP-Port 445 verwendet. Bei Port 139 liegt zwischen SMB und TCP formell noch der *Netbios Session Service* [RFC1001,RFC1002], was allerdings nur beim Verbindungsaufbau durch zwei zusätzliche Protokoll-Nachrichten bemerkbar ist. SMB bietet Freigaben von Dateisystemen und Druckern. Die spezielle „IPC\$“-Freigabe ermöglicht darüberhinaus Kommunikation über *Named Pipes*.

3.2.5 DCERPC

Distributed Computing Environment / Remote Procedure Call (DCERPC) [OpGr01] ist ein generisches Protokoll zur Übertragung von Funktionsaufrufen auf einem entfernten Computer. DCERPC kann unterschiedliche Transport-Protokolle verwenden. In Active-Directory werden TCP („ncacn_ip_tcp“) und SMB-Named-Pipes („ncacn_np“) als Transport-Protokolle verwendet. Bei DCERPC werden verwandte Funktionen zu Schnittstellen zusammengefasst. Diese Schnittstellen werden mit Hilfe der *Interface Definition Language* (IDL) [OpGr03] spezifiziert. Die Übertragung der Funktionsparameter und Rückgabewerte wird mit Hilfe der *Network Data Representation* (NDR) [OpGr02] realisiert. DCERPC kümmert sich um die Sitzungsverwaltung inklusive Authentifizierung und optionaler Verschlüsselung. Server verwenden bei „ncacn_ip_tcp“ meist keine festen TCP-Ports. Statt dessen registrieren sie sich beim *Endpoint-Mapper* (siehe [OpGr04]), dort können Clients dann die jeweils benutzten TCP-Ports erfragen.

3.2.6 DCERPC basierte Schnittstellen/Protokolle

In Active-Directory werden eine Reihe von DCERPC basierter Protokolle verwendet. Die von Microsoft verwendeten IDL-Spezifikationen sind allerdings nicht veröffentlicht. Die besten öffentlichen, aber unvollständigen IDL-Spezifikationen zu diesen Protokollen werden vom Samba-Team und einer Reihe anderer Freiwilliger gepflegt (siehe [SaTe03]).

3.2.7 SAMR

Das SAMR-Protokoll stellt Funktionen zur Benutzer- und Gruppenverwaltung bereit (siehe samr.idl unter [SaTe03]). Der Name kommt vom, unter Windows-NT 4.0 verwendeten, *Security Account Manager* (SAM).

⁴SMB ist auch unter dem Namen *Common Internet File System* (CIFS) bekannt.

3.2.8 NETLOGON

Das NETLOGON-Protokoll stellt Funktionen zur Authentifizierung innerhalb von Windows-Domänen zur Verfügung. Diese Funktionen werden unter anderem zur NTLMSSP-Authentifizierung benutzt. Außerdem stellt die NETLOGON-Schnittstelle Funktionen für die unter Windows-NT 4.0 verwendete SAM-Replikation bereit (siehe netlogon.idl unter [SaTe03]).

3.2.9 DRSUAPI

Das Protokoll *Directory Replication Service Update API* (DRSUAPI) bietet Funktionen, die zur Active-Directory-Replikation verwendet werden (siehe drsuapi.idl unter [SaTe03]). Außerdem wird noch die `DsCrackNames()` angeboten, welche Namen von Verzeichnis-Objekten zwischen unterschiedlichen Namensformaten umwandelt.

3.3 Grundlegende Konzepte des Replikationssystems

3.3.1 Source-DSA vs. Destination-DSA

Bei der Replikation müssen Änderungen am Datenbestand jeweils von einem Server, dem sogenannten Source-DSA, zu einem anderen Server, dem sogenannten Destination-DSA, übertragen werden.

3.3.2 Singlemaster-Replikation vs. Multimaster-Replikation

Für die Replikation von Datenbeständen unterscheidet man grundsätzlich zwischen dem *Singlemaster*- und dem *Multimaster-Modell*.

Singlemaster-Replikation

Bei der relativ einfachen *Singlemaster-Replikation* sind Schreibzugriffe nur auf einem Server möglich und alle anderen Server dürfen nur Lesezugriff anbieten, da der Datenbestand ansonsten inkonsistent werden kann.

Multimaster-Replikation

Bei der komplexeren *Multimaster-Replikation* sind Schreibzugriffe auf allen Servern möglich. Hier müssen Inkonsistenzen aber fest im Konzept verankert werden. Konflikte sollten daher durch Algorithmen gelöst werden, damit jeder Server eigenständig arbeiten kann. D. h. ein Konflikt wird auf jedem Server auf die selbe Weise aufgelöst, damit der Datenbestand wieder konsistent wird.

Bei Active-Directory wird die *Multimaster-Replikation* verwendet, wobei allerdings bestimmte systemkritische Schreiboperationen nur auf jeweils einem Server dem *FSMO-Role-Owner*⁶ erlaubt sind.

3.3.3 Push-Replikation vs. Pull-Replikation

Generell wird bei der Replikation von Datenbeständen zwischen *Push-Replikation* und *Pull-Replikation* unterschieden.

Push-Replikation

Bei der *Push-Replikation* entscheidet der Server, auf dem die Daten geändert wurden, wann und welche Daten er an seine Replikations-Partner schicken muss. Um dieses System effizient zu halten, muss der Source-DSA für jeden Destination-DSA buchführen, welche Daten schon übertragen wurden. Durch Verbindungsabbrüche oder andere Fehler kann es aber vorkommen, dass Daten nicht korrekt beim Destination-DSA ankommen, der Source-DSA aber von einer fehlerfreien Übertragung ausgeht. In diesem Fall wird der Destination-DSA einige Änderungen niemals erhalten.

Pull-Replikation

Bei der *Pull-Replikation* weiß jeder der Server, welche Daten er schon empfangen hat. Der Destination-DSA sagt dem Source-DSA bei jeder Replikations-Anfrage, welche Daten er schon hat. Darauf hin kann der Source-DSA bestimmen, welche Daten er dem Destination-DSA bereitstellen muss, damit die Datenbestände auf beiden Servern übereinstimmen. Dieses Verfahren ist bei weitem nicht so empfindlich gegenüber Übertragungsfehlern, wie die *Push-Replikation*. Die *Pull-Replikation* wird typischerweise in konfigurierbaren Intervallen ausgelöst, da der Destination-DSA nicht weiß, wann Änderungen auf dem Source-DSA

stattfinden. Zusätzlich kommt oft ein *Change-Notify* genannter Mechanismus zum Einsatz. Dabei schickt ein Source-DSA Änderungsbenachrichtigungen an den Destination-DSA, welcher dann mit einer sofortigen *Pull-Replikation* reagieren kann.

Bei Active-Directory wird die *Pull-Replikation* mit *Change-Notify* benutzt.

3.3.4 Log-basierte vs. Status-basierte Replikation

Um eine Replikation von Daten effizient zu gestalten, ist es nötig, über Schreibzugriffe Buch zu führen. Hierbei gibt es auch zwei Ansätze, den Log-basierten und den Status-basierten.

Log-basierte Replikation

Bei der Log-basierten Replikation wird jede Änderung in eine Logdatei geschrieben. Der Vorteil bei diesem Verfahren ist, dass jede Änderung zurückverfolgt werden kann und alte Zustände wiederhergestellt werden können. Der benötigte Speicherplatz wächst allerdings linear mit der Anzahl der Änderungen und nicht mit der Größe des jeweils aktuellen Datenbestandes.

Status-basierte Replikation

Die Status-basierte Replikation geht hier einen anderen Weg, und zwar wird nur der jeweils aktuelle Datenbestand gespeichert. Dabei wird eine Sequenznummer verwendet, welche bei jeder Änderung inkrementiert wird. Bei jeder Änderung wird dem geänderten Datensatz die aktuelle Sequenznummer zugeordnet. Bei der Replikation muss nur bekannt sein, bis zu welcher Sequenznummer schon erfolgreich übertragen wurde, dann müssen nur noch Datensätze mit einer höheren Sequenznummer übertragen werden. Der Speicherplatz ist hier linear zur Größe des aktuellen Datenbestandes.

Active-Directory benutzt die Status-basierte Replikation.

3.3.5 Store-and-Forward-Replikation

Bei der Replikation mit vielen beteiligten Servern ist es wünschenswert, dass nicht jeder Server mit jedem anderen replizieren muss. Das Prinzip der *Store-and-Forward-Replikation* ermöglicht eine indirekte Übertragung von Änderungen zwischen zwei Servern, die keine direkte Netzwerkverbindung haben. Die Replikation könnte wie folgt aussehen: Server A repliziert mit Server B, Server B repliziert mit Server C. Bei solch einer *Replikations-Topologie* werden Änderungen auf Server A erst nach Server B und von dort aus nach Server C repliziert. Diese Topologie kann beliebig erweitert werden. Jeder Server weiß, welche Änderungen er von welchem anderen Server schon erhalten hat. Der Weg (direkt oder indirekt), auf den die Änderungen übertragen werden, ist dabei irrelevant.

Bei Active-Directory wird das *Store-and-Forward-Prinzip* angewendet. Auf den Aufbau der verwendeten *Replikations-Topologie* (siehe [MSTN02]) wird in dieser Bachelorarbeit nicht eingegangen.

3.4 Grundlegende Attribute und Strukturen

Um die in Active-Directory verwendete Replikation zu beschreiben, werden eine Reihe grundlegender Attribute, Strukturen und Begriffe benötigt. Diese werden im Folgenden jeweils kurz erklärt.

3.4.1 Das „objectGUID“ Attribut

Da Objekte innerhalb einer Verzeichnis-Partition verschoben bzw. umbenannt werden können, ist der Objekt-DN nicht konstant eindeutig. Jedes Verzeichnis-Objekt hat deshalb einen sogenannten *Global Unique Identifier* (GUID)⁵, der sich niemals ändert. Grob gesehen ist eine GUID eine 128-Bit (16 Byte) Zufallszahl. Teilweise wird eine GUID durch eine IDL-Struktur beschrieben (siehe Abbildung 3.6), deren NDR-Kodierung ein Array mit 16 Bytes ist. Oft wird aber auch eine String-Repräsentation verwendet, z. B. „077157f1-eccc-47bf-ba03-9753c93e2e73“. Die GUID des Verzeichnis-Objekts wird NDR-kodiert im Attribut „objectGUID“ gespeichert.

3.4.2 Das „objectSid“ Attribut

Domänen, Benutzer-Konten und Gruppen haben zusätzlich einen *Security Identifier* (SID)⁶, welcher zur Identifikation bei der Berechtigungsüberprüfung benutzt wird. Die SID wird als `struct dom_sid`⁷ in IDL beschrieben (siehe Abbildung 3.6). Üblicherweise wird allerdings die String-Repräsentation benutzt, z. B. „S-1-5-21-4258141497-2604123113-439883119-1286“ verwendet. Dabei setzt sich die SID aus der Domänen-SID und dem *Relative Identifier* (RID) für den Benutzer oder die Gruppe zusammen. Im Beispiel ist „S-1-5-21-4258141497-2604123113-439883119“ die Domänen-SID und „1286“ die RID des Benutzers. Die SID eines Verzeichnis-Objekts wird NDR-kodiert im Attribut „objectSid“ gespeichert.

3.4.3 Die Struktur „DsReplicaObjectIdentifier“

Innerhalb des DRSUAPI-Protokolls wird ein Objekt durch einen *Verzeichnis-Objekt-Identifizier*⁸ referenziert, der sich aus der Objekt-GUID, der Objekt-SID und dem Objekt-DN zusammensetzt. Dabei muss jeweils immer eins der drei Elemente einen gültigen Wert haben, die anderen zwei sind optional und können auch mit NULL-Bytes initialisiert werden.

⁵Obwohl es im deutschen eigentlich „der GUID“ heißen müsste, wird in der Literatur immer von „einer GUID“ gesprochen. Eine GUID ist auch als *Universal Unique Identifier* (UUID) bekannt.

⁶Auch hier wird in der Literatur von „einer SID“ gesprochen.

⁷`struct dom_sid2` und `struct dom_sid28` unterscheiden sich minimal in der NDR-kodierung. Dieser Unterschied kann hier aber vernachlässigt werden.

⁸Zum größten Teil wird `struct drsuapi_DsReplicaObjectIdentifier` benutzt (siehe Anhang A2 ab Zeile 128). Es gibt allerdings auch Abwandlungen, z. B. `struct drsuapi_DsReplicaObjectIdentifier3`, welche aber nur auf NDR-Ebene Unterschiede aufweisen.


```
/* 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;
```

Abbildung 3.6 Die IDL-Beschreibung der Verzeichnis-Objekt-Identifizier Grundstrukturen.

3.4.4 Das Objekt „NTDS Settings“

Jeder DSA speichert seine eigene Konfiguration in einem Objekt der Klasse „nTDSDSA“ und dem RDN „CN=NTDS Settings“. Ein Beispiel-Objekt zeigt Abbildung 3.7. Besonders relevant sind hier die Attribute „objectGUID“ und „invocationId“. Beide enthalten eine NDR-kopierte GUID. Die sogenannte *DSA-GUID*, die den DSA eindeutig repräsentiert und sich nie ändert, ist im Attribut „objectGUID“ gespeichert. Der jeweils aktuell von DSA benutzte Verzeichnis-Datenbestand hat seine eigene GUID, die sogenannte *DSA-Invocation-Id*, welche im Attribut „invocationId“ gespeichert wird. Die *DSA-Invocation-Id* wird bei der Installation des DSA generiert. Beim zu allererst installierten DSA im Domänen-Wald haben die *DSA-GUID* und die *DSA-Invocation-Id* identische Werte. Nur in besonderen Fällen, z. B. beim Zurückspielen einer Sicherung, wird die *DSA-Invocation-Id* geändert.

```
dn: 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
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/C99YL1FZw==
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
```

Abbildung 3.7 Ein typisches „NTDS Settings“ Objekt.

3.4.5 Das Attribut „highestCommittedUSN“

Bei Active-Directory wird eine 64-Bit („unsigned“) *Update Sequence Number* (USN) benutzt, welche bei jedem schreibenden Zugriff inkrementiert wird. Diese USN ist jeweils lokal für den jeweiligen DSA. Die jeweils aktuell höchste USN stellt jeder DSA im Attribut „highestCommittedUSN“ des RootDSE bereit (siehe Abbildung 3.2).

3.4.6 Die Strukturen „DsReplicaCursor“ und „DsReplicaCursorCtr“

Um das *Store-and-Forward-Prinzip* umzusetzen, wird in Active-Directory jeder Änderungsvorgang eindeutig durch die DSA-Invocation-Id und die zugehörige USN beschrieben. Diese Daten werden im DRSUAPI-Protokoll als struct `drsuapi_DsReplicaCursorCtr` übertragen, d. h. als Array mit struct `drsuapi_DsReplicaCursor` Elementen (siehe Abbildung 3.8). Es gibt diese Strukturen auch mit zusätzlichem Zeitstempel, welcher die Zeit der letzten erfolgreichen Replikation angibt. struct `drsuapi_DsReplicaCursorCtr` wird auch als *Up-To-Date-Vector* genannt.

```
/* 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;
```

Abbildung 3.8 Die IDL-Beschreibung der *Up-To-Date-Vector* Grundstrukturen.

3.4.7 Die Struktur „DsReplicaHighWaterMark“

Bei der direkten Replikation zwischen zwei Servern wird ein sogenannter *High-Water-Mark* ausgetauscht. Abbildung 3.9 zeigt seine Struktur. Der *High-Water-Mark* enthält die dem *Destination-DSA* von *Source-DSA* schon bekannten USNs.

```
/* 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;
```

Abbildung 3.9 Die IDL-Beschreibung der *High-Water-Mark* Struktur.

3.4.8 Die Attribute „repsFrom“ und „repsTo“

Gespeichert wird der *High-Water-Mark*, zusammen einigen anderen Informationen zum dazugehörigen DSA, im Attribut „repsFrom“. Für jeden Source-DSA, von dem repliziert wird, existiert ein eigener Attribut-Wert, welcher die NDR-kodierte Form von struct `repsFromToBlob` enthält (Abbildung 3.10). Für jeden Destination-DSA, der mittels *Change-Notify* über Änderungen informiert werden soll, ist analog ein Attribut-Wert des Attributs „repsTo“ vorhanden. Die DSAs, mit denen der lokale DSA in irgendeiner Form repliziert, werden auch *Replikations-Nachbarn* genannt.

```

/* 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;
} repsFromTo10therInfo;

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] repsFromTo10therInfo *other_info;
    [value(ndr_size_repsFromTo10therInfo(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 */

```

Abbildung 3.10 Die IDL-Beschreibung der „repsFromToBlob“ Struktur.

```

/* 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;

```

Abbildung 3.11 Die Bit-Flags „DsReplicaNeighbourFlags“.

3.4.9 ASN.1 „Object Identifier“

Attribute und Objekt-Klassen werden im Schema eindeutig über einen *ASN.1 Object Identifier* (OID) identifiziert, z. B. „1.3.6.1.4.1.7165.4.255.1“ (siehe [ITUT01]). Im Active-Directory-Schema wird der OID-Wert als ASCII-String gespeichert. Bei Attributen wird das „attributeID“ Attribut und bei Klassen das „governsID“ Attribut gespeichert. Die OID hat nichts mit der „DsReplicaObjectIdentifier“ Struktur zu tun!

3.4.10 Das „Prefix-Mapping“

Innerhalb des DRSUAPI-Protokolls werden Attribute und Objekt-Klassen nicht mit dem Namen referenziert, sondern über einen 32-Bit (unsigned) Integerwert. Dieser 32-Bit-Wert wird vom OID-Wert abgeleitet. Dabei wird der OID-String beim letzten Vorkommen des Zeichens „,“, in den Prefix und das letzte Element aufgetrennt (siehe Abbildung 3.12). Da die im Schema verwendeten OIDs sich größtenteils nur im letzten Element unterscheiden, wird eine Tabelle mit sogenannten *Prefix-Mappings* benutzt. Hier werden die OID-Präfixe auf 16-Bit (unsigned) Integerwerte abgebildet. Der 16-Bit-Präfix wird in die 16 höherwertigen Bits des 32-Bit-Wertes geschrieben. Das letzte Element wird auch als 16-Bit (unsigned) Integerwert interpretiert und in die niederwertigen 16-Bit des 32-Bit-Wertes geschrieben. Abbildung 3.12 zeigt die Liste mit den bei Active-Directory benutzten *Prefix-Mappings*. Der 32-Bit-Wert wird bei Attributen auch als *AttId* bezeichnet.

```

/* 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
*/

```

Abbildung 3.12 Beispiel für das *Prefix-Mapping* inklusive der in Active-Directory verwendeten Basis-Tabelle.

3.4.11 Die Strukturen „DsReplicaOIDMapping“ und „DsReplicaOIDMapping_Ctr“

Innerhalb des DRSUAPI-Protokolls wird die *Prefix-Mapping* Tabelle als Array mit struct drsuapi_DsReplicaOIDMapping Elementen übertragen (siehe Abbildung 3.13).

```

/* aus drsuapi.idl unter [5aTe03] bzw. Anhang A2 ab Zeile 327 */
typedef [inopush,nopull] struct {
    [range(0,10000),value(ndr_size_drsuapi_DsReplicaOID_oid(oid, 0))] uint32 __ndr_size;
    [size_is(__ndr_size),charset(D05)] 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;

```

Abbildung 3.13 Die IDL-Beschreibung der *Prefix-Mapping* Grundstrukturen.

3.4.12 Originating-Updates vs. Replicating-Updates

Änderungen, die durch Client-Applikationen generiert wurden, also nicht durch die Replikation hervorgerufen werden, nennt man *Originating-Updates*. Replizierte Änderungen werden als *Replicating-Updates* bezeichnet.

3.4.13 Das Attribute „replPropertyMetaData“

Die zur Replikation benötigten Metadaten werden für jedes Verzeichnis-Objekt im „replPropertyMetaData“ Attribut gespeichert. In Active-Directory werden nicht alle Attribute repliziert, und manche Attribute werden bei einer Such-Anfrage dynamisch erzeugt. Das Attribut „replPropertyMetaData“ ist ein nicht repliziertes Attribut, welches nur eine Bedeutung für den lokalen DSA hat. Der Inhalt dieses Attributs ist NDR-kodiert und als struct replPropertyMetaDataBlob in IDL beschrieben (siehe Abbildung 3.14). Die Metadaten für jedes replizierte Attribut werden in einem Array aus struct replPropertyMetaData1 Elementen gespeichert. Attribute werden über die 32-Bit *AttId* referenziert. Bei jedem *Originating-Update* eines Attributs wird das Versionsfeld inkrementiert und die Änderungszeit, die DSA-Invocation-Id und die USN („originating“ und „local“) gesetzt. Bei einem *Replicating-Update* werden alle Metadaten-Elemente, außer der lokalen USN, als struct drsuapi_DsReplicaMetaDataCtr vom Source-DSA zum Destination-DSA übertragen. Die lokale USN wird dann beim Einspielen der replizierten Änderung vom Destination-DSA hinzugefügt.

```

/* 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;

```

Abbildung 3.14 Die IDL-Beschreibung der „replPropertyMetaData“ und „DsReplicaMetaDataCtr“ Grundstrukturen.

3.4.14 Die Struktur „DsReplicaObject“

Im DRSUAPI-Protokoll werden Verzeichnis-Objekte als struct drsuapi_DsReplicaObject übertragen (siehe Abbildung 3.15). Ein Objekt besteht aus einem *Verzeichnis-Objekt-Identifizier* und einem Attribut-Array aus struct drsuapi_DsReplicaAttribute Elementen. Jedes Attribut wird wieder über die *AttId* referenziert. Außerdem hat jedes Attribut ein Werte-Array aus struct drsuapi_DsAttributeValue Elementen. Jeder Wert besteht aus einem 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;

```

Abbildung 3.15 Die IDL-Beschreibung der „DsReplicaObject“ Grundstrukturen.

3.4.15 Die Listen-Elemente „DsReplicaObjectListItem“ und „DsReplicaObjectListItemEx“

Im DRSUAPI-Protokoll werden Verzeichnis-Objekte als einfach verkettete Liste übertragen. Hierbei gibt es zwei Varianten. Entweder enthalten die Listen-Elemente nur das struct `drsuapi_DsReplicaObject` oder zusätzlich noch die GUID des Vater-Objekts sowie ein Metadaten-Array, welches die selbe Größe hat wie das Attribut-Array, wobei die Array-Elemente mit dem gleichen Index jeweils auch zusammen gehören (siehe Abbildung 3.16).

```

/* 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;

```

Abbildung 3.16 Die IDL-Beschreibung der „DsReplicaObjectListItem“ und „DsReplicaObjectListItemEx“ Listenelemente.

3.4.16 Die Attribute „uSNCreate“ und „uSNChanged“

Die Attribute „uSNCreate“ und „uSNChanged“ enthalten jeweils die lokale USNs. „uSNCreate“ wird bei der Erzeugung des Verzeichnis-Objekts auf dem lokalen DSAs gesetzt. „uSNChanged“ enthält die USN der letzten Attribut-Änderung des Verzeichnis-Objekts. Analog zu den USN-Attributen gibt es noch die Attribute „whenCreated“ und „whenChanged“, welche Zeitstempel in der Form „20070309000909.0Z“ enthalten.

3.4.17 Das Attribut „instanceType“

Das Attribut „instanceType“ enthält durch Disjunktion verknüpfte Bit-Flags. Jedes Verzeichnis-Objekt besitzt dieses Attribut. Bei schreibbaren Objekten ist das INSTANCE_TYPE_WRITE Bit-Flag gesetzt, außerdem hat das Wurzel-Objekt der Wurzel-Domäne im Domänen-Wald zusätzlich das INSTANCE_TYPE_IS_NC_HEAD Bit-Flag gesetzt. Alle anderen Wurzel-Objekte einer Verzeichnis-Partition haben zusätzlich die Bit-Flags INSTANCE_TYPE_IS_NC_HEAD und INSTANCE_TYPE_NC_HEAD_ABOVE gesetzt. Abbildung 3.17 zeigt alle möglichen Bit-Flags.

```
/* aus source/dsdb/common/flags.h unter [5aTe01] */
#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
```

Abbildung 3.17 Die Bit-Flags für das „instanceType“ Attribut.

3.4.18 Das Attribut „systemFlags“

Das Attribut „systemFlags“ enthält durch Disjunktion verknüpfte Bit-Flags. Es ist nur bei systemkritischen Verzeichnis-Objekten vorhanden, um z. B. ein Löschen zu verhindern. Abbildung 3.18 zeigt alle möglichen Bit-Flags.

```
/* aus source/dsdb/common/flags.h unter [5aTe01] */
#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_DELETE 0x80000000
```

Abbildung 3.18 Die Bit-Flags für das „systemFlags“ Attribut.

3.4.19 Das Attribut „msDs-Behavior-Version“

Um von Release zu Release neue Funktionen in Active-Directory einzubauen, verwendet Microsoft sogenannte Funktions-Level (siehe [MSTN06]). Domänen-Controller, Domänen und Domänen-Wälder haben jeweils ein Funktions-Level zugeordnet, welches im Attribut „msDs-Behavior-Version“ gespeichert wird. Abbildung 3.19 zeigt die möglichen Werte.

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

Abbildung 3.19 Die möglichen Werte für das „msDs-Behavior-Version“ Attribut.

3.5 Originating-Updates

Um die in Active-Directory verwendete *Status-basierte Store-and-Forward-Replikation* umzusetzen, müssen eine Reihe von Metadaten vom DSA gepflegt werden. Nachdem alle benötigten Strukturen, Begriffe und Attribute erklärt sind, wird die Behandlung der Metadaten bei *Originating-Updates* analysiert.

Generell werden Änderungen bei einem *Originating-Update* anhand des Verzeichnis-Schemas validiert.

Befindet sich der Domänen-Wald im Funktions-Level DS_BEHAVIOR_2000, so ist das kleinste zu replizierende Element ein Attribut samt aller seiner Attribut-Werte. Die Funktions-Level DS_BEHAVIOR_2003_INTERIM und DS_BEHAVIOR_2003 ermöglichen das unabhängige Replizieren von einzelnen Attribut-Werten, wenn es sich um ein sogenanntes *Linked-Attributes* handelt. Hier wird auf *Linked-Attributes* allerdings nicht näher eingegangen.

3.5.1 MetadatenSpeicherung beim Originating-Add

Beim Erstellen eines neuen Verzeichnis-Objekts werden die zur Replikation verwendeten Attribute vom DSA automatisch generiert. Für das Attribut „objectGUID“ wird eine neue eindeutige GUID generiert. Das Attribut „instanceType“ erhält den Wert INSTANCE_TYPE_WRITE. Außerdem werden die Attribute „whenCreated“ und „whenChanged“ mit dem aktuellen Zeitstempel initialisiert. Die Attribute „uSNCreated“ und „uSNChanged“ werden mit der der Änderung zugeordneten USN initialisiert. Automatisch wird auch ein Attribut basierend auf dem RDN des neuen Objekts, z. B. „dc“ oder „cn“ generiert. Zusätzlich wird das Attribut „name“ mit dem Attribut-Wert des auf dem RDN basierenden Attribut hinzugefügt.

Das Attribut „replPropertyMetaData“ speichert ein Array mit Metadaten für jedes zu replizierende Attribut⁹ (siehe Abbildung 3.14). Dabei wird jeweils die *AttId* anhand des Attribut-Namen gesetzt und das Element *version* wird mit 1 initialisiert. Der aktuelle Zeitstempel wird im Element *originating_change_time* gespeichert. Die lokale DSA-Invocation-Id wird im Element *originating_invocation_id* abgelegt. Die der Änderung zugeordnete USN wird in die Elemente *originating_usn* und *local_usn* geschrieben.

3.5.2 MetadatenSpeicherung beim Originating-Modify

Bei Änderungen an einem Verzeichnis-Objekt werden Attribute generell ersetzt, d. h. beim Löschen eines Attributs wird es durch ein Attribut mit keinem Attribut-Wert gespeichert. Vorher wird allerdings überprüft, ob die Attribut-Werte sich durch die Änderung wirklich ändern. Ist dies nicht der Fall, so wird die Attribut-Änderung ignoriert. Werden alle Attribut-Änderungen ignoriert, so wird die gesamte Änderung ignoriert.

Wird die Änderung nicht ignoriert, so werden die Attribute „whenChanged“ und „uSNchanged“ jeweils auf die aktuelle Zeit bzw. USN gesetzt.

Die Metadaten im Attribut „replPropertyMetaData“ werden für die geänderten bzw. hinzugefügten Attribute angepasst. Das Element *version* wird dabei inkrementiert bzw. bei hinzugefügten Attributen mit 1 initialisiert. Die Elemente *originating_change_time*, *originating_invocation_id*, *originating_usn* und *local_usn* werden analog zum *Originating-Add* gesetzt.

⁹Manche Attribute sind nur lokal für den DSA relevant, außerdem werden einige Attribute automatisch bei einer Anfrage konstruiert

3.5.3 Metadatenstorage beim Originating-Move

Microsoft TechNet behauptet im Abschnitt „Originating Move“ unter [MSTN01], dass ein Verschieben eines Verzeichnis-Objekts innerhalb einer Verzeichnis-Partition wie ein *Originating-Modify* des Attribut „name“ behandelt wird. Dieses Verhalten ist noch nicht durch eigene Experimente verifiziert worden.

3.5.4 Metadatenstorage beim Originating-Delete

Da gelöschte Verzeichnis-Objekte mit Hilfe des *Store-and-Forward-Prinzips* an alle Domänen-Controller propagiert werden müssen, werden Objekte beim *Originating-Delete* nicht wirklich gelöscht, sondern in sogenannte *Tombstones* umgewandelt.

Das Attribut „isDeleted“ wird dabei auf „TRUE“ gesetzt, wodurch eine Suchanfrage das Objekt ignoriert. Zusätzlich werden alle Attribute, welche das Objekt weder eindeutig identifizieren, noch zur Replikation benötigt werden, durch Attribute ohne Attribut-Werte ersetzt. Der RDN-Wert wird durch einen Wert ersetzt, der nicht bei einem *Originating-Add* oder *Originating-Move* verwendet werden kann. Dem RDN-Wert wird der String „\nDEL:“, gefolgt von der String-Repräsentation der Objekt-GUID, angehängen. Ist im Attribut „systemFlags“ nicht das Bit-Flag `SYSTEM_FLAG_DISALLOW_MOVE_ON_DELETE` gesetzt, wird das Objekt in den versteckten Container „CN=Deleted Objects“, unterhalb des Wurzel-Objekts der Verzeichnis-Partition, verschoben. Abbildung 3.20 zeigt ein Beispiel für ein *Tombstone* Objekt.

```
dn: CN=smbtorturedc2\0ADEL:b9cf25c3-ffc3-4dd8-9bbd-5b3776d204c0,CN=Deleted Objects,DC=sub1,DC=w2k3,DC=vmnet1,DC=vm,DC=base
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: user
objectClass: computer
cn:: c21idG9ydHVyZWRjMgpERUw6Yj1jZjI1YzMtZmZjMy00ZGQ4LTliYmQtNWlzMzZDIwNGMw
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:: c21idG9ydHVyZWRjMgpERUw6Yj1jZjI1YzMtZmZjMy00ZGQ4LTliYmQtNWlzMzZDIwNGMw
objectGUID:: wyXPucP/2E2bvVs3dtIEwA==
userAccountControl: 4096
objectSid:: AQUAAAAAAAAUAAAAORX0/enDN5tvFTgaGwUAAA==
sAMAccountName: sbmtorturedc2$
lastKnownParent: CN=Computers,DC=sub1,DC=w2k3,DC=vmnet1,DC=vm,DC=base
```

Abbildung 3.20 Ein Beispiel für ein *Tombstone-Objekt*.

3.6 Replikation von Verzeichnisdaten

Bei Active-Directory werden die Verzeichnis-Partitionen unabhängig voneinander repliziert. D. h. der DSA pflegt für jede Partition ein eigenes „replUpToDateVector“, „repsFrom“ und „repsTo“ Attribut (siehe Abbildung 3.8 und Abbildung 3.10). Das Attribut „repsFrom“ enthält Informationen über alle vom DSA benutzten Source-DSAs, wobei die Attribut-Werte von sogenannten *Knowledge Consistency Checker* (KCC) beim Erstellen der *Replikations-Topologie* automatisch erzeugt werden. Das Attribut „repsFrom“ enthält Informationen über alle Destination-DSAs, die der lokale DSA bei Änderungen benachrichtigen soll. Nähere Informationen darüber sind unter [MSTN01] und [MSTN02] nachzulesen.

3.6.1 Die Funktion DsGetNCChanges ()

Die Funktion DsGetNCChanges () implementiert die *Pull-Replikation*. Stark vereinfacht, erwartet diese Funktion ein struct drsuapi_DsGetNCChangesRequest8 (siehe Abbildung 3.21) als Parameter und liefert ein struct drsuapi_DsGetNCChangesCtr6 (siehe Abbildung 3.22) zurück. Wobei unter Windows-2000 struct drsuapi_DsGetNCChangesRequest5 und struct drsuapi_DsGetNCChangesCtr1 verwendet wurden, die jeweils eine Teilmenge der neueren Versionen darstellen, und deshalb nicht gesondert analysiert werden müssen.

```
/* 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;
```

Abbildung 3.21 Die IDL-Beschreibung der „DsGetNCChangesRequest8“ Struktur.

```

/* 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_lsec 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;

```

Abbildung 3.22 Die IDL-Beschreibung der „DsGetNCChangesCtr6“ Struktur.

Bei der Füllung der „DsGetNCChangesRequest8“ Struktur ist Folgendes zu beachten:

- Das Element `naming_context` muss mit den DN der Verzeichnis-Partition initialisiert werden.
- Um auch die Attribute mit den Passwörtern geliefert zu bekommen, muss das `DRSUAPI_DS_REPLICA_NEIGHBOUR_WRITEABLE` Bit-Flag im Element `replica_flags` gesetzt werden.
- Das Element `max_object_count` gibt an, wie viele Objekte der Source-DSA auf einmal zurückgeben soll. Windows-2003 benutzt hier einen Wert von 133.
- Das Element `max_ndr_size` hat keine wirkliche Bedeutung, hier wird der Wert 1336811 gesendet.
- Das Element `mapping_ctr` sowie alle anderen (bis jetzt) unbekannt Elemente sind mit 0 bzw. NULL initialisiert.
- Der Destination-DSA sollte alle ihm schon bekannten Elemente mit den richtigen Werten oder mit 0 besetzen.

Die Rückgabewerte in der „DsGetNCChangesCtr6“ Struktur liefern folgende Informationen:

- Das Element `source_dsa_guid` enthält die DSA-GUID des Source-DSAs.
- Das Element `source_dsa_invocation_id` enthält die aktuelle DSA-Invocation-GUID des Source-DSAs. Diese muss nicht die vom Destination-DSA erfragte sein.
- Das Element `old_highwatermark` ist die Kopie des vom Destination-DSA gesendeten

High-Water-Mark. Und ist deshalb nicht unbedingt auf den Wert von `source_dsa_invocation_id` bezogen.

- Das Element `new_highwatermark` ist bezogen auf den Wert von `source_dsa_invocation_id`. Die USN in `new_highwatermark.tmp_highest_usn` enthält die höchste USN der Objekte für den aktuellen Aufruf von `DsGetNCChanges()`. Hat die USN in `new_highwatermark.highest_usn` den gleichen Wert, so hat der Source-DSA keine weiteren Änderungen für den Destination-DSA. Ist der Wert von `new_highwatermark.highest_usn` aber kleiner als der von `new_highwatermark.tmp_highest_usn`, so muss der Destination-DSA weitere Änderungen mit einem weiteren Aufruf von `DsGetNCChanges()` erfragen.
- Das Element `uptodateness_vector` wird vom Source-DSA nur zurückgeliefert, wenn keine weiteren Änderungen beim Source-DSA für den Destination-DSA vorliegen (vgl. Element `new_highwatermark`).
- Das Element `mapping_ctr` hat eine große Bedeutung für die Replikation, es enthält die im Verzeichnis-Schema verwendeten *Prefix-Mappings*. Außerdem ist im letzten Array-Element kein *Prefix-Mapping* enthalten, sondern der Wert des Attributs „schemaInfo“, des Wurzel-Objekts der Schema-Partition. Außer bei der Replikation der Schema-Partition, muss der Destination-DSA die *Prefix-Mappings* und das Attribut „schemaInfo“ mit seinen eigenen Werten vergleichen. Stimmen diese nicht überein, so muss der Destination-DSA das komplette Resultat des aktuellen Aufrufes von `DsGetNCChanges()` verwerfen und zuerst die Schema-Partition replizieren.
- Das Element `total_object_count` wird bei Windows-2003 auf „0“ gesetzt. Windows-2000 liefert die Anzahl aller Objekte in der replizierten Verzeichnis-Partition.
- Das Element `object_count` enthält die Anzahl der im aktuellen Aufruf von `DsGetNCChanges` enthaltenen Objekte.
- Das Element `first_object` ist der Kopf einer einfach verketteten Liste von `struct drsuapi_DsReplicaObjectListItemEx` Elementen. Jedes Element enthält ein Verzeichnis-Objekt inklusive der zu replizierenden Attribute und deren Metadaten. Wurde vom Destination-DSA das `DRSUAPI_DS_REPLICA_NEIGHBOUR_RETURN_PARENT_OBJECTS` Bit-Flag im Element `replica_flags` gesetzt, enthält jedes Listen-Element auch die GUID des jeweiligen Vater-Objekts. Wie die einzelnen Listen-Elemente zu interpretieren sind, wird im nächsten Absatz beschrieben.
- Das Element `linked_attributes_count` enthält die Anzahl der replizierten *Linked-Attributes*.
- Das Element `linked_attributes` ist ein Array aus `struct drsuapi_DsReplicaLinkedAttribute` Elementen. Auf *Linked-Attributes* wird in dieser Bachelorarbeit nicht detailliert eingegangen. Wie das Grundprinzip der Replikation von *Linked-Attributes* funktioniert, ist unter [MSTN01] nachzulesen.

Das `DRSUAPI_DS_REPLICA_NEIGHBOUR_COMPRESS_CHANGES` Bit-Flag kann der Destination-DSA im Element `replica_flags` setzen, damit der Source-DSA die komplette „`DsGetNCChangesCtr6`“ Struktur komprimiert. Als Algorithmen werden dabei „MSZIP“ oder ein „XPRESS“ genannter Algorithmus. „MSZIP“ basiert auf dem „Deflate“ Algorithmus [RFC1951]. Der „XPRESS“ genannte Algorithmus ist bislang noch unbekannt. Da Samba die „MSZIP“-Dekomprimierung transparent für den Programmierer erledigt, wird hier nicht auf Details eingegangen. In `source/librpc/ndr/ndr_compression.c` und `source/lib/compression/mszip.c` unter [SaTe01] ist die Implementierung zu finden.

3.6.2 Interpretation der replizierten Verzeichnis-Objekte

Zur Interpretation der replizierten Verzeichnis-Objekte wird das Verzeichnis-Schema

zusammen mit den *Prefix-Mappings* zwingend benötigt. `struct drsuapi_DsReplicaObjectListItemEx *li` stellt jeweils ein zu replizierendes Verzeichnis-Objekt dar (siehe Abbildung 3.21 und Abbildung 3.16). Welche logischen Schritte bei der Interpretation vollzogen werden müssen, wird im Folgenden beschrieben:

- Der Objekt-DN ist über `li->object.identifizier->dn` als String verfügbar.
- Aus dem Objekt-DN wird das auf dem RDN basierende Attribut konstruiert (auch RDN-Attribut genannt).
- Die Objekt-GUID ist über `li->object.identifizier->guid` als `struct GUID` verfügbar, daraus muss das Attribut „objectGUID“ mit der NDR-kodierten GUID konstruiert werden.
- Das Attribut „whenChanged“ wird aus dem höchsten aller `li->meta_data_ctr->meta_data[i].originating_change_time` Werte konstruiert.
- Einige Attribute enthalten sicherheitskritische Passwort-Informationen, z. B. „unicodePwd“, „dBCSPwd“ oder „supplementalCredentials“. Die Attribut-Werte dieser Attribute werden sitzungsspezifisch verschlüsselt. Bevor die Attribut-Werte benutzt werden können, muss die Verschlüsselung rückgängig gemacht werden. Das dabei verwendete Verfahren habe ich durch systematisches Ausprobieren bekannter Algorithmen gefunden. Hierbei kommen folgende Algorithmen zum Einsatz:
 - *Message Digest* (MD5) [RFC1321], eine kryptografische *Hash-Funktion* (siehe auch `source/lib/crypto/md5.c` unter [SaTe01]).
 - *Cyclic Redundancy Check 32* (CRC32) [RFC1952] (siehe Absatz „8. Appendix: Sample CRC Code“), eine *Prüfsummen-Funktion*. (siehe auch `source/lib/crypto/crc32.c` unter [SaTe01]).
 - ARCFOUR¹⁰[WikiPe03],[GoGr01], eine *Stromverschlüsselungs-Funktion*. (siehe auch `source/lib/crypto/arcfour.c` unter [SaTe01]).

Bei der Kerberos- oder NTLMSSP-Authentifizierung, auf DCERPC-Ebene, wird ein Sitzungsschlüssel ausgehandelt, der in die Verschlüsselung einfließt. Die sitzungsspezifische Verschlüsselung wird wie folgt vollzogen:

- Es wird eine 16-Byte-Zufallszahl generiert, diese wird als *Confounder* bezeichnet.
- Aus dem Sitzungsschlüssel gefolgt vom *Confounder* wird mit Hilfe der MD5-Hash-Funktion ein Verschlüsselungsschlüssel generiert.
- Die Passwortdaten, mit vorangestellter CRC32-Prüfsumme, werden unter Verwendung des Verschlüsselungsschlüssels mit der ARCFOUR-Funktion verschlüsselt.
- Zur Übertragung wird den verschlüsselten Daten der *Confounder* vorangestellt.

Abbildung 3.23 verdeutlicht dieses Verfahren. (siehe `dsdb_decrypt_attribute_value()` in `source/dsdb/repl/replicated_objects.c` unter [SaTe01] bzw. Anhang A12 ab Zeile 33).

- Danach werden alle Attribute `li->object.attribute_ctr.attributes[i]` in einer Schleife konvertiert. Dabei wird die 32-Bit *AttId* mit Hilfe des *Prefix-Mappings* und dem geladenen Schema-Cache in den Attribut-Namen umgewandelt. Anhand der Attribut-Definition im Schema-Cache wird dann die sogenannte *Attribut-Syntax* ermittelt, z. B. „Boolean“, „Integer“, „String(UNICODE)“ oder „Object(DS-DN)“. Jeder *Attribut-Syntax* sind Konvertierungs-Funktionen zugeordnet, welche die übertragenen Attribut-Werte in die intern verwendete Repräsentation umwandeln. Samba speichert Attribut-Werte, wie sie im LDAP-Protokoll verwendet werden. Integer-Werte werden z. B. in der String-Repräsentation gespeichert. Alle möglichen *Attribut-Syntaxen* und ihre jeweiligen Konvertierungs-Funktionen sind unter `source/dsdb/schema/schema_syntax.c` unter [SaTe01] bzw. im Anhang A11 implementiert. Ab Zeile 1113 sind die *Attribut-Syntaxen* als Array aus

¹⁰Der Name „RC4“ ist ein Warenzeichen von RSA Data Security, Inc. Deshalb wird meist der Name „ARCFOUR“ benutzt.

struct dsdb_syntax (siehe source/dsdb/schema/schema.h unter [SaTe01] bzw. Anhang A4 ab Zeile 30) Elementen definiert.

- Das Metadaten-Array muss aus den struct drsuapi_DsReplicaMetaData Elementen in ein Array aus struct replPropertyMetaDatum Elementen umgewandelt werden. Dabei wird ein weiteres struct replPropertyMetaDatum Element für das RDN-Attribut angehängt, die Metadaten werden dabei vom Attribut „name“ kopiert. Die Metadaten werden dann im Attribut „replPropertyMetaDatum“ gespeichert. Siehe auch Abbildung 3.14.

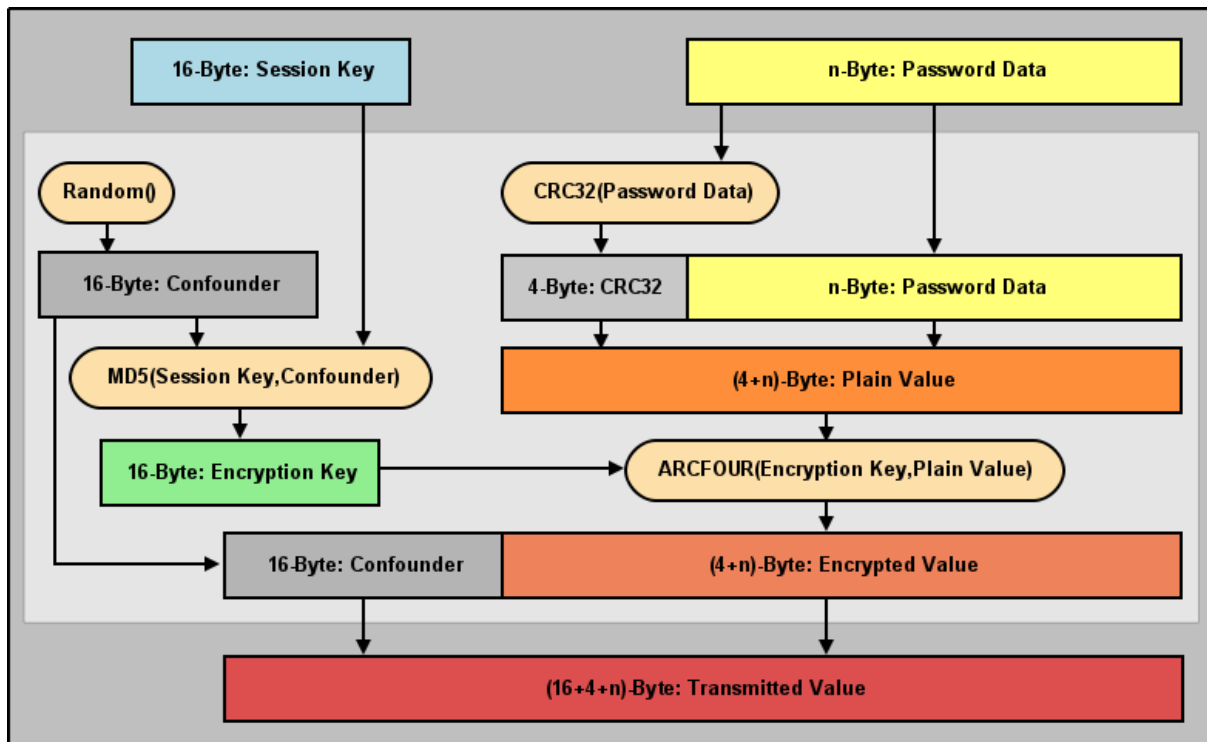


Abbildung 3.23 Die sitzungsspezifische Verschlüsselung der Passwort-Informationen.

3.6.3 Methoden zum Filtern der zu replizierenden Daten

Der Source-DSA filtert die zum Destination-DSA gesendeten Daten. Dadurch werden nur die Änderungen übertragen, die der Destination-DSA nicht schon von einem anderen Source-DSA repliziert hat.

Filtern anhand des High-Water-Mark

Bei einer Replikations-Anforderung muss der Source-DSA anhand des vom Destination-DSA geschickten *High-Water-Mark* entscheiden, welche Verzeichnis-Objekte er an den Destination-DSA schicken muss. Dabei listet er einfach alle Objekte auf, deren „uSNChanged“ Wert höher ist als der *highest_usn* Wert im *High-Water-Mark*. Dabei werden maximal 133 Objekte, sortiert in aufsteigender Reihenfolge der „uSNChanged“ Werte, zum Verschicken an den Destination-DSA ausgewählt. Der Einsatz des *High-Water-Mark* sorgt dafür, dass Änderungen nur einmal zwischen einem Paar aus Source-DSA und Destination-DSA übertragen werden.

Filtern anhand des Up-To-Date-Vectors

Bevor der Source-DSA die mittels *High-Water-Mark* erstellte Objekt-Liste an den Destination-DSA schickt, wird überprüft, ob die zu replizierenden Daten noch weiter minimiert werden können. Der vom Destination-DSA geschickte *Up-To-Date-Vector* enthält eine Liste aller dem Destination-DSA bekannten DSA-Invocation-Ids mit der jeweils dazugehörigen höchsten bekannten USN. Deshalb wird für jedes Attribut anhand seiner Metadaten (*originating_invocation_id* und *originating_usn*) sowie dem *Up-To-Date-Vectors* überprüft, ob das Attribut auf dem Destination-DSA schon vorhanden ist. Ergibt diese Prüfung, dass das Attribut nicht übertragen werden muss, wird es vom Objekt entfernt. Bleiben bei einem Objekt keine zu übertragenden Attribute übrig, so wird das gesamte Objekt aus der Liste entfernt. Eventuell rückt dann ein weiteres Objekt nach.

3.6.4 Algorithmus zur Konfliktauflösung bei Multimaster-Replikation

Beim Einspielen von *Replicating-Updates* können bei der *Multimaster-Replikation* generell Konflikte auftreten. Diese werden bei Active-Directory durch einen Algorithmus gelöst, der garantiert auf jedem DSA das selbe Ergebnis liefert. Dabei lautet das Motto: „Der letzte Schreibzugriff gewinnt“.

Konflikte auf Attribut-Ebene

Bevor ein Attribut samt aller seiner Attribut-Werte durch ein *Replicating-Update* ersetzt wird, werden die alten und neuen Metadaten (`struct replPropertyMetaData1`) verglichen. Zuerst wird das *version* Element verglichen. Bei Gleichheit wird dann das *originating_change_time* Element verglichen. Im statistisch fast ausgeschlossenen Fall, dass das selbe Attribut innerhalb der selben Sekunde auf zwei DSAs gleichzeitig geändert wurde, wird das *originating_invocation_id* Element verglichen, was den Konflikt immer eindeutig auflöst. Hat das replizierte Attribut die „höheren“ Metadaten, wird das alte Attribut ersetzt, andernfalls wird das replizierte Attribut verworfen.

Konflikte auf DN-Ebene

Durch den Einsatz der *Multimaster-Replikation* kann es auch vorkommen, dass Objekte erzeugt werden, deren Vater-Objekt auf einem anderen DSA gelöscht wurde. Laut [MSTN01] (Absatz „Replication Conflict Resolution“) werden solche Objekte in den „CN=Lost And Found“ Container unterhalb des Wurzel-Objekts der Verzeichnis-Partition verschoben. Dieser Aspekt wurde allerdings vom Samba-Team noch nicht detailliert untersucht.

Durch den Einsatz der *Multimaster-Replikation* kann es auch vorkommen, dass Objekte mit dem gleichen Objekt-DN auf zwei verschiedenen DSAs erzeugt werden. Laut [MSTN01] (Absatz „Replication Conflict Resolution“) werden die RDNs (effektiv das Attribut „name“), der beiden Objekte dem für Attribut-Konflikte geltenden, Vergleichs-Algorithmus unterzogen, wobei das Objekt mit den „höheren“ Metadaten seinen Objekt-DN behalten darf. Bei dem Objekt mit den „niedrigeren“ Metadaten wird dem RDN der String „*CNF:“ gefolgt von der String-Repräsentation der Objekt-GUID angehängen. Als Beispiel würde „CN=NewUser, CN=Users, ...“ zu „CN=NewUser*CNF:077157f1-eccc-47bf-ba03-9753c93e2e73, CN=Users, ...“. Dieser Aspekt wurde allerdings vom Samba-Team noch nicht detailliert untersucht.

3.6.5 Change-Notify via DsReplicaSync()

Es ist wünschenswert, Änderungen mit möglichst geringer Latenzzeit an alle DSAs zu propagieren. Bei Active-Directory wird mit Hilfe der Funktion `DsReplicaSync()` ein Mechanismus, der *Change-Notify* genannt wird, implementiert. Bei jedem *Originating-Update* setzt der DSA einen Timer von standardmässig 15 Sekunden. Danach werden alle, dem DSA bekannten, Destination-DSAs in Abständen von 3 Sekunden über die Änderung benachrichtigt. Die dabei verwendete Funktion `DsReplicaSync()` erwartet ein `struct drsuapi_DsReplicaSyncRequest1` als Parameter (siehe Abbildung 3.24). Der Source-DSA übergibt den DN der Verzeichnis-Partition und seine DSA-GUID, damit der Destination-DSA weiß, bei welchem Source-DSA er welche Verzeichnis-Partition replizieren soll. Standardmässig wird das Element `other_info` mit NULL und das Element `options` mit 0 initialisiert.

```

/* 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 repsFromTo10therInfo dns_name */
    drsuapi_DsReplicaSyncOptions options;
} drsuapi_DsReplicaSyncRequest1;

```

Abbildung 3.24 Die IDL-Beschreibung der „DsReplicaSyncRequest1“ Struktur.

3.6.6 Periodische Pull-Replikation

Zusätzlich zum *Change-Notify* führt jeder DSA standardmässig alle 15 Minuten eine *Pull-Replikation* mit jedem bekannten Source-DSA aus.

4 Implementierung von Teilaspekten basierend auf Samba 4.0

In diesem Kapitel sollen die logischen Schritte und Zusammenhänge der zu implementierenden Teilaspekte grob beschrieben werden. Die Erläuterung von Details würde wegen der ca. 10 Tausend Quelltextzeilen den Rahmen dieser Bachelorarbeit bei weitem übersteigen. Deshalb wird jeweils direkt auf die Funktionen im Quelltext (siehe Anhang) verwiesen.

4.1 Implementierungskonzepte in Samba 4.0

4.1.1 Programmiersprachen

Die Programmiersprache „C“

Der Hauptteil von Samba ist in der Programmiersprache „C“ geschrieben. Dazu gehören eine Reihe von Client-Bibliotheken unter anderem für alle in Active-Directory relevanten Protokolle. Außerdem das Server-Programm `smbd` und viele kleine Tools, die diverse Aufgaben erfüllen. Einen großen Teil nimmt das Test-Programm `smbtorture` ein, damit werden Protokoll-Aspekte sowie interne Schnittstellen getestet.

Die Programmiersprache „EJS“

Ein weiterer Teil von Samba ist in der Sprache „JavaScript“ programmiert, dies scheint auf den ersten Blick eher untypisch, da diese Sprache meist nur im Umfeld von clientseitigen Web-Applikationen bekannt ist. Der Grund für die Nutzung innerhalb von Samba ist der eingebaute Web-Server, der ein Administrations-Interface bietet. Hierbei wird auf der Server-Seite die Variante „Embedded-Javascript“ (EJS) und auf der Client-Seite Standard „JavaScript“ verwendet. Dafür wurde der EJS-Interpreter aus dem Appweb-Projekt (siehe [AppWeb01]) integriert.

Eine Zwischenschicht erlaubt den Aufruf von C-Funktionen aus EJS-Programmen. Außerdem bietet das Programm `smbscript` die Möglichkeit für Kommandozeilen-Skripte, ähnlich wie Perl oder Python. Umgekehrt ist aber auch der Aufruf von EJS-Programmen aus C-Quelltext möglich.

Die Beschreibungssprache „IDL“

Die Definition der DCERPC-Schnittstellen ist in Sprache „IDL“ realisiert. Sambas IDL-Compiler „`pidl`“ erzeugt daraus C-Funktionen, welche die NDR-Kodierung und -Dekodierung der C-Strukturen vornehmen. Außerdem werden DCERPC-Client-Funktionen für „C“ und „EJS“ generiert. Durch das automatische Generieren des C-Quelltextes wird sehr viel fehleranfällige Handarbeit eingespart (momentan ca. 280 Tausend Zeilen).

4.1.2 Betriebssysteme

Samba ist auf einer Vielzahl von Unix-Betriebssystemen einsatzfähig. Dazu gehören AIX, FreeBSD, HP-UX, IRIX, Linux, MacOS, NetBSD, OpenBSD, Solaris und Tru64.

Wichtig ist, dass das Betriebssystem die Funktion `select()` (siehe [Man01]) oder `epoll()` (siehe [Man02]) unterstützt. Damit ist es möglich, mit einer Funktion eine Vielzahl von Datei- und Socket-Deskriptoren auf Ereignisse zu überwachen. Diese Funktion ist dann die einzige, die den Unix-Prozess „Schlafen“ legen kann. Alle I/O-Operationen werden daraufhin immer nicht blockierend ausgeführt. Dies wird *Non-Blocking-I/O* genannt.

4.1.3 TALLOC der „Tree-Allocator“

Samba benutzt nicht die Funktionen `malloc()`, `strdup()` und `free()`. Samba verwendet ein hierarchisches Speichermanagement-System namens „TALLOC“ (siehe `source/lib/talloc/talloc_guide.txt` unter [SaTe01]). Es unterstützt Speicher-Referenzen und Destruktoren. Außerdem wird Typsicherheit geboten. Abbildung 4.1 zeigt eine beispielhafte Nutzung.

```
#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;
}
```

Abbildung 4.1 Ein beispielhafte Nutzung von „TALLOC“.

4.1.4 Das Subsystem „EVENTS“

Samba ist grundsätzlich für ein asynchrones Arbeiten und *Non-Blocking-I/O* ausgelegt worden. Dabei wurde die Abstraktionsschicht *EVENTS* (siehe `source/lib/events/events.h` unter [SaTe01]) entwickelt, welche die Betriebssystem-spezifischen Details versteckt.

Grundsätzlich können folgende Ereignisse überwacht werden:

- `fd_event`: Datei- und Socket-Deskriptoren, die lesbar oder schreibbar werden.
- `timed_event`: Zeitpunkte, die verstreichen.
- `aio_event`: asynchrone Dateisystem-Zugriffe (unter Linux).
- `signal_event`: Das Auftreten von Unix-Signalen (der Gebrauch sollte möglichst vermieden werden).

Abbildung 4.2 zeigt eine beispielhafte Nutzung.

```
#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;
}
```

Abbildung 4.2 Ein beispielhafte Nutzung der Funktionen des Subsystem „EVENTS“.

4.1.5 Asynchrone Funktionsaufrufe

Im allgemeinen werden in Samba alle Funktionen, die über *Sockets* oder *Pipes* Kommunikation mit anderen Prozessen betreiben, asynchron programmiert. Dabei gibt es immer eine C-Funktion (z. B. `foo_send()`), um die Funktions-Parameter zu „senden“ und eine C-Funktion (z. B. `foo_recv()`), um die Rückgabewerte zu „empfangen“. Dabei erwartet die Sende-Funktion unter anderem ein `struct event_context` als Parameter und gibt ein *Request-Handle* zurück. Will die aufrufende Funktion aktiv auf das Eintreffen der Rückgabewerte warten, so ruft sie sofort die Empfangs-Funktion auf und übergibt ihr das *Request-Handle*. Dies wird häufig, zur vereinfachten Nutzung in Testprogrammen, durch eine synchrone C-Funktion (z. B. `foo()`) realisiert. Will die aufrufende Funktion nicht aktiv auf das Eintreffen der Rückgabewerte warten, so kann sie im *Request-Handle* eine *Callback-Funktion* hinterlegen und sich ihrerseits beenden. Das Hauptprogramm fährt dann typischerweise innerhalb der Funktion `event_loop_wait()` mit der Abarbeitung anderer Ereignisse fort oder legt den Prozess bis zum Eintreffen eines neuen Ereignisses „Schlafen“. Stehen nun irgendwann die Rückgabewerte der Funktion bereit, wird die im *Request-Handle* hinterlegte *Callback-Funktion* aufgerufen. Die *Callback-Funktion* muss dann ihrerseits die Empfangs-Funktion aufrufen, um an die Rückgabewerte zu gelangen. Durch den Aufruf der Empfangs-Funktion wird das *Request-Handle* wieder freigegeben. Abbildung 4.3 zeigt ein Beispiel.


```

#include <stdio.h>
#include <stdint.h>
#include <sys/time.h>
#include <malloc.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_rcv(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_rcv(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;
}

```

Abbildung 4.3 Ein Beispiel einer asynchron aufgerufenen Funktion.

4.1.6 TDB, die „Triviale Datenbank“

Die in Samba verwendete *Trivial Database* (TDB) bietet eine sehr effiziente, auf *Shared-Memory* basierende, *Hash-Tabelle* zum Speichern von *Schlüssel-Wert-Paaren*. TDB unterstützt *Multitasking* und *Transaktionen* (siehe `source/lib/tdb/docs/README` unter [SaTe01]).

4.1.7 LDB, die „Leichtgewichtige Datenbank“

Die im Samba verwendete *Lightweight Database* (LDB) [SaTe05] bietet folgende Funktionen:

- Die Programmierschnittstelle (LDB-API) und das Datenmodell sind sehr ähnlich zu LDAP.
- Die Daten werden entweder in einer TDB-Datei oder in einem wirklichen LDAP-Server gespeichert. Außerdem existiert ein experimentelles „sqlite3“ LDB-Backend.
- Beim Einsatz einer TDB-Datei wird kein gesonderter Server-Prozess benötigt. Hier entfällt das meist komplizierte Aufsetzen eines LDAP-Servers. Außerdem werden die Zugriffszeiten dadurch optimiert.
- Die LDB-API kann auch als Alternative zur herkömmlichen LDAP-API verwendet werden.
- Standardmässig wird kein Schema verwendet.
- Sehr einfache Administration von indizierten Attributen.
- Durch LDB-Module können zusätzliche Funktionen implementiert werden. Z.B. Unterstützung für ein Verzeichnis-Schema.
- Es existieren leicht zu bedienende Administrations-Programme.
- Ein Importieren und Exportieren im LDIF-Format ist möglich.
- Es werden LDB-Controls und LDB-Extended-Operations analog zu *LDAP-Controls* und *LDAP-Extended-Operations* unterstützt (siehe auch [RFC4510]).

4.2 Das Subsystem „DSDB“ (oder „SAMDB“)

In Samba werden die C-Funktionen, die zur Implementierung des Verzeichnis-Dienstes verwendet werden, im Subsystem *Directory Service Database* (DSDB) zusammengefasst. Teilweise wird der Begriff *Security Account Manager Database* (SAMDB) synonym verwendet. Nach und nach soll DSDB aber den Vorzug erhalten.

Bei der Implementierung eines Verzeichnis-Dienstes übernehmen mehrere aufeinander aufbauende Schichten die Aufgabe, die Verzeichnis-Objekte auf einem Sekundärspeichermedium abzuspeichern.

4.2.1 DSA-Operational-Layer

Der *DSA-Operational-Layer* kümmert sich um die Semantik der vom Verzeichnis-Dienste zur Verfügung gestellten Funktionen „Add“, „Modify“, „Delete“ und „Search“. D. h. Schreibzugriffe werden vom Verzeichnis-Schema validiert und die zur Replikation benötigten Metadaten gepflegt. Außerdem werden Berechtigungen bei Lese- und Schreibzugriffen überprüft. Der *DSA-Operational-Layer* ist unabhängig vom zum Zugriff verwendeten Netzwerkprotokoll. Die Server-Komponenten der jeweiligen Netzwerkprotokolle, z.B. LDAP, DRSUAPI, SAMR und NETLOGON, bauen auf dem *DSA-Operational-Layer* auf. Abbildung 4.4 zeigt eine Übersicht der verwendeten Schichten.

Laut [MSTN01] (im Abschnitt „Replication Subsystem“) ist der *DSA-Operational-Layer* unter Windows-2003 in der Datei `ntdsa.dll` implementiert. Samba benutzt hier eine Reihe von LDB-Modulen (siehe Abbildung 4.4).

4.2.2 Database-Layer

Der *Database-Layer* kümmert sich darum, wie die Verzeichnis-Objekte gespeichert werden. Außerdem werden in dieser Schicht die Attribut-Indizes verwaltet.

Laut [MSTN01] (im Abschnitt „Replication Subsystem“) ist der *Database-Layer* unter Windows-2003 in der Datei `ntdsa.dll` implementiert. Im Samba werden die Aufgaben des *Database-Layer* im LDB-BACKEND „ldb_tdb“ erledigt (siehe Abbildung 4.4).

4.2.3 Storage-Layer

Der *Storage-Layer* kümmert sich darum, wie die Daten letztlich auf dem Sekundärspeichermedium verwaltet werden. Hierzu gehört auch die Implementierung von Transaktionen.

Laut [MSTN01] (im Abschnitt „Replication Subsystem“) ist der *Storage-Layer* unter Windows-2003 in der Datei `Esent.dll` implementiert. Die Daten werden im Dateisystem in der Datei „Ntds.dit“ gespeichert. Im Samba werden die Aufgaben des *Storage-Layer* von TDB erledigt. Im Dateisystem werden die Daten in den Dateien `samdb.ldb`, `domain.ldb`, `config.ldb` und `schema.ldb` abgespeichert (siehe Abbildung 4.4).

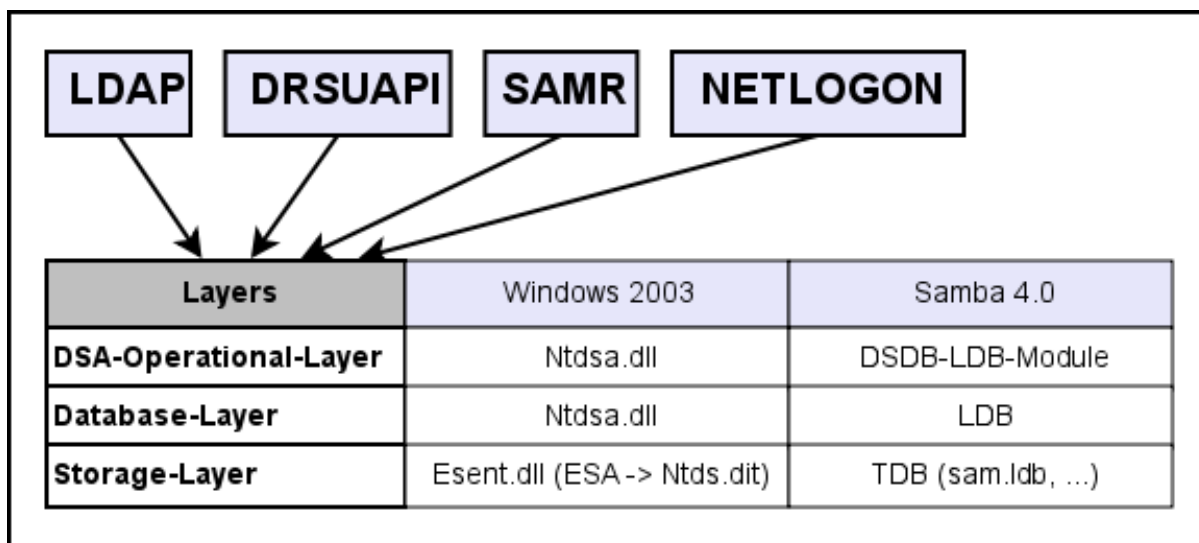


Abbildung 4.4 Schichtenvergleich zwischen Windows-2003 und Samba 4.0 (vgl. „Replication Subsystem Components“ in [MSTN01]).

4.2.4 DSDB LDB-Module

In Samba stellt die LDB-API die protokollunabhängige Schnittstelle des *DSA-Operational-Layer* dar. Die Aufgaben des *DSA-Operational-Layer* übernehmen LDB-Module. In der folgenden Abbildung 4.5 wird das Zusammenspiel der LDB-Module gezeigt. Danach werden die jeweiligen Aufgaben der LDB-Module kurz beschrieben.

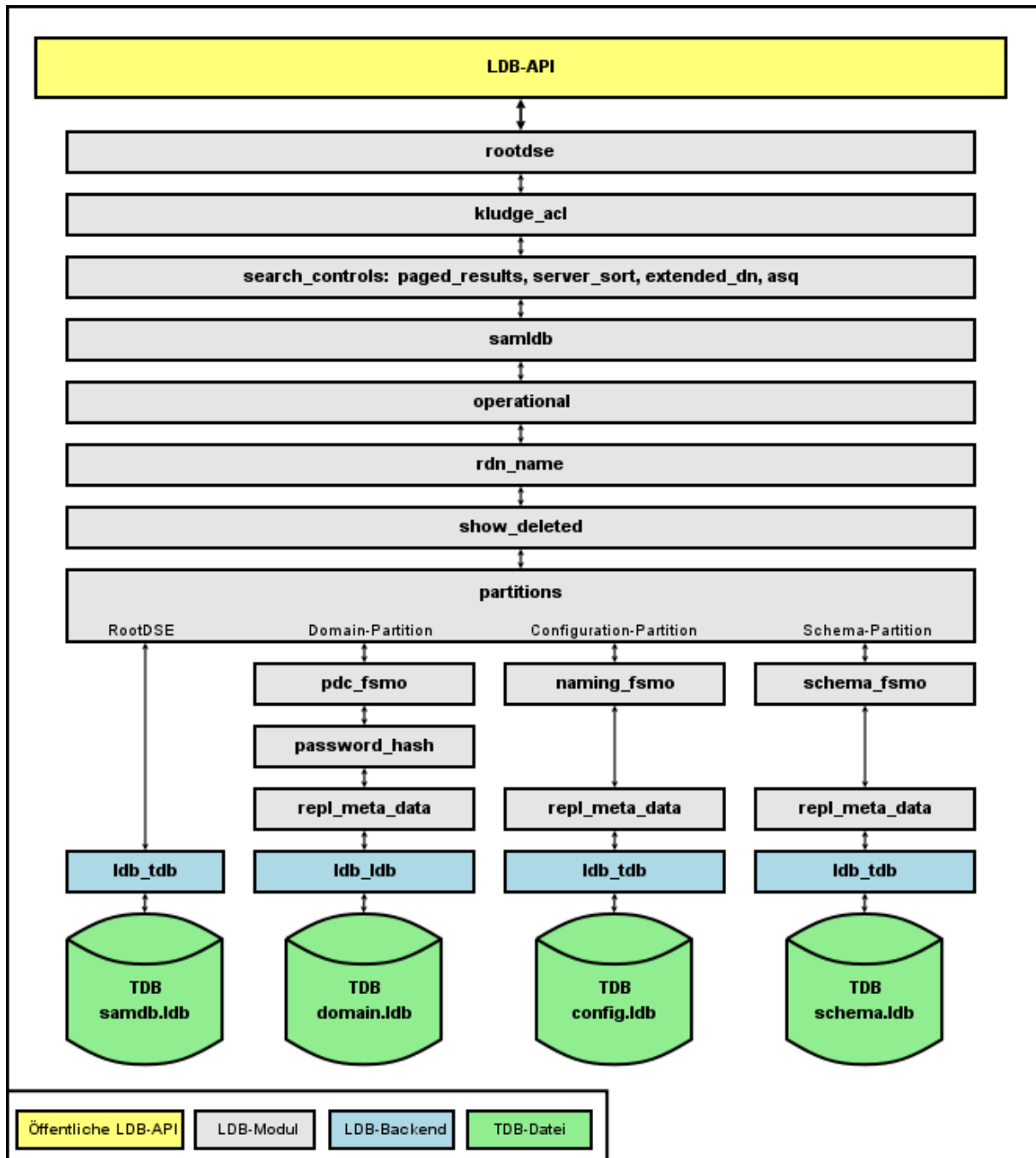


Abbildung 4.5 Übersicht der LDB-Module, die im Subsystem „DSDB“ verwendet werden.

LDB-Modul „rootse“

Das LDB-Modul „rootse“ implementiert die Eigenheiten des RootDSE-Objekts (siehe source/dsdb/samdb/ldb_modules/rootse.c unter [SaTe01]). Z. B. liefern die Attribute „highestCommittedUsn“ und „currentTime“ bei jeder Suchanfrage aktuelle Werte.

LDB-Modul „kludge_acl“

Das LDB-Modul „kludge_acl“ bietet eine triviale Implementierung zur Durchsetzung von

Berechtigungen (siehe `source/dsdb/samdb/ldb_modules/kludge_acl.c` unter [SaTe01]). Administratoren dürfen dabei alle Objekte und Attribute lesen und schreiben. Alle anderen Benutzer dürfen alle Objekte und fast alle Attribute lesen, nur die Attribute mit Passwort-Informationen werden zurückgehalten. Später soll dieses LDB-Modul durch ein Modul mit weit größerem Funktionsumfang ersetzt werden.

LDB-Module „paged_results“, „server_sort“, „extended_dn“ und „asq“

Die LDB-Module „paged_results“, „server_sort“, „extended_dn“ und „asq“ implementieren jeweils ein *LDAP-Control* (siehe [RFC4510]). Da sie für die Replikation keine große Bedeutung haben, wird hier nicht weiter darauf eingegangen (siehe `source/dsdb/samdb/ldb_modules/` und `lib/ldb/modules/` unter [SaTe01]).

LDB-Modul „samldb“

Das LDB-Modul „samldb“ sorgt am *Originating-Add* dafür, dass Benutzer- und Gruppen-Objekte eine SID zugeordnet bekommen, außerdem werden einige Attribute mit Standard-Werten initialisiert (siehe `source/dsdb/samdb/ldb_modules/samldb.c` unter [SaTe01]).

LDB-Modul „operational“

Das LDB-Modul „operational“ konstruiert bei Suchanfragen konstruierte Attribute, wie z. B. „createTimestamp“, „modifyTimestamp“, „structuralObjectClass“ und „canonicalName“ (siehe `source/lib/ldb/modules/operational.c` unter [SaTe01]). Später muss dieses LDB-Modul durch Benutzung der Schema-API erweitert werden.

LDB-Modul „rdn_name“

Das LDB-Modul „rdn_name“ sorgt dafür, dass das RDN-Attribut und das Attribut „name“ immer den selben Wert haben (siehe `source/lib/ldb/modules/rdn_name.c` unter [SaTe01]).

LDB-Modul „show_deleted“

Das LDB-Modul „show_deleted“ sorgt dafür, dass Objekte, bei denen das Attribut „isDeleted“ den Wert „TRUE“ besitzt, bei einer Suchanfrage ignoriert werden. Das LDAP-Control „SHOW_DELETED“ (siehe [MSDN01]) ist ebenfalls implementiert und sorgt dafür, dass die *Tombstones* nicht ignoriert werden.

Dieses LDB-Modul wurde im Rahmen dieser Bachelorarbeit programmiert (siehe `source/dsdb/samdb/ldb_modules/show_deleted.c` unter [SaTe01] bzw. Anhang A15).

LDB-Modul „partition“

Das LDB-Modul „partition“ sorgt dafür, dass Anfragen an die richtige Verzeichnis-Partition weitergeleitet werden (siehe `source/dsdb/samdb/ldb_modules/partition.c` unter [SaTe01]). Dabei wird der Anfrage das LDB-Control `DSDB_CONTROL_CURRENT_PARTITION_OID` samt `struct dsdb_control_current_partition` angehängen, um den Partitions-spezifischen LDB-Modulen Informationen zur aktuellen Verzeichnis-Partition zu übermitteln (siehe `source/dsdb/samdb/samdb.h` unter [SaTe01]).

LDB-Module „pdc_fsmo“ und „naming_fsmo“

Die LDB-Module „pdc_fsmo“ und „naming_fsmo“ machen momentan noch nichts (siehe `source/dsdb/samdb/ldb_modules/` unter [SaTe01]). Später sollen die *PDC-Emulator-Master* bzw. *Domain-Naming-Master* Funktionalitäten implementiert werden (siehe [MSTN03]).

LDB-Modul „password_hash“

Das LDB-Modul „password_hash“ sorgt dafür, dass bei Passwortänderungen die Passwort-Hash-Attribute „unicodePwd“, „dBCSPwd“ und „supplementalCredentials“ generiert werden und standardmässig das Klartextpasswort nicht gespeichert wird. Außerdem werden die Attribute „ntPwdHistory“, „lmPwdHistory“ und „pwdLastSet“ gepflegt (siehe source/dsdb/samdb/ldb_modules/password_hash.c unter [SaTe01]).

LDB-Modul „schema_fsmo“

Das LDB-Modul „schema_fsmo“ sorgt für das Laden des Verzeichnis-Schemas mit Hilfe der Schema-API. Später soll es auch die *Schema-Master* Funktionalität implementieren (siehe [MSTN03]).

Dieses LDB-Modul wurde im Rahmen dieser Bachelorarbeit programmiert (siehe source/dsdb/samdb/ldb_modules/schema_fsmo.c unter [SaTe01] bzw. Anhang A14).

LDB-Modul „repl_meta_data“

Das LDB-Modul „repl_meta_data“ sorgt dafür, dass bei allen *Originating-Updates* die zur Replikation benötigten Metadaten gepflegt werden. Momentan ist dies nur beispielhaft für den Fall des *Originating-Add* implementiert, später müssen auch alle anderen Fälle berücksichtigt werden. Außerdem stellt das „repl_meta_data“ Modul die LDB-Extended-Operation `DSDB_EXTENDED_REPLICATED_OBJECTS_OID` bereit, über die *Replicating-Updates* angewendet werden (siehe source/dsdb/samdb/samdb.h unter [SaTe01]).

Dieses LDB-Modul wurde im Rahmen dieser Bachelorarbeit programmiert (siehe source/dsdb/samdb/ldb_modules/repl_meta_data.c unter [SaTe01] bzw. Anhang A13).

4.3 Änderung der Server-Rolle in die Domänen-Controller-Rolle

Im Rahmen dieser Bachelorarbeit wurde die Funktion `libnet_BecomeDC()` implementiert. Hier wird grob die Funktionalität dieser Funktion beschrieben. Dabei wird jeweils auf Stellen im Quelltext (im Anhang) verwiesen, wo dann Details nachzulesen sind. Siehe `source/libnet/libnet_become_dc.h` und `source/libnet/libnet_become_dc.c` unter [SaTe01] bzw. Anhang A5 und Anhang A8.

Der Vollständigkeit halber wurde analog zu `libnet_BecomeDC()` die Funktion `libnet_UnbecomeDC()` entwickelt, die einen Domänen-Controller wieder zum Domänen-Mitglied herunterstuft. Hier wird allerdings nur auf den Quelltext verwiesen. Siehe `source/libnet/libnet_unbecome_dc.h` und `source/libnet/libnet_unbecome_dc.c` unter [SaTe01] bzw. Anhang A6 und Anhang A9.

4.3.1 Die Vorbedingungen für `libnet_BecomeDC()`

Als Vorbedingung wird von `libnet_BecomeDC()` erwartet, dass ein Computer-Konto für ein Domänen-Mitglied schon im Active-Directory vorhanden ist.

4.3.2 Die Eingabe-Parameter von `libnet_BecomeDC()`

Die Eingabe-Parameter werden in `struct libnet_BecomeDC` zusammengefasst:

- Der DNS-Name der Domäne ist im Element `in.domain_dns_name` zu übergeben. Intern wird teilweise `realm` (meist in Großbuchstaben) synonym zu `domain_dns_name` (meist in Kleinbuchstaben) verwendet.
- Der NETBIOS-Name der Domäne ist im Element `in.domain_netbios_name` zu übergeben.
- Die Domänen-SID wird im Element `in.domain_sid` übergeben.
- Die IP-Adresse wird als String im Element `in.source_dsa_address` übergeben.
- Der NETBIOS-Name des Computer-Kontos, welches zum Domänen-Controller heraufgestuft werden soll, wird im Element `in.dest_dsa_netbios_name` übergeben. Intern wird teilweise `hostname` (meist in Kleinbuchstaben) synonym zu `netbios_name` (meist in Großbuchstaben) verwendet. Der DNS-Name wird aus `hostname` und `domain_dns_name` konstruiert.
- Die Funktion `libnet_BecomeDC()` ist so ausgelegt, dass nach jedem logischen Schritt schon ermittelte Ergebnisse mittels *Callback-Funktionen* an die aufrufende Funktion übergeben werden (siehe `struct libnet_BecomeDC_Callbacks`). Dabei ist es auch möglich, dass die *Callback-Funktion* dafür sorgt, dass die Funktion `libnet_BecomeDC()` die Ausführung abbricht und der aufrufenden Funktion einen Fehler meldet. Die einzelnen *Callback-Funktionen* werden später erläutert.

4.3.3 Die logischen Schritte von `libnet_BecomeDC()`

Die Funktion `libnet_BecomeDC` erledigt eine komplexe Folge von Netzwerk-Anfragen (LDAP und DRSUAPI). Dabei werden die einzelnen asynchronen Anfragen zu einer einzigen asynchronen Funktion zusammengefasst, die Komplexität bleibt dadurch dem Aufrufer verborgen. In Samba stellt das Subsystem *COMPOSITE* Helferfunktionen zum Verketteten von asynchronen Funktionen zur Verfügung (siehe `source/libcli/composite/composite.c` unter [SaTe01]). Eine detaillierte Beschreibung der logischen Schritte und Teilaufgaben ist als Kommentar in `source/libnet/libnet_become_dc.c` unter [SaTe01] bzw. Anhang A8 ab Zeile 37

verfügbar. Grob werden die folgenden Teilaufgaben abgearbeitet:

- Zuerst werden eine Reihe von Informationen über die Domäne, den Domänen-Wald und den Source-DSA eingeholt. Danach werden diese Informationen als `struct libnet_BecomeDC_CheckOptions` an die *Callback-Funktion* `check_options()` übergeben.
- Danach wird die Konfiguration des neuen Domänen-Controllers erstellt. Unter anderem wird das „NTDS Settings“ mit Hilfe der Funktion `DsAddEntry` (siehe auch `drsuapi.idl` unter [SaTe03] bzw. Anhang A2 ab Zeile 1018) erstellt. Danach werden die Informationen über die Domäne, dem Domänen-Wald, den Source-DSA und den Destination-DSA als `struct libnet_BecomeDC_PrepareDB` an die *Callback-Funktion* `prepare_db()` übergeben. Hiermit hat die aufrufende Funktion (bzw. deren *Callback-Funktion*) genügend Informationen um die zum Speichern der Verzeichnis-Daten verwendeten Dateien zu erzeugen.
- Im nächsten Schritt wird die Schema-Partition repliziert. Dabei wird die Funktion `DsGetNCChanges()` in einer Schleife solange aufgerufen, bis die Partition komplett übertragen wurde. Nach jedem erfolgreichen `DsGetNCChanges()` Aufruf wird das Ergebnis (`struct drsuapi_DsGetNCChangesCtrl` oder `struct drsuapi_DsGetNCChangesCtrl6`) zusammen mit dem zur Verschlüsselung verwendeten Sitzungsschlüssel und den Informationen über die Domäne, dem Domänen-Wald, den Source-DSA, den Destination-DSA und die Verzeichnis-Partition als `struct libnet_BecomeDC_StoreChunk` an die *Callback-Funktion* `schema_chunk()` übergeben. Die aufrufende Funktion (bzw. deren *Callback-Funktion*) kann die jeweils replizierten Verzeichnis-Objekte (typischerweise in Blöcken von 133 Objekten) verarbeiten und speichern.
- Danach wird die Konfigurations-Partition repliziert. Dabei wird wie bei der Schema-Partition verfahren. Hier wird allerdings die *Callback-Funktion* `config_chunk()` verwendet.
- Als nächstes wird das Computer-Konto des Destination-DSAs zum Domänen-Controller heraufgestuft.
- Nun wird die Domänen-Partition repliziert. Dabei wird wie bei der Schema-Partition verfahren. Hier wird allerdings die *Callback-Funktion* `domain_chunk()` verwendet.
- Als letztes wird die Funktion `DsReplicaUpdateRefs()` (siehe auch `drsuapi.idl` unter [SaTe03] bzw. Anhang A2 ab Zeile 633) nacheinander für die Schema-Partition, die Konfigurations-Partition und die Domänen-Partition aufgerufen. Ich vermute, dass die Funktion `DsReplicaUpdateRefs()` jeweils einen Wert des Attributs „`repsTo`“ generiert, damit der Source-DSA den Destination-DSA später über Änderungen benachrichtigt. Dies muss aber noch weiter erforscht werden.

4.4 Zugriff auf Schema-Informationen

Um die Informationen des Verzeichnis-Schemas innerhalb von Samba einfach zugreifbar zu machen, wurde im Rahmen dieser Bachelorarbeit eine *Schema-API* und ein *Schema-Cache* entwickelt. Hier werden die verwendeten Strukturen und Konzepte kurz vorgestellt und auf Abschnitte im Quelltext verwiesen, wo Details nachzulesen sind. Siehe `source/dsdb/schema/schema.h`, `source/dsdb/schema/schema_init.c` und `source/dsdb/schema/schema_syntax.c` unter [SaTe01] bzw. Anhang A4, A10, A11.

4.4.1 Schema-Cache

Der *Schema-Cache* wird durch `struct dsdb_schema` realisiert (siehe `source/dsdb/schema/schema.h` unter [SaTe01] bzw. Anhang A4). Dort werden die folgenden Informationen bereitgehalten:

- Ein Array aus `struct dsdb_schema_oid_prefix` Elementen enthält die im Schema verwendeten *Prefix-Mappings*.
- Der Inhalt des Attributs „schemaInfo“ des Wurzel-Objekts der Schema-Partition wird als hexadezimaler String vorgehalten. Die hexadezimale String-Repräsentation wird benutzt, weil Sambas NDR-Schicht es so erwartet.
- Eine doppelt verkettete Liste aus `struct dsdb_attribute` Elementen enthält die Definitionen aller im Schema vorhandenen Attribute. Weil in `source/lib/util/dlinklist.h` (unter [SaTe01]) C-Macros zur einfachen Handhabung von doppelt verketteten Listen bereitstehen, werden diese hier verwendet.
- Eine doppelt verkettete Liste aus `struct dsdb_class` Elementen enthält die Definitionen aller im Schema vorhandenen Objekt-Klassen.

Definition der Attribute

Persistent werden die Attribut-Definitionen in Verzeichnis-Objekten der Objekt-Klasse „attributeSchema“ innerhalb der Schema-Partition gespeichert. Das `struct dsdb_attribute` enthält alle schemarelevanten Informationen der Attribut-Definition im Schema-Cache. Eine Beschreibung aller Elemente der Attribut-Definition ist unter [MSTN04] nachzulesen, hier werden nur die für die Replikation relevanten Elemente beschrieben:

- Der RDN des Verzeichnis-Objekts ist als String im Element `cn` abgelegt.
- Innerhalb von LDAP wird ein Attribut über einen Namen identifiziert, dieser ist im Element `LDAPDisplayName` zu finden.
- Jede Attribut-Definition hat eine weltweit eindeutige OID, diese ist in der String-Repräsentation im Attribut „attributeID“ gespeichert. Im `struct dsdb_attribute` wird sie im Element `attributeID_oid` abgelegt. Das Element `attributeID_id` enthält die durch das *Prefix-Mapping* generierte 32-Bit-ID der OID.
- Das Element `systemFlags` eine 32-Bit Integerzahl und enthält folgende Bit-Flags:
 - `0x00000001` bedeutet, das Attribut wird nicht repliziert.
 - `0x00000002` bedeutet, das Attribut muss zwingend zum *Global Catalog* (GC) repliziert werden. Weitere Informationen zum GC sind unter [MSTN07] verfügbar.
 - `0x00000004` bedeutet, das Attribut wird nicht gespeichert sondern bei Anfragen konstruiert.
- Jedes Attribut ist einer Attribut-Syntax zugeordnet. Dabei wird die benutzte Syntax durch die Kombination der Attribute „attributeSyntax“ „oMSyntax“ und „oMObjectClass“ festgelegt. Die Liste mit den in Active-Directory verwendeten Attribut-Syntaxen ist im

Abschnitt „Valid Syntaxes for Attributes in the Active Directory Schema“ in [MSTN04] beschrieben.

- In Samba werden die Attribut-Syntaxen durch das `struct dsdb_syntax` repräsentiert (siehe `source/dsdb/schema/schema.h` unter [SaTe1] bzw. Anhang A4 ab Zeile 30). Das Element `syntax` in `struct dsdb_attribute` referenziert die jeweils verwendete Syntax. Jeder Attribut-Syntax sind Konvertierungs-Funktionen zum Konvertieren zwischen den DRSUAPI- und der LDB-Repräsentationen zugeordnet, welche in `source/dsdb/schema/schema_syntaxes.c` unter [SaTe01] bzw. Anhang A11 implementiert sind.

Definition der Objekt-Klassen

Persistent werden die Klassen-Definitionen in Verzeichnis-Objekten der Objekt-Klasse „classSchema“ innerhalb der Schema-Partition gespeichert. Das `struct dsdb_class` enthält alle schemarelevanten Informationen der Attribut-Definition im Schema-Cache. Eine Beschreibung aller Elemente der Klassen-Definition ist unter [MSTN04] nachzulesen, hier werden nur die für die Replikation relevanten Elemente beschrieben:

- Der RDN des Verzeichnis-Objekts ist als String im Element `cn` abgelegt.
- Innerhalb von LDAP wird eine Objekt-Klasse über einen Namen identifiziert, dieser ist im Element `LDAPDisplayName` zu finden.
- Jede Klassen-Definition hat eine weltweit eindeutige OID, diese ist in der String-Repräsentation im Attribut „governsID“ gespeichert. Im `struct dsdb_class` wird sie im Element `governsID_oid` abgelegt. Das Element `governsID_id` enthält die durch das *Prefix-Mapping* generierte 32-Bit-ID der OID.

4.4.2 Die Schema-API-Funktionen zum Erstellen des Schema-Caches

Zum Laden des Verzeichnis-Schemas wurden eine Reihe von Funktionen entwickelt. Beim Erstellen des Schema-Caches sind 3 Grundschritte nötig, für die jeweils zwei Funktionsvarianten, eine für Informationen in DRSUAPI-Strukturen und eine für Informationen in LDB-Strukturen, bereitstehen:

- Zuerst muss das *Prefix-Mapping* und das Attribut „schemaInfo“ geladen werden. Siehe `dsdb_load_oid_mappings_ldb()` und `dsdb_load_oid_mappings_drсуapi()` in `source/dsdb/schema/schema_init.c` unter [SaTe01] bzw. Anhang A10 ab Zeile 33. Bei der LDB-Variante ist zubeachten, dass die *Prefix-Mappings* im nicht replizierten Attribut „prefixMap“ gespeichert werden. Da es bei Windows-2003 nicht möglich ist, den Attribut-Wert über LDAP abzufragen, wurde ein eigenes Format gewählt (siehe `drsblobs.idl` unter [SaTe03] bzw. Anhang A1 ab Zeile 147).
- Im zweiten Schritt müssen alle Attribute-Definitionen in den Schema-Cache geladen werden. Siehe `dsdb_attribute_from_ldb()` und `dsdb_attribute_from_drсуapi()` in `source/dsdb/schema/schema_init.c` unter [SaTe01] bzw. Anhang A10 ab Zeile 365 sowie ab Zeile 709.
- Im letzten Schritt müssen alle Klassen-Definitionen in den Schema-Cache geladen werden. Siehe `dsdb_class_from_ldb()` und `dsdb_class_from_drсуapi()` in `source/dsdb/schema/schema_init.c` unter [SaTe01] bzw. Anhang A10 ab Zeile 436 sowie ab Zeile 771.

Die Verwendung der DRSUAPI-Funktions-Varianten dient nur zum Replizieren der Schema-Partition und ist deshalb nur für diesen Zweck programmiert. Hier werden nur die notwendig-

sten Informationen im Schema-Cache abgelegt.

Das Laden der Klassen-Definitionen ist in der heutigen Implementation noch unvollständig und deckt nur die in dieser Bachelorarbeit verwendeten Funktionalitäten ab. Insbesondere fehlen noch die Attribut-Listen und die Beziehungen zwischen den Objekt-Klassen.

Der vollständige geladene Schema-Cache wird mit der Funktion `dsdb_set_schema()` an eine LDB-Verbindung gebunden (siehe `source/dsdb/schema/schema_init.c` unter [SaTe01] bzw. Anhang A10 ab Zeile 952).

4.4.3 Die Schema-API-Funktionen zum Benutzen des Schema-Caches

Der verwendete Schema-Cache einer LDB-Verbindung ist über die Funktion `dsdb_get_schema()` zugreifbar (siehe `source/dsdb/schema/schema_init.c` unter [SaTe01] bzw. Anhang A10 ab Zeile 952). Grundsätzlich sollte auf die Informationen im Schema-Cache nur über die dafür vorgesehenen Funktionen zugegriffen werden.

Zugriff auf die Prefix-Mappings

Die Funktionen `dsdb_get_oid_mappings_drstuapi()` und `dsdb_get_oid_mappings_ldb()` bieten Zugriff auf die Prefix-Mappings (siehe `source/dsdb/schema/schema_init.c` unter [SaTe01] bzw. Anhang A10 ab Zeile 123 und ab Zeile 160). Die Funktion `dsdb_verify_oid_mappings_drstuapi()` wird bei der Pull-Replikation zur Überprüfung der Schema-Version zwischen Source-DSA und Destination-DSA verwendet (siehe `source/dsdb/schema/schema_init.c` unter [SaTe01] bzw. Anhang A10 ab Zeile 188).

Suche von Attribut- und Klassen-Definitionen

Die Schema-API bietet Funktionen, um auf Attribut- und Klassen-Definitionen per OID, 32-Bit-ID oder LDAP-Attribut-Namen zuzugreifen. Im einzelnen sind dies die Funktionen:

- `dsdb_attribute_by_attributeID_id()`
(siehe `source/dsdb/schema/schema_init.c` unter [SaTe01] bzw. Anhang A10 ab Zeile 822)
- `dsdb_attribute_by_attributeID_oid()`
(siehe `source/dsdb/schema/schema_init.c` unter [SaTe01] bzw. Anhang A10 ab Zeile 843)
- `dsdb_attribute_by_LDAPDisplayName()`
(siehe `source/dsdb/schema/schema_init.c` unter [SaTe01] bzw. Anhang A10 ab Zeile 860)
- `dsdb_class_by_governsID_id()`
(siehe `source/dsdb/schema/schema_init.c` unter [SaTe01] bzw. Anhang A10 ab Zeile 877)
- `dsdb_class_by_governsID_oid()`
(siehe `source/dsdb/schema/schema_init.c` unter [SaTe01] bzw. Anhang A10 ab Zeile 898)
- `dsdb_class_by_LDAPDisplayName()`
(siehe `source/dsdb/schema/schema_init.c` unter [SaTe01] bzw. Anhang A10 ab Zeile 915)

Um den LDAP-Attribut-Namen anhand der 32-Bit-ID zu suchen, steht die Funktion `dsdb_LDAPDisplayName_by_id()` zur Verfügung, wobei es egal ist, ob mit der 32-Bit-ID ein Attribut oder eine Klasse identifiziert wird.

Konvertieren der Attribute zwischen DRSUAPI- und LDB-Repräsentation

Beim Laden der Attribut-Definitionen wird jedem Attribut mit Hilfe der Funktion `dsdb_syntax_for_attribute()` ein `struct dsdb_syntax` zugeordnet (siehe `source/dsdb/schema/schema_syntax.c` unter [SaTe01] bzw. Anhang A11 ab Zeile 1294). Mit

Hilfe der Funktionen `dsdb_attribute_drstuapi_to_ldb()` und `dsdb_attribute_ldb_to_drstuapi()` können Attribute spezifisch zur Attribut-Syntax zwischen der DRSUAPI-Repräsentation und der LDB-Repräsentation konvertiert werden (siehe `source/dsdb/schema/schema_syntax.c` unter [SaTe01] bzw. Anhang A11 ab Zeile 1319 und ab Zeile 1334). Im Speziellen sind damit `struct drstuapi_DsReplicaAttribute` (siehe `drstuapi.idl` unter [SaTe03] bzw. Anhang A2 ab Zeile 467) und `struct ldb_message_element` (siehe `source/lib/ldb/include/ldb.h` unter [SaTe01] ab Zeile 143) gemeint.

4.4.4 LDB-Modul zum Laden des Schema-Caches

Im Rahmen dieser Bachelorarbeit wurde das LDB-Modul „`schema_fsmo`“ entwickelt. Abbildung 4.4 zeigt, in welchem Zusammenhang es mit den anderen LDB-Modulen verwendet wird. Beim LDB-Verbindungsaufbau lädt es den Schema-Cache automatisch und bindet ihn mit der Funktion `dsdb_set_schema()` an die LDB-Verbindung, danach können andere Module darauf zugreifen. Der Quelltext ist unter `source/dsdb/samdb/ldb_modules/schema_fsmo.c` unter [SaTe01] bzw. Anhang A14 verfügbar.

4.5 Speicherung der replizierten Daten inklusive von Metadaten

Um die von einem Source-DSA replizierten Daten zu interpretieren und in den lokalen Datenbestand aufzunehmen, ist eine komplexe Abfolge von Operationen notwendig. Im Rahmen dieser Bachelorarbeit wurde die Funktion `dsdb_extended_replicated_objects_commit()` entwickelt (siehe `source/dsdb/repl/replicated_objects.c` unter [SaTe01] bzw. Anhang A12 ab Zeile 338). Die Funktion ist grundlegend in zwei Phasen unterteilt. In der ersten Phase werden die empfangenen Daten, wie in Abschnitt 3.6.2 beschrieben, interpretiert und aufbereitet. In der zweiten Phase werden Konflikte, wie in Abschnitt 3.6.4 erklärt, gelöst und die Daten persistent gespeichert.

4.5.1 Erste Phase: Interpretation und Aufbereitung

Da das Interpretieren und Aufbereiten der Daten sehr komplex ist und einen nicht trivialen Anteil der CPU-Zeit, bezogen auf die Gesamtfunktion, in Anspruch nimmt, spielt sich die erste Phase „oberhalb“ der LDB-API ab. Also nicht innerhalb einer LDB-Transaktion. Konkret werden folgende Schritte abgearbeitet:

- Zuerst wird anhand des geladenen Verzeichnis-Schemas und des vom Source-DSA gesendeten *Prefix-Mappings* überprüft, ob auf dem Source-DSA und dem Destination-DSA die gleiche Version des Schemas vorhanden ist. Ist dies nicht der Fall, wird mit einer Fehler-Meldung abgebrochen.
- Dann werden in einer Schleife alle Verzeichnis-Objekte von der DRSUAPI-Repräsentation in die LDB-Repräsentation umgewandelt. Dies erledigt die Hilfsfunktion `dsdb_convert_object()` (siehe `source/dsdb/repl/replicated_objects.c` unter [SaTe01] bzw. Anhang A12 ab Zeile 190).
- Bei jedem Verzeichnis-Objekt werden dabei alle Attribute zuerst gegebenenfalls entschlüsselt, was in den Funktionen `dsdb_decrypt_attribute()` und `dsdb_decrypt_attribute_value()` implementiert ist (siehe `source/dsdb/repl/replicated_objects.c` unter [SaTe01] bzw. Anhang A12 ab Zeile 132 und ab Zeile 33). Danach werden die Attribut-Werte mit Hilfe der Funktion `dsdb_attribute_drsuapi_to_ldb()` (siehe `source/dsdb/schema/schema_syntax.c` unter [SaTe01] bzw. Anhang A11 ab Zeile 1319) in die LDB-Repräsentation gebracht.

Dabei werden die Daten zu einem `struct dsdb_extended_replicated_objects` aufbereitet, worin die Verzeichnis-Objekte, als Array von `struct dsdb_extended_replicated_object` Elementen, sowie der DN der Verzeichnis-Partition, der *Up-To-Date-Vector* des Source-DSAs und die Informationen für das Attribut „*repsFrom*“ bezüglich des Source-DSAs enthalten sind (siehe `source/dsdb/samdb/samdb.h` unter [SaTe01] bzw. Anhang A3 ab Zeile 55). Unter Verwendung der *LDB-Extended-Operation* `DSDB_EXTENDED_REPLICATED_OBJECTS_OID` wird das `struct dsdb_extended_replicated_objects` der LDB-API übergeben.

4.5.2 Zweite Phase: Konfliktauflösung und Speicherung

Die zweite Phase beinhaltet das Auflösen von eventuell auftretenden Konflikten und letztendlich das Speichern der replizierten Verzeichnis-Objekte sowie der Attribute „*replUpToDateVector*“ und „*repsFrom*“. All dies wird innerhalb einer LDB-Transaktion erledigt, d. h. garantiert werden alle Änderungen vollständig oder gar nicht gespeichert. Die *LDB-Extended-Operation* `DSDB_EXTENDED_REPLICATED_OBJECTS_OID` wird von der

Funktion `replmd_extended_replicated_objects()` des LDB-Modules „`repl_meta_data`“ implementiert (siehe `source/dsdb/samdb/ldb_modules/repl_meta_data.c` unter [SaTe01] bzw. Anhang A13 ab Zeile 55). Wo das LDB-Modul „`repl_meta_data`“ im Bezug zu den anderen LDB-Modulen zu finden ist, zeigt Abbildung 4.4.

Innerhalb der Funktion `replmd_extended_replicated_objects()` wird die folgende Logik in einer Schleife auf jedes replizierte Verzeichnis-Objekt angewendet:

- In der Funktion `replmd_replicated_apply_search()` wird jedes Objekt anhand seiner `objectGUID` gesucht (siehe `source/dsdb/samdb/ldb_modules/repl_meta_data.c` unter [SaTe01] bzw. Anhang A13 ab Zeile 1095).
- Wird das Objekt nicht gefunden, setzt die Funktion `replmd_replicated_apply_add()` noch die lokale USN für das Objekt und speichert es dann (siehe `source/dsdb/samdb/ldb_modules/repl_meta_data.c` unter [SaTe01] bzw. Anhang A13 ab Zeile 723).
- Ist das Objekt schon vorhanden, fügt die Funktion `replmd_replicated_apply_merge()` die Daten, der alten und neuen Version des Objekts zusammen. Dies geschieht anhand der in Funktion `replmd_replPropertyMetaData1_conflict_compare()` implementierten Konfliktauflösung auf Attribut-Ebene. Dann wird die lokale USN an den benötigten Stellen gesetzt und das Objekt gespeichert (siehe `source/dsdb/samdb/ldb_modules/repl_meta_data.c` unter [SaTe01] bzw. Anhang A13 ab Zeile 865 und ab Zeile 817).
- Die letzte Teilaufgabe besteht darin, die Attribute „`replUpToDateVector`“ und „`repsFrom`“ zu aktualisieren. Die Funktionen `replmd_replicated_uptodate_search()` und `replmd_replicated_uptodate_modify()` erledigen diese Aufgabe (siehe `source/dsdb/samdb/ldb_modules/repl_meta_data.c` unter [SaTe01] bzw. Anhang A13 ab Zeile 1512 und ab Zeile 1180).

Die Konflikte auf DN-Ebene sind bislang noch nicht implementiert, weil diese Konflikte bei der initialen Replikation nicht auftreten. Außerdem besteht noch weiterer Forschungsbedarf, um diese Fälle korrekt implementieren zu können.

4.5.3 Originating-Add

Das LDB-Modul „`repl_meta_data`“ ist zusätzlich für die Pflege der Metadaten bei *Originating-Updates* verantwortlich. Im Rahmen dieser Bachelorarbeit wurde der Fall des *Originating-Add* implementiert. Die Funktion `replmd_add_originating()` erledigt die unter Abschnitt 3.5.1 beschriebenen Aufgaben (siehe `source/dsdb/samdb/ldb_modules/repl_meta_data.c` unter [SaTe01] bzw. Anhang A13 ab Zeile 290). Um die anderen Fälle von *Originating-Updates* korrekt implementieren zu können, müssen sie noch detaillierter erforscht werden.

4.6 Der „NET-API-BECOME-DC“ Test

Um die in dieser Bachelorarbeit implementierten Funktionen zu testen, wurde Sambas Testprogramm `smbtorture` um den Test „NET-API-BECOME-DC“ erweitert. Dieser Test ist in der Funktion `torture_net_become_dc()` implementiert (siehe `source/torture/libnet/libnet_BecomeDC.c` unter [SaTe01] bzw. Anhang A7 ab Zeile 729).

4.6.1 Erstellen eines Test-Computer-Kontos

In `smbtorture` stehen die Funktionen `torture_join_domain()` und `torture_leave_domain()` zum Erstellen bzw. Löschen eines Test-Computer-Kontos bereit (siehe `source/torture/rpc/testjoin.c` unter [SaTe01] ab Zeile 297 und ab Zeile 491). Diese Funktionen werden zu Beginn bzw. zum Ende von der Funktion `torture_net_become_dc()` aufgerufen.

4.6.2 Heraufstufen zum Domänen-Controller

Unter Benutzung des temporären Computer-Kontos wird die Funktion `libnet_BecomeDC()` getestet. Alle zum Testen benötigten Daten werden im `struct test_become_dc_state` (siehe `source/torture/libnet/libnet_BecomeDC.c` unter [SaTe01] ab Zeile 97) gesammelt und allen Hilfsfunktionen bereitgestellt. Im Folgenden wird beschrieben, welche Aufgaben im Einzelnen durch die an `libnet_BecomeDC()` übergebenen *Callback-Funktionen* erledigt werden.

Callback-Funktion für `check_options()`

Die Funktion `test_become_dc_check_options()` gibt die von `libnet_BecomeDC()` eingeholten Information am Bildschirm aus (siehe `source/torture/libnet/libnet_BecomeDC.c` unter [SaTe01] ab Zeile 97).

Callback-Funktion für `prepare_db()`

Die Funktion `test_become_dc_prepare_db()` bereitet die LDB-Dateien vor und konfiguriert, welche LDB-Module benutzt werden sollen. Dieses sogenannte *Provisioning* ist in der Programmiersprache EJS implementiert. Deshalb konstruiert die Funktion `test_become_dc_prepare_db()` in einem String ein EJS-Programm und führt damit die EJS-Funktion `provision_become_dc()` aus. Danach wird die vorbereitete LDB verbunden (siehe `source/torture/libnet/libnet_BecomeDC.c` unter [SaTe01] ab Zeile 149 und `source/scripting/libjs/provision.js` unter [SaTe01] ab Zeile 454).

Callback-Funktion für `schema_chunk()`

Die Funktion `test_become_dc_schema_chunk()` nimmt die replizierten Verzeichnis-Objekte der Schema-Partition entgegen. Diese werden erstmal im `struct test_become_dc_state` zwischengespeichert (siehe `source/torture/libnet/libnet_BecomeDC.c` unter [SaTe01] bzw. Anhang A7 ab Zeile 519). Wurden alle Objekte der Schema-Partition empfangen, wird in der Funktion `test_apply_schema()` mit Hilfe der Funktionen `dsdb_attribute_from_drstuapi()` und `dsdb_class_from_drstuapi()` ein Hilfs-Schema erstellt, welches daraufhin mit der Funktion `dsdb_set_schema()` an die LDB-Verbindung gebunden wird. Danach werden die Verzeichnis-Objekte durch einen Aufruf der Funktion `dsdb_extended_replicated_objects_commit()` in der LDB gespeichert (siehe `source/torture/libnet/libnet_BecomeDC.c` unter [SaTe01] bzw. Anhang A7 ab Zeile 285). Danach werden die *Prefix-Mappings* in das nicht replizierte

Attribut „prefixMap“ geschrieben. Daraufhin wird die LDB-Verbindung geschlossen und wiederhergestellt, bei diesem Vorgang wird das Hilfs-Schema verworfen und das vollständig replizierte Verzeichnis-Schema vom LDB-Modul „schema_fsmo“ geladen.

Callback-Funktion für `config_chunk()` und `domain_chunk()`

Die replizierten Verzeichnis-Objekte der Konfigurations-Partition und der Domänen-Partition werden von der Funktion `test_become_dc_store_chunk()` entgegengenommen, welche daraufhin mit Hilfe der Funktion `dsdb_extended_replicated_objects_commit()` in der LDB gespeichert werden (siehe `source/torture/libnet/libnet_BecomeDC.c` unter [SaTe01] bzw. Anhang A7 ab Zeile 590).

Nach dem alle Daten repliziert wurden, wird die LDB-Verbindung zum Testen noch mal geschlossen und wiederhergestellt.

4.6.3 Herunterstufen und Löschen des Computer-Kontos

Am Ende des „NET-API-BECOME-DC“ Tests wird das Computer-Konto durch die Funktion `libnet_UnbecomeDC()` wieder zu einem Domänen-Mitglied heruntergestuft und anschliessend mit der Funktion `torture_leave_domain()` gelöscht.

4.6.4 Ausführen von „NET-API-BECOME-DC“ via `smbtorture`

Achtung: `smbtorture` sollte ausschließlich in Test-Umgebungen benutzt werden, es können wichtige Daten verloren gehen!

Die Installation von Samba 4.0 ist in Anhang A17 beschrieben.

Der „NET-API-BECOME-DC“ Test wird wie folgt aufgerufen:

```
#>bin/smbtorture ncacl_np:w2k3-21.w2k3-thesis.vynet24.vm.base \
-W W2K3-THESIS -TODO-realm w2k3-thesis.vynet24.vm.base \
-U administrator%test \
NET-API-BECOME-DC
```

Mit einer Zusatzoption werden alle replizierten Daten detailliert ausgegeben.

Achtung: Die Benutzung von `less` ist wichtig, da ungefähr 500 Tausend Zeilen generiert werden!:

```
#>bin/smbtorture ncacl_np:w2k3-21.w2k3-thesis.vynet24.vm.base \
-W W2K3-THESIS -TODO-realm w2k3-thesis.vynet24.vm.base \
-U administrator%test \
NET-API-BECOME-DC \
--option="become dc:dump objects=yes" | less
```

5 Fazit

Die im Kapitel „Zielsetzung“ gesetzten Ziele dieser Bachelorarbeit wurden erfolgreich erarbeitet.

5.1 Was erreicht wurde

Die wichtigsten Aspekte der Active-Directory-Replikation sind erforscht und dokumentiert worden, dazu gehören:

- Die benötigten Schritte, um ein Computer-Konto für einen Domänen-Controller im Active-Directory zu erstellen.
- Die benötigten Schritte, um alle Verzeichnis-Daten von einem Windows-2003-Server zu replizieren.
- Die benötigten Schritte, um die replizierten Verzeichnis-Daten zu interpretieren. Insbesondere können auch die Passwort-Informationen interpretiert werden.

Außerdem bilden die implementierten Funktionen dieser Bachelorarbeit eine solide Grundlage für die zukünftige Entwicklung von Samba 4.0.

5.2 Was noch zu tun ist

Da Samba 4.0 ein vollwertiger Domänen-Controller für Active-Directory-Domänen werden soll, sind natürlich noch sehr viele Details zu erforschen und zu implementieren.

Die folgenden Punkte sind noch zu erforschen:

- Die Konfliktauflösung auf DN-Ebene.
- Die Behandlung von *Linked-Attributes*.
- Der gesamte Themenbereich der *Replikations-Topologie*.
- Die Durchsetzung der Schema-Restriktionen.
- Die Durchsetzung von Zugriffs-Beschränkungen.

Die folgenden Punkte sind noch zu implementieren:

- Die Metadaten müssen bei allen *Originating-Updates* gepflegt werden.
- Die *Pull-Replikation* muss in Sambas Server-Programm `smbd` für die Rollen Source-DSA und Destination-DSA implementiert werden.
- Und natürlich alle Aspekte, die noch zu erforschen sind.

6 Literaturverzeichnis

- [AppWeb01] Mbedthis Software LLC: Embedded JavaScript;
<http://www.appwebserver.org/products/ejs/ejs.html>; Mbedthis Software LLC, 2003-2006 [letzter Besuch: 16.03.2007].
- [FSF01] Free Software Foundation: GNU General Public License;
<http://www.gnu.org/licenses/gpl.html>; Free Software Foundation, 1991 [letzter Besuch: 16.03.2007].
- [FSF02] Free Software Foundation: The Free Software Definition;
<http://www.gnu.org/philosophy/free-sw.html>; Free Software Foundation, 1996-2006 [letzter Besuch: 16.03.2007].
- [Gla2006] E. Glass: The NTLM Authentication Protocol and Security Support Provider;
<http://davenport.sourceforge.net/ntlm.html>; 2003,2006 [letzter Besuch: 16.03.2007].
- [GoGr01] D. Sterndark: RC4 Algorithm revealed;
<http://groups.google.com/group/sci.crypt/msg/10a300c9d21afca0>; 1994 [letzter Besuch: 16.03.2007].
- [Her2003] C. Hertel: Implementing CIFS; <http://ubiqx.org/cifs/SMB.html>; 1999-2003 [letzter Besuch: 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 [letzter Besuch: 16.03.2007].
- [KoMe2005] M. Kohlgraf; S. Metzmaker: Active Directory Replikation; (Studienarbeit im Fach Datennetze, WS04/05; Betreuer Prof. Dr.-Ing. Andreas Grebe) FH Köln, Institut für Nachrichtentechnik 2005, (vgl. a. [KoMe2005W]).
- [KoMe2005W] M. Kohlgraf; S. Metzmaker: Active Directory Replikation;
http://samba.org/~metze/presentations/2005/Datennetze1/WS0405_Acitive_Directory_Replication.pdf; 2005 [letzter Besuch: 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 [letzter Besuch: 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 [letzter Besuch: 16.03.2007].
- [MSDN01] Microsoft MSDN: LDAP_SERVER_SHOW_DELETED_OID;
<http://msdn2.microsoft.com/en-us/library/aa366989.aspx>; Microsoft Corporation, 2007 [letzter Besuch: 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.msp>; Microsoft Corporation, 2006 [letzter Besuch: 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 [letzter Besuch: 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 [letzter Besuch: 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 [letzter Besuch: 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 [letzter Besuch: 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 [letzter Besuch: 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 [letzter Besuch: 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 [letzter Besuch: 16.03.2007].
- [OpGr02] The Open Group: Transfer Syntax NDR;
<http://www.opengroup.org/onlinepubs/9629399/chap14.htm>; The Open Group, 1997 [letzter Besuch: 16.03.2007].
- [OpGr03] The Open Group: Interface Definition Language;
<http://www.opengroup.org/onlinepubs/9629399/chap4.htm>; The Open Group, 1997 [letzter Besuch: 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 [letzter Besuch: 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 [letzter Besuch: 16.03.2007].
- [RFC791] J. Postel: INTERNET PROTOCOL; <http://www.ietf.org/rfc/rfc791.txt>; The Internet Engineering Task Force, 1980 [letzter Besuch: 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 [letzter Besuch: 16.03.2007].

- [RFC1001] Network Working Group: PROTOCOL STANDARD FOR A NetBIOS SERVICE ON A TCP/UDP TRANSPORT: CONCEPTS AND METHODS; <http://www.ietf.org/rfc/rfc1001.txt> The Internet Engineering Task Force, 1987 [letzter Besuch: 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 [letzter Besuch: 16.03.2007].
- [RFC1034] P. Mockapetris: DOMAIN NAMES - CONCEPTS AND FACILITIES; <http://www.ietf.org/rfc/rfc1034.txt>; The Internet Engineering Task Force, 1987 [letzter Besuch: 16.03.2007].
- [RFC1035] P. Mockapetris: DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION; <http://www.ietf.org/rfc/rfc1035.txt>; The Internet Engineering Task Force, 1987 [letzter Besuch: 16.03.2007].
- [RFC1321] R. Rivest: The MD5 Message-Digest Algorithm; <http://www.ietf.org/rfc/rfc1321.txt>; The Internet Engineering Task Force, 1992 [letzter Besuch: 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 [letzter Besuch: 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 [letzter Besuch: 16.03.2007].
- [RFC2222] J. Myers: Simple Authentication and Security Layer; <http://www.ietf.org/rfc/rfc2222.txt>; The Internet Engineering Task Force, 1997 [letzter Besuch: 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 [letzter Besuch: 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 [letzter Besuch: 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 [letzter Besuch: 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 [letzter Besuch: 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 [letzter Besuch: 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 [letzter Besuch: 16.03.2007].

- [RFC4648] S. Josefsson: The Base16, Base32, and Base64 Data Encodings; <http://www.ietf.org/rfc/rfc4647.txt>; The Internet Engineering Task Force, 2006; [letzter Besuch: 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 [letzter Besuch: 16.03.2007].
- [SaTe02] Samba-Team: Samba Development; <http://devel.samba.org/>; Samba-Team, 2007 [letzter Besuch: 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/ibrpc/idl/?rev=21859; Samba-Team, 2007 [letzter Besuch: 16.03.2007].
- [SaTe04] Samba-Team: Samba - Opening Windows to a Wider World; <http://www.samba.org/>; Samba-Team, 2007 [letzter Besuch: 16.03.2007].
- [SaTe04] Samba-Team: LDB; <http://ldb.samba.org/>; Samba-Team, 2007 [letzter Besuch: 16.03.2007].
- [Tridge01] A. Triggell: How Samba was written; http://samba.org/ftp/tridge/misc/french_cafe.txt; 2003 [letzter Besuch: 16.03.2007].
- [Tridge02] A. Triggell: Network Analysis Techniques; http://us4.samba.org/samba/ftp/slides/net_analysis.pdf; 2003 [letzter Besuch: 16.03.2007].
- [WikiPe01] Wikimedia Foundation Inc.: OSI-Modell; <http://de.wikipedia.org/wiki/OSI-Modell>; Wikimedia Foundation Inc., 2007 [letzter Besuch: 16.03.2007].
- [WikiPe02] Wikimedia Foundation Inc.: Linux; <http://de.wikipedia.org/wiki/OSI-Linux>; Wikimedia Foundation Inc., 2007 [letzter Besuch: 16.03.2007].
- [WikiPe03] Wikimedia Foundation Inc.: RC4; <http://de.wikipedia.org/wiki/RC4>; Wikimedia Foundation Inc., 2007 [letzter Besuch: 16.03.2007].

7 Abkürzungsverzeichnis

BDC	Backup-Domänen-Controller
CIFS	Common Internet File System
CLDAP	Connection-less Lightweight Directory Access Protocol
C	Programmiersprache C
CRC32	Cyclic Redundancy Check 32
DC	Domänen-Controller
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 Protokoll
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	Namens-Kontext (engl. „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-Domänen-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

8 Stichwortverzeichnis

Symbole

32-Bit-Id 23, 54
 32-Bit-ID 52, 53

A

aio_event 40
 ARCFOUR 34
 asq 48
 Asynchrone Funktion 50
 Asynchrone Funktionsaufrufe 42
 AttId 23, 34, 52, 54
 Attribut 8, 10, 52
 Attribut-Definition 52, 52, 53, 54
 Attribut-Syntax 34, 54
 attributeID 23, 52
 attributeSyntax 52

B

Backup-Domänen-Controller (BDC) 1
 Base64 8

C

C (Programmiersprache) 39
 Callback-Funktion 42, 50, 50, 58, 58, 58,
 59
 Change-Notify 16, 38, 38
 ChangeNotify 22
 check_options() 50, 58
 CIFS 2
 CLDAP 14
 COMPOSITE 50
 Computer-Konto 50, 50, 50, 58, 59
 config_chunk() 50, 59
 CRC32 34

D

Database-Layer 45, 45
 dBCSPwd 34, 49
 DCERPC 2, 4, 14, 14
 Deflate (Komprimierung) 33
 Destination-DSA 16, 21, 22, 32, 33, 33,
 36, 36, 37, 38, 50, 50, 54, 56
 DIGEST-MD5 1
 DIT 7
 DN 7
 DNS 1
 DNS-Domänen-Name 8, 50
 DNS-Name 50

domain_chunk() 50
 Domänen-Baum 8
 Domänen-Controller 3, 3, 3, 5, 50, 50, 58
 Domänen-Controller (DC) 1
 Domänen-Mitglied 3, 5, 50, 50, 50, 59
 Domänen-Partition 8, 11, 50, 59
 Domänen-SID 18, 50
 Domänen-Wald 8, 11, 50
 DRSUAPI 1, 3, 3, 4, 15, 21, 45, 50, 53, 54,
 54
 DRSUAPI-Client 3
 DRSUAPI-Repräsentation 56
 DRSUAPI-Server 3
 drsuapi_DsGetNCChangeCtrl 32
 drsuapi_DsGetNCChangeCtrl6 32, 33
 drsuapi_DsGetNCChangeRequest5 32
 drsuapi_DsGetNCChangeRequest8 32, 33
 drsuapi_DsReplicaMetaData 34
 drsuapi_DsReplicaObjectListItemEx 33,
 34
 DSA 7, 8, 20, 22, 32, 37, 37, 37, 37
 DSA-GUID 19, 33, 38, 38
 DSA-Invocation-Id 19, 21, 33, 37
 DSA-Operational-Layer 45, 45, 46
 DsAddEntry() 4, 50
 DsAttributeValue 26
 DsAttributeValueCtr 26
 DSDB 5, 45
 dsdb_attribute 52
 dsdb_class 53
 dsdb_decrypt_attribute_value() 34
 dsdb_extended_replicated_object 56, 56,
 57
 dsdb_extended_replicated_objects 56, 56,
 57
 dsdb_extended_replicated_objects_commit()
 56, 56, 56, 57, 58, 59
 DSDB_EXTENDED_REPLICATED_OBJE
 CTS_OID 49, 56, 56, 57
 dsdb_syntax 34, 52
 DsGetNCChanges() 4, 4, 32, 33, 33, 34,
 50
 DsReplicaAttribute 26
 DsReplicaAttributeCtr 26
 DsReplicaCursor 21
 DsReplicaCursorCtr 21
 DsReplicaHighWaterMark 21
 DsReplicaMetaDataCtr 25, 27
 DsReplicaObject 26
 DsReplicaObjectIdentifier 18, 26, 27
 DsReplicaObjectListItem 27
 DsReplicaObjectListItemEx 27

- DsReplicaOIDMapping 25
 DsReplicaOIDMapping_Ctr 25
 DsReplicaSync 38
 DsReplicaUpdateRefs() 4, 50
- E**
- EJS 39, 39, 58
 Endpoint-Mapper 14
 epoll() 39
 EVENTS 40
 extended_dn 48
- F**
- fd_event 40
 first_object 33
 Freie Software 2
 FSMO 5
- G**
- Global Catalog (GC) 52
 governsID 23, 53
 GPL 2
 GSSAPI 12
 GUID 18
- H**
- HASH-Funktion 34
 High-Water-Mark 33, 33, 36, 37
 highestCommittedUSN 20
 Hostname 50
- I**
- IDL 4, 14, 14, 39
 instanceType 28, 30
 IP 12
 IP-Adresse 50
 isDeleted 31, 48
- K**
- KCC 32
 Kerberos 5 1, 3, 4, 12
 Kerberos-Authentifizierung 3
 Key Distribution Center (KDC) 3
 Klasse 10, 53
 Klassen-Definition 52, 53, 53, 54
 kludge_acl 47
 Komprimierung 33
 Konfiguration-Partition 11
 Konfigurations-Partition 8, 50, 59
- L**
- LDAP 1, 1, 3, 4, 13, 44, 45, 50, 53, 54, 54
 LDAP-API 44
 LDAP-Control 44, 48
 LDAP-Extended-Operation 44
 LDAPDisplayName 52, 53, 54
 LDB 44, 53, 54, 54, 58, 58, 59
 LDB-API 44, 46, 56, 56, 57
 LDB-Backend 44, 45
 LDB-Control 44
 LDB-Extended-Operation 6, 44, 49, 56, 56
 LDB-Modul 6, 6, 44, 45, 46, 47, 47, 48,
 48, 48, 48, 48, 48, 48, 49, 49, 49, 55, 56, 57,
 57
 LDB-Module 5
 LDB-Repräsentation 56
 LDB-Transaktion 56
 LDB-Verbindung 54, 55, 58, 59
 ldb_add() 6
 ldb_extended() 6
 LDIF 8, 44
 LIBNET 5, 50
 libnet_BecomeDC() 5, 5, 50, 50, 50, 50,
 50, 58, 58, 58, 58, 59
 libnet_BecomeDC_Callbacks 50, 50
 libnet_UnbecomeDC() 50, 59
 Linked-Attributes 6, 33
 Linux 2
 ImpWdHistory 49
 Log-basierte Replikation 17, 17
 Lost And Found 37
- M**
- MD5 34
 Metadaten 6, 25, 27, 33, 34, 37, 37, 37, 57,
 57
 Mixed-Mode 5
 msDs-Behavior-Version 29
 MSZIP (Komprimierung) 33
 Multimaster-Replikation 16, 16, 37, 37, 37
- N**
- name 30
 Namens-Kontext 8
 naming_fsmo 48
 Native-Mode 5
 NBT 12
 NBT-Session-Service 14
 NDR 4, 14
 NET-API-BECOME-DC 58, 59, 59
 NETBIOS-Domänen-Name 50
 NETBIOS-Name 50
 NETLOGON 1, 15, 45

- Non-Blocking-I/O 39, 40
 NT Directory Service (NTDS) 1
 NTDS Settings 19
 nTDSDSA 19
 NTLMSSP 12
 ntPwdHistory 49
- O**
- objectGUID 18, 30, 37
 objectSid 18
 Objekt-DN 34, 37
 Objekt-GUID 34
 Objekt-Klasse 10, 53
 OID 23, 52, 53, 54
 operational 48
 Originating-Add 30, 49, 57
 Originating-Delete 31
 Originating-Modify 30, 31
 Originating-Move 31
 Originating-Update 38
 Originating-Updates 25, 30, 49, 57
 originating_invocation_id 37
 originating_usn 37
 OSI-Modell 4
- P**
- paged_results 48
 partition 48
 partition_hash 49
 Passwörter 34, 49
 pdc_fsmo 48
 pidl 39
 Prefix-Mapping 33, 34, 52, 53, 53, 54, 54
 prefixMap 23, 53
 prepare_db() 50, 58
 Primary-Domänen-Controller (PDC) 1
 Provisioning 58
 provision_become_dc() 58
 Pull-Replikation 16, 16, 32, 38
 Push-Replikation 16, 16
 pwdLastSet 49
- R**
- rdb_name 48
 RDN 7, 30, 34, 37, 52, 53
 RDN-Attribut 34
 Replicating-Updates 25, 37, 37, 49
 replica_flags 33, 33, 34
 Replikations-Nachbarn 22
 Replikations-Topologie 17, 32
 replPropertyMetaData 25, 30, 30
 replPropertyMetaData1 34
- replUpToDateVector 32, 56, 57
 repl_meta_data 6, 6, 49, 56, 57, 57
 repsFrom 22, 32, 56, 56, 57
 repsFromToBlob 22
 repsTo 22, 32
 Request-Handle 42
 RID 18
 RootDSE 7, 8, 47
- S**
- Samba 2, 59
 Samba-Team 2
 SAMDB 45, 45
 samldb 48
 SAMR 1, 14, 45
 SASL 12
 Schema 3, 5, 5, 10, 11, 44, 45, 49, 53, 54,
 54, 54, 54, 55, 56, 58
 Schema-API 5, 49, 53, 54, 54, 54, 55, 58
 Schema-Cache 5, 34, 49, 52, 53, 53, 54,
 54, 54, 54, 55, 58, 59
 Schema-Master 5, 49
 Schema-Partition 8, 33, 50, 58
 Schema-Version 54, 56
 schemaInfo 33
 schema_chunk() 50, 58
 schema_fsmo 5, 49, 55, 58
 Security Account Manager (SAM) 1
 select() 39
 Server-Rollen 5
 server_sort 48
 show_deleted 48
 SID 18
 signal_event 40
 Singlemaster-Replikation 16, 16
 Sitzungsschlüssel 34, 50
 SMB 2, 2, 14
 smbld 39
 smbscript 39
 smbtorure 39, 59
 smbtorure 58, 58, 58
 Sockets 42
 Source-DSA 16, 21, 22, 32, 33, 33, 36, 36,
 37, 38, 38, 50, 50, 54, 56, 56
 SPNEGO 12
 Status-basierte Replikation 17, 17
 Storage-Layer 45, 45
 Store-and-Forward 17, 31
 supplementalCredentials 49
 supplenetalCredentials 34
 systemFlags 28, 31, 52

T

TALLOC	40
TCP	12
TDB	44, 44, 45
timed_event	40
Tombstone	31, 31

U

UDP	12
unicodePwd	34, 49
Unix-Betriebssystem	39, 39
Unix/Linux	2
Up-To-Date-Vector	21, 33, 33, 37, 56
USN	20, 21, 33, 57
uSNChanged	28, 30, 30, 36
uSNCreated	28, 30

V

Verschlüsselung	34
Verzeichnis	5
Verzeichnis-Hierarchie	13
Verzeichnis-Partition	8, 33, 33, 56
Verzeichnisdienst	7

W

W2K	1
W2K3	1
whenChanged	28, 30, 30, 34
whenCreated	28, 30
Windows-2000	1, 32
Windows-2000-Server	1
Windows-2003	1, 45, 45, 45
Windows-2003-Server	1, 3, 5, 8
Windows-Domäne	1
Windows-NT4-Domäne	1

X

XPRESS (Komprimierung)	33
------------------------------	----

9 Anhang

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_lsec 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 |     } replPropertyMetaDataCtrl;
35 |
36 |     typedef [nodiscriminant] union {
37 |         [case(1)] replPropertyMetaDataCtrl ctrl;
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 |     } replUpToDateVectorCtrl;
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)] replUpToDateVectorCtrl ctrl;
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_lsec last_success;
96     NTTIME_lsec 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 ctrl1;
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 } partialAttributeSetCtrl;
132
133 typedef [nodiscriminant] union {
134     [case(1)] partialAttributeSetCtrl ctrl1;
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 [vl_enum] enum {
158     PREFIX_MAP_VERSION_DSDB = 0x44544442
159 } prefixMapVersion;

```

```

160 |
161 |     typedef [nodiscard] 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 [nodiscard, 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,
->
189 |         extra.uptodateness_vector.version, 0))]
190 |             uint32 extra_length;
191 |         drsuapi_DsReplicaHighWaterMark highwatermark;
192 |         GUID guid1;
193 |         [switch_is(extra_length)] ldapControlDirSyncExtra extra;
194 |     } ldapControlDirSyncBlob;
195 |
196 |     typedef [public, relative_base] struct {
197 |         [charset(DOS), value("MSDS")] uint8 msds[4];
198 |         [subcontext(0)] ldapControlDirSyncBlob blob;
199 |     } ldapControlDirSyncCookie;
200 |
201 |     void decode_ldapControlDirSync(
202 |         [in] ldapControlDirSyncCookie cookie
203 |     );
204 |
205 |     typedef struct {
206 |         [value(2*strlen_m(name))] uint16 name_len;
207 |         [value(strlen(data))] uint16 data_len;
208 |         uint16 unknown1; /* 2 for name = 'Packages', 1 for name = 'Primary:*' */
209 |         [charset(UTF16)] uint8 name[name_len];
210 |         /*
211 |         * the data field contains data as HEX strings
212 |         *
213 |         * 'Packages':
214 |         *   data contains the list of packages
215 |         *   as non terminated UTF16 strings with
216 |         *   a UTF16 NULL byte as separator
217 |         *
218 |         * 'Primary:Kerberos':
219 |         *   ...
220 |         *
221 |         * 'Primary:WDigest':
222 |         *   ...
223 |         *
224 |         * 'Primary:CLEARTEXT':
225 |         *   data contains the cleartext password
226 |         *   as UTF16 string encoded as HEX string
227 |         */
228 |         [charset(DOS)] uint8 data[data_len];
229 |     } supplementalCredentialsPackage;
230 |
231 |     /* this are 0x30 (48) whitespaces (0x20) followed by 'P' (0x50) */
232 |     const string SUPPLEMENTAL_CREDENTIALS_PREFIX = "
->
233 |         P";
234 |
235 |     typedef [gensize] struct {
236 |         [value(SUPPLEMENTAL_CREDENTIALS_PREFIX), charset(UTF16)] uint16
->
237 |         prefix[0x31];
238 |         uint16 num_packages;
239 |         supplementalCredentialsPackage packages[num_packages];
240 |     } supplementalCredentialsSubBlob;
241 |
242 |     typedef [public] struct {
243 |         [value(0)] uint32 unknown1;
244 |         [value(ndr_size_supplementalCredentialsSubBlob(&sub, ndr->flags))] uint32
->
245 |         __ndr_size;
246 |         [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_REMAINING)] string *string;
->     } package_PrimaryKerberosString;
263 |
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 |         uulong unknown3[num_keys];
282 |         uulong 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)] uulong unknown4;
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 |     } trustAuthInOutCtrl;
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)] trustAuthInOutCtrl ctrl;
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("\\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_CRYPT0_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 ul;
66 |    } drsuapi_DsBindInfo24;
67 |
68 |    /* this is used by w2k3 */
69 |    typedef struct {
70 |        drsuapi_SupportedExtensions supported_extensions;
71 |        GUID site_guid;
72 |        uint32 ul;
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,gen_size] 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
-> |     repsFromToOtherInfo 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 hl;
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_lsec
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_lsec 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
-> 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_DsGetNCChangesCtrl(r, ndr->flags)+55)] uint32
-> __ndr_size;
515 |         drsuapi_DsReplicaObjectListItemEx *first_object;
516 |         uint32 unknown4;
517 |     } drsuapi_DsGetNCChangesCtrl;
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_lsec 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_DsGetNCChangesCtrl6(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_DsGetNCChangesCtrl *ctrl;
563 |     } drsuapi_DsGetNCChangesMSZIPCtrl;
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_DsGetNCChangesXPRESSCtrl;
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_DsGetNCChangesMSZIPCtrl mszip1;
596 |         [case(6)|(DRSUAPI_COMPRESSION_TYPE_MSZIP<<16))]
-> | drsuapi_DsGetNCChangesMSZIPCtr6 mszip6;
597 |         [case(1)|(DRSUAPI_COMPRESSION_TYPE_XPRESS<<16))]
-> | drsuapi_DsGetNCChangesXPRESSCtrl 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 knows
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_DsGetNCChangesCtrl 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 [vl_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_DsGetMembershipsCtrl;

```

```

713 |
714 |     typedef [switch_type(int32)] union {
715 |         [case(1)] drsuapi_DsGetMembershipsCtrl ctrl;
716 |     } drsuapi_DsGetMembershipsCtrl;
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_DsGetMembershipsCtrl 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 [vl_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 [vl_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 [vl_enum] enum {
803         DRSUAPI_DS_NAME_FORMAT_UNKNOWN           = 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_DsNameCtrl;
844
845     typedef [switch_type(int32)] union {
846         [case(1)] drsuapi_DsNameCtrl *ctrl;
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 [vl_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; /* 0x00000001 */
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_DsGetDCInfoCtrl;
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_DsGetDCInfoCtrl2;
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 [vl_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_DsGetDCInfoCtrl ctrl1;
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 "serverReference" 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 /*[noprnt]*/ 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 [vl_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 ul; /* session number? */
1360 |         uint32 u2;
1361 |         uint32 u3;
1362 |         GUID bind_guid;
1363 |         NTTIME_lsec bind_time;
1364 |         [flag(NDR_BIG_ENDIAN)] ipv4address client_ip_address;
1365 |         uint32 u5; /* this is the same value the client used as ul 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_DsGetMembershipsCtrl **ctrl_array;
1427|     } drsuapi_DsGetMemberships2Ctrl;
1428|
1429|     typedef [switch_type(int32)] union {
1430|         [case(1)] drsuapi_DsGetMembershipsCtrl ctrl;
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 repsFromTol *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:

```

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 #ifndef _DSDB_SCHEMA_H
24 #define _DSDB_SCHEMA_H
25
26 struct dsdb_attribute;
27 struct dsdb_class;
28 struct dsdb_schema;
29
30 struct dsdb_syntax {
31     const char *name;
32     const char *ldap_oid;
33     uint32_t oMSyntax;
34     struct ldb_val oObjectClass;
35     const char *attributeSyntax_oid;
36
37     WERROR (*drsuapi_to_ldb)(const struct dsdb_schema *schema,
38                             const struct dsdb_attribute *attr,
39                             const struct drsuapi_DsReplicaAttribute *in,
40                             TALLOC_CTX *mem_ctx,
41                             struct ldb_message_element *out);
42     WERROR (*ldb_to_drsuapi)(const struct dsdb_schema *schema,
43                             const struct dsdb_attribute *attr,
44                             const struct ldb_message_element *in,
45                             TALLOC_CTX *mem_ctx,
46                             struct drsuapi_DsReplicaAttribute *out);
47 };
48
49 struct dsdb_attribute {
50     struct dsdb_attribute *prev, *next;
51
52     const char *cn;
53     const char *LDAPDisplayName;
54     const char *attributeID_oid;
55     uint32_t attributeID_id;
56     struct GUID schemaIDGUID;
57     uint32_t mAPIID;
58
59     struct GUID attributeSecurityGUID;
60
61     uint32_t searchFlags;
62     uint32_t systemFlags;
63     BOOL isMemberOfPartialAttributeSet;
64     uint32_t linkID;
65
66     const char *attributeSyntax_oid;
67     uint32_t attributeSyntax_id;
68     uint32_t oMSyntax;
69     struct ldb_val oObjectClass;
70
71     BOOL isSingleValued;
72     uint32_t rangeLower;
73     uint32_t rangeUpper;
74     BOOL extendedCharsAllowed;
75
76     uint32_t schemaFlagsEx;
77     struct ldb_val msDs_Schema_Extensions;
78
79     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:

```

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 #ifndef _LIBNET_BECOME_DC_H
22 #define _LIBNET_BECOME_DC_H
23
24 #include "librpc/gen_ndr/drsuapi.h"
25
26 struct libnet_BecomeDC_Domain {
27     /* input */
28     const char *dns_name;
29     const char *netbios_name;
30     const struct dom_sid *sid;
31
32     /* constructed */
33     struct GUID guid;
34     const char *dn_str;
35     uint32_t behavior_version;
36     uint32_t w2k3_update_revision;
37 };
38
39 struct libnet_BecomeDC_Forest {
40     /* constructed */
41     const char *dns_name;
42     const char *root_dn_str;
43     const char *config_dn_str;
44     uint32_t crossref_behavior_version;
45     const char *schema_dn_str;
46     uint32_t schema_object_version;
47 };
48
49 struct libnet_BecomeDC_SourceDSA {
50     /* input */
51     const char *address;
52
53     /* constructed */
54     const char *dns_name;
55     const char *netbios_name;
56     const char *site_name;
57     const char *server_dn_str;
58     const char *ntds_dn_str;
59 };
60
61 struct libnet_BecomeDC_CheckOptions {
62     const struct libnet_BecomeDC_Domain *domain;
63     const struct libnet_BecomeDC_Forest *forest;
64     const struct libnet_BecomeDC_SourceDSA *source_dsa;
65 };
66
67 struct libnet_BecomeDC_DestDSA {
68     /* input */
69     const char *netbios_name;
70
71     /* constructed */
72     const char *dns_name;
73     const char *site_name;
74     struct GUID site_guid;
75     const char *computer_dn_str;
76     const char *server_dn_str;
77     const char *ntds_dn_str;
78     struct GUID ntds_guid;
79     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, &msg) == -1) {
83 |         d_printf("smbscript: ejsEvalScript(): %s\n", msg);
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            "SAMBDB[%s] SECRETS[%s] KEYTAB[%s]\n",
167            lp_private_dir(),
168            s->path.sambdb_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(\\\"rootse\\\",\\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.sambdb = \\\"%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.sambdb_ldb,           /* paths.sambdb */
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 repsFromTol *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 repsFromTol);
309 |     NT_STATUS_HAVE_NO_MEMORY(s_dsa);
310 |     s_dsa->other_info = talloc(s_dsa, struct repsFromTolOtherInfo);
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 = malloc_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 = malloc_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 = malloc_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,
->
454 |             objs->objects[i].meta_data);
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,
->
462 |         &schemaInfo_val);
463 |         if (!W_ERROR_IS_OK(status)) {
464 |             DEBUG(0,("Failed dsdb_get_oid_mappings_ldb(%s)\n", win_errstr(status)));
465 |             return werror_to_ntstatus(status);
466 |         }
467 |
468 |         /* we only add prefixMap here, because schemaInfo is a replicated attribute and
->
469 |         already applied */
470 |         ret = ldb_msg_add_value(msg, "prefixMap", &prefixMap_val, &prefixMap_el);
471 |         if (ret != LDB_SUCCESS) {
472 |             return NT_STATUS_FOOBAR;
473 |         }
474 |         prefixMap_el->flags = LDB_FLAG_MOD_REPLACE;
475 |
476 |         ret = ldb_modify(s->ldb, msg);
477 |         if (ret != LDB_SUCCESS) {
478 |             DEBUG(0,("Failed to add prefixMap and schemaInfo %s\n",
->
479 |             ldb_strerror(ret)));
480 |             return NT_STATUS_FOOBAR;
481 |         }
482 |
483 |         talloc_free(s_dsa);
484 |         talloc_free(objs);
485 |
486 |         /* reopen the ldb */
487 |         talloc_free(s->ldb); /* this also free's the s->schema, because dsdb_set_schema()
->
488 |         steals it */
489 |         s->schema = NULL;
490 |
491 |         DEBUG(0,("Reopen the SAM LDB with system credentials and a already stored schema:
->
492 |         %s\n", s->path.samdb_ldb));
493 |         s->ldb = ldb_wrap_connect(s, s->path.samdb_ldb,
494 |                                 system_session(s),
495 |                                 NULL, 0, NULL);
496 |
497 |         if (!s->ldb) {
498 |             DEBUG(0,("Failed to open '%s'\n",

```



```

660 |                                     mapping_ctr,
661 |                                     object_count,
662 |                                     first_object,
663 |                                     linked_attributes_count,
664 |                                     linked_attributes,
665 |                                     s_dsa,
666 |                                     uptodateness_vector,
667 |                                     c->genssec_skey,
668 |                                     s, &objs);
669 |
670 |     if (!W_ERROR_IS_OK(status)) {
671 |         DEBUG(0,("Failed to commit objects: %s\n", win_errstr(status)));
672 |         return werror_to_ntstatus(status);
673 |     }
674 |
675 |     if (lp_parm_bool(-1, "become dc", "dump objects", False)) {
676 |         for (i=0; i < objs->num_objects; i++) {
677 |             struct ldb_ldif ldif;
678 |             fprintf(stdout, "#\n");
679 |             ldif.changetype = LDB_CHANGETYPE_NONE;
680 |             ldif.msg = objs->objects[i].msg;
681 |             ldb_ldif_write_file(s->ldb, stdout, &ldif);
682 |             NDR_PRINT_DEBUG(replPropertyMetaDataTable,
-> | objs->objects[i].meta_data);
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(drstuapi_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 |
717 | BOOL torture_net_become_dc(struct torture_context *torture)
718 | {
719 |     BOOL ret = True;
720 |     NTSTATUS status;
721 |     struct libnet_BecomeDC b;
722 |     struct libnet_UnbecomeDC u;
723 |     struct test_become_dc_state *s;
724 |     struct ldb_message *msg;
725 |     int ldb_ret;
726 |     uint32_t i;
727 |
728 |     s = talloc_zero(torture, struct test_become_dc_state);
729 |     if (!s) return False;
730 |
731 |     s->netbios_name = lp_parm_string(-1, "become dc", "smbtorture dc");
732 |     if (!s->netbios_name || !s->netbios_name[0]) {
733 |         s->netbios_name = "smbtorturedc";
734 |     }
735 |
736 |     s->path.samdb_ldb = talloc_asprintf(s, "%s_samdb.ldb", s->netbios_name);
737 |     if (!s->path.samdb_ldb) return false;
738 |     s->path.domaindn_ldb = talloc_asprintf(s, "%s_domain.ldb", s->netbios_name);
739 |     if (!s->path.domaindn_ldb) return false;
740 |     s->path.configdn_ldb = talloc_asprintf(s, "%s_config.ldb", s->netbios_name);
741 |     if (!s->path.configdn_ldb) return false;
742 |     s->path.schemadn_ldb = talloc_asprintf(s, "%s_schema.ldb", s->netbios_name);
743 |     if (!s->path.schemadn_ldb) return false;
744 |     s->path.secrets_ldb = talloc_asprintf(s, "%s_secrets.ldb", s->netbios_name);
745 |     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_ldapl() and becomeDC_ldap_connect()
57  */
58
59 /*
60  * LDAP search 1st LDAP connection:
61  *
62  * see: becomeDC_ldapl_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 | *
84 | *          supportedLDAPPolicies:    ...
85 | *          highestCommittedUSN:      ...
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 | *          isSynchronized:           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=2fbac1870adell1d297c400c04fd8d5cd,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_accou
-> 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 | *   userAccountControl:   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_ldapl_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_ldapl_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_drсуapi_connect_send(), becomeDC_drсуapi1_connect_recv(),
472 | *     becomeDC_drсуapi_bind_send(), becomeDC_drсуapi_bind_recv() and
-> becomeDC_drсуapi1_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_drсуapi1_add_entry_send() and becomeDC_drсуapi1_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_drсуapi_connect_send(), becomeDC_drсуapi2_connect_recv(),
495 | *     becomeDC_drсуapi_bind_send(), becomeDC_drсуapi_bind_recv(),
-> becomeDC_drсуapi2_bind_recv()
496 | *     and becomeDC_drсуapi3_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_drstuapi_pull_partition_send(), becomeDC_drstuapi_pull_partition_rcv(),
504 *      becomeDC_drstuapi3_pull_schema_send() and becomeDC_drstuapi3_pull_schema_rcv()
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_drstuapi_pull_partition_send(), becomeDC_drstuapi_pull_partition_rcv(),
516 *      becomeDC_drstuapi3_pull_config_send() and becomeDC_drstuapi3_pull_config_rcv()
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_accou
-> 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 *   userAccountControl: 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 *   userAccountControl: 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 *   userAccountControl: 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_drstuapi_pull_partition_send(), becomeDC_drstuapi_pull_partition_rcv(),
635 | *   becomeDC_drstuapi3_pull_domain_send() and becomeDC_drstuapi3_pull_domain_rcv()
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           : 0x0000000ae (174)
647 | *             __ndr_size_sid       : 0x000000000 (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                 : 0x00000001c (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_drstuapi_update_refs_send(), becomeDC_drstuapi2_update_refs_schema_rcv(),
-> |
666 | *   becomeDC_drstuapi2_update_refs_config_rcv() and becomeDC_drstuapi2_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_drstuapi {
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->ldapl.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->ldapl.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->ldapl.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_ldapl_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->ldapl.ldb, s->forest.config_dn_str);
860 |     NT_STATUS_HAVE_NO_MEMORY(basedn);
861 |
862 |     ret = ldb_search(s->ldapl.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_ldapl_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->ldapl.ldb, s->domain.dn_str);
889 |     NT_STATUS_HAVE_NO_MEMORY(basedn);
890 |
891 |     ret = ldb_search(s->ldapl.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_ldapl_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_ldapl_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->ldapl.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->ldapl.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_ldapl_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->ldapl.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->ldapl.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_ldapl_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_ldapl_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->ldapl.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->ldapl.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_ldapl_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->ldapl.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->ldapl.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->ldapl.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_drstuapi_connect_send(struct libnet_BecomeDC_state *s,
1446 |                                           struct becomeDC_drstuapi *drstuapi,
1447 |                                           void (*recv_fn)(struct composite_context
->
1448 | *req));
1449 | static void becomeDC_drstuapi1_connect_recv(struct composite_context *req);
1450 | static void becomeDC_connect_ldap2(struct libnet_BecomeDC_state *s);
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_drstuapi_connect_send(s, &s->drstuapi1, becomeDC_drstuapi1_connect_recv);

```

```

1501 | }
1502 |
1503 | static void becomeDC_drstuapi_connect_send(struct libnet_BecomeDC_state *s,
1504 |                                           struct becomeDC_drstuapi *drstuapi,
1505 |                                           void (*recv_fn)(struct composite_context
->
1506 | {
1507 |     struct composite_context *c = s->creq;
1508 |     struct composite_context *creq;
1509 |     char *binding_str;
1510 |
1511 |     drstuapi->s = s;
1512 |
1513 |     if (!drstuapi->binding) {
1514 |         if (lp_parm_bool(-1, "become_dc", "print", False)) {
1515 |             binding_str = talloc_asprintf(s, "ncacn_ip_tcp:%s[krb5,print,seal
->
1516 | ]", s->source_dsa.dns_name);
1517 |             if (composite_nomem(binding_str, c)) return;
1518 |         } else {
1519 |             binding_str = talloc_asprintf(s, "ncacn_ip_tcp:%s[krb5,seal]",
->
1520 | s->source_dsa.dns_name);
1521 |             if (composite_nomem(binding_str, c)) return;
1522 |         }
1523 |         c->status = dcerpc_parse_binding(s, binding_str, &drstuapi->binding);
1524 |         talloc_free(binding_str);
1525 |         if (!composite_is_ok(c)) return;
1526 |     }
1527 |
1528 |     creq = dcerpc_pipe_connect_b_send(s, drstuapi->binding, &dcerpc_table_drstuapi,
1529 |                                     s->libnet->cred, s->libnet->event_ctx);
1530 |     composite_continue(c, creq, recv_fn, s);
1531 | }
1532 |
1533 | static void becomeDC_drstuapi_bind_send(struct libnet_BecomeDC_state *s,
1534 |                                         struct becomeDC_drstuapi *drstuapi,
1535 |                                         void (*recv_fn)(struct rpc_request *req));
1536 | static void becomeDC_drstuapi_bind_recv(struct rpc_request *req);
1537 |
1538 | static void becomeDC_drstuapi_connect_recv(struct composite_context *req)
1539 | {
1540 |     struct libnet_BecomeDC_state *s = talloc_get_type(req->async.private_data,
1541 |                                                       struct libnet_BecomeDC_state);
1542 |     struct composite_context *c = s->creq;
1543 |
1544 |     c->status = dcerpc_pipe_connect_b_recv(req, s, &s->drstuapi.pipe);
1545 |     if (!composite_is_ok(c)) return;
1546 |
1547 |     c->status = gensec_session_key(s->drstuapi.pipe->conn->security_state.generic_sta
->
1548 | te,
1549 |                                     &s->drstuapi.gensec_skey);
1550 |     if (!composite_is_ok(c)) return;
1551 |
1552 |     becomeDC_drstuapi_bind_send(s, &s->drstuapi, becomeDC_drstuapi_bind_recv);
1553 | }
1554 |
1555 | static void becomeDC_drstuapi_bind_send(struct libnet_BecomeDC_state *s,
1556 |                                         struct becomeDC_drstuapi *drstuapi,
1557 |                                         void (*recv_fn)(struct rpc_request *req))
1558 | {
1559 |     struct composite_context *c = s->creq;
1560 |     struct rpc_request *req;
1561 |     struct drstuapi_DsBindInfo28 *bind_info28;
1562 |
1563 |     GUID_from_string(DRSUAPI_DS_BIND_GUID_W2K3, &drstuapi->bind_guid);
1564 |
1565 |     bind_info28
1566 |         = &drstuapi->local_info28;
1567 |     bind_info28->supported_extensions
1568 |         |= DRSUAPI_SUPPORTED_EXTENSION_BASE;
1569 |     bind_info28->supported_extensions
1570 |         |= DRSUAPI_SUPPORTED_EXTENSION_ASYNC_REPL
->
1571 | ICATION;
1572 |     bind_info28->supported_extensions
1573 |         |= DRSUAPI_SUPPORTED_EXTENSION_REMOVEAPI;
1574 |     bind_info28->supported_extensions
1575 |         |= DRSUAPI_SUPPORTED_EXTENSION_MOVEREQ_V2
->
1576 | ;
1577 |     bind_info28->supported_extensions
1578 |         |= DRSUAPI_SUPPORTED_EXTENSION_GETCHG_COM
->
1579 | PRESS;
1580 |     bind_info28->supported_extensions
1581 |         |= DRSUAPI_SUPPORTED_EXTENSION_DCINFO_V1;
1582 |     bind_info28->supported_extensions
1583 |         |= DRSUAPI_SUPPORTED_EXTENSION_RESTORE_US
->
1584 | N_OPTIMIZATION;
1585 |     bind_info28->supported_extensions
1586 |         |= DRSUAPI_SUPPORTED_EXTENSION_KCC_EXECUT
->
1587 | E;
1588 |     bind_info28->supported_extensions
1589 |         |= DRSUAPI_SUPPORTED_EXTENSION_ADDENTRY_V
->
1590 | 2;
1591 |     if (s->domain.behavior_version == 2) {
1592 |         /* TODO: find out how this is really triggered! */
1593 |         bind_info28->supported_extensions
1594 |             |=
->
1595 | 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_CRYPT_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->ul                      = 528;
1602 |     } else {
1603 |         bind_info28->ul                      = 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_rcv(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.ul                          = info24->ul;
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_drсуапil_add_entry_send(struct libnet_BecomeDC_state *s);
1647
1648 static void becomeDC_drсуапil_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(drсуапil_DsBind, &s->drсуапil.bind_r);
1666     }
1667
1668     status = becomeDC_drсуапil_bind_recv(s, &s->drсуапil);
1669     if (!W_ERROR_IS_OK(status)) {
1670         composite_error(c, werror_to_ntstatus(status));
1671         return;
1672     }
1673
1674     becomeDC_drсуапil_add_entry_send(s);
1675 }
1676
1677 static void becomeDC_drсуапil_add_entry_recv(struct rpc_request *req);
1678
1679 static void becomeDC_drсуапil_add_entry_send(struct libnet_BecomeDC_state *s)
1680 {
1681     struct composite_context *c = s->creq;
1682     struct rpc_request *req;
1683     struct drсуапil_DsAddEntry *r;
1684     struct drсуапil_DsReplicaObjectIdentifier *identifier;
1685     uint32_t num_attrs, i = 0;
1686     struct drсуапil_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 drсуапil_DsAddEntry);
1703     if (composite_nomem(r, c)) return;
1704
1705     /* setup identifier */
1706     identifier = talloc(r, struct drсуапil_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 drсуапil_DsReplicaAttribute, num_attrs);
1717     if (composite_nomem(attrs, c)) return;
1718
1719     /* ntSecurityDescriptor */
1720     {
1721         struct drсуапil_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
1841                                         ReplicaObjectIdentifier3);
1842         if (!composite_is_ok(c)) return;
1843
1844         vs[0].blob          = &vd[0];
1845
1846         attrs[i].attid      = DRSUAPI_ATTRIBUTE_objectCategory;
1847         attrs[i].value_ctr.num_values = 1;
1848         attrs[i].value_ctr.values = vs;
1849
1850         i++;
1851     }
1852
1853     /* invocationId: random guid */
1854     {
1855         struct drsuapi_DsAttributeValue *vs;
1856         DATA_BLOB *vd;
1857         const struct GUID *v;
1858
1859         vs = talloc_array(attrs, struct drsuapi_DsAttributeValue, 1);
1860         if (composite_nomem(vs, c)) return;
1861
1862         vd = talloc_array(vs, DATA_BLOB, 1);
1863         if (composite_nomem(vd, c)) return;
1864
1865         v = &s->dest_dsa.invocation_id;
1866
1867         c->status = ndr_push_struct_blob(&vd[0], vd, v,
1868                                         (ndr_push_flags_fn_t) ndr_push_GUID);
1869         if (!composite_is_ok(c)) return;
1870
1871         vs[0].blob          = &vd[0];
1872
1873         attrs[i].attid      = DRSUAPI_ATTRIBUTE_invocationId;
1874         attrs[i].value_ctr.num_values = 1;
1875         attrs[i].value_ctr.values = vs;
1876
1877         i++;
1878     }
1879
1880     /* hasMasterNCs: ... */
1881     {
1882         struct drsuapi_DsAttributeValue *vs;
1883         DATA_BLOB *vd;
1884         struct drsuapi_DsReplicaObjectIdentifier3 v[3];
1885
1886         vs = talloc_array(attrs, struct drsuapi_DsAttributeValue, 3);
1887         if (composite_nomem(vs, c)) return;
1888
1889         vd = talloc_array(vs, DATA_BLOB, 3);
1890         if (composite_nomem(vd, c)) return;
1891
1892         v[0].guid           = GUID_zero();
1893         v[0].sid            = s->zero_sid;
1894         v[0].dn             = s->forest.config_dn_str;
1895
1896         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_drstuapi_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_drstuapi_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_drstuapi_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_drstuapi_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_drstuapi_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_drstuapi_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->drsuapil.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.unknownl = 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->drsuapil.pipe, r, r);
2122 |     composite_continue_rpc(c, req, becomeDC_drsuapil_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_drsuapil_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
->
2196 |                 s));
2197 |                 return;
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_rcv);
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_drstuapi2_bind_recv(struct rpc_request *req);
2241 |
2242 | static void becomeDC_drstuapi2_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->drstuapi2.pipe);
2249 |     if (!composite_is_ok(c)) return;
2250 |
2251 |     c->status = gensec_session_key(s->drstuapi2.pipe->conn->security_state.generic_sta
->
2252 | te, &s->drstuapi2.gensec_skey);
2253 |     if (!composite_is_ok(c)) return;
2254 |
2255 |     becomeDC_drstuapi_bind_send(s, &s->drstuapi2, becomeDC_drstuapi2_bind_recv);
2256 | }
2257 |
2258 | static void becomeDC_drstuapi3_connect_recv(struct composite_context *req);
2259 |
2260 | static void becomeDC_drstuapi2_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(drstuapi_DsBind, &s->drstuapi2.bind_r);
2279 |     }
2280 |
2281 |     status = becomeDC_drstuapi_bind_recv(s, &s->drstuapi2);
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->drstuapi1.binding);
2289 |     if (composite_nomem(binding_str, c)) return;
2290 |
2291 |     c->status = dcerpc_parse_binding(s, binding_str, &s->drstuapi3.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->drstuapi3.binding->assoc_group_id = s->drstuapi1.pipe->assoc_group_id;
2297 |     /* w2k3 uses the concurrent multiplex feature on the 3rd connection, so we do */
2298 |     s->drstuapi3.binding->flags          |= DCERPC_CONCURRENT_MULTIPLEX;
2299 |
2300 |     becomeDC_drstuapi_connect_send(s, &s->drstuapi3, becomeDC_drstuapi3_connect_recv);
2301 | }
2302 |
2303 | static void becomeDC_drstuapi3_pull_schema_send(struct libnet_BecomeDC_state *s);
2304 |
2305 | static void becomeDC_drstuapi3_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->drstuapi3.pipe);
2312 |     if (!composite_is_ok(c)) return;
2313 |
2314 |     c->status = gensec_session_key(s->drstuapi3.pipe->conn->security_state.generic_sta

```

```

-> te,
2315 |                                     &s->drsuapi3.gensec_skey);
2316 |     if (!composite_is_ok(c)) return;
2317 |
2318 |     becomeDC_drсуapi3_pull_schema_send(s);
2319 | }
2320 |
2321 | static void becomeDC_drсуapi_pull_partition_send(struct libnet_BecomeDC_state *s,
2322 |                                                 struct becomeDC_drсуapi *drsuapi_h,
2323 |                                                 struct becomeDC_drсуapi *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.hl = 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.hl = 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_drсуapi_DsGetNCChanges_send(drsuapi_p->pipe, r, r);
2377 |     composite_continue_rpc(c, req, recv_fn, s);
2378 | }
2379 |
2380 | static WERROR becomeDC_drсуapi_pull_partition_recv(struct libnet_BecomeDC_state *s,
2381 |                                                  struct becomeDC_drсуapi *drsuapi_h,
2382 |                                                  struct becomeDC_drсуapi *drsuapi_p,
2383 |                                                  struct libnet_BecomeDC_Partition
-> *partition,
2384 |                                                  struct drsuapi_DsGetNCChanges *r)
2385 | {
2386 |     uint32_t ctr_level = 0;
2387 |     struct drsuapi_DsGetNCChangesCtrl1 *ctrl1 = NULL;
2388 |     struct drsuapi_DsGetNCChangesCtrl6 *ctrl6 = NULL;
2389 |     struct GUID *source_dsa_guid;
2390 |     struct GUID *source_dsa_invocation_id;
2391 |     struct drsuapi_DsReplicaHighWaterMark *new_highwatermark;
2392 |     NTSTATUS nt_status;

```



```

2478 | }
2479 |
2480 | static void becomeDC_drсуapi3_pull_config_send(struct libnet_BecomeDC_state *s);
2481 |
2482 | static void becomeDC_drсуapi3_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_drсуapi_pull_partition_recv(s, &s->drsuapi2, &s->drsuapi3,
->
2505 | &s->schema_part, r);
2506 |     if (!W_ERROR_IS_OK(status)) {
2507 |         composite_error(c, werror_to_ntstatus(status));
2508 |         return;
2509 |     }
2510 |     talloc_free(r);
2511 |
2512 |     if (s->schema_part.highwatermark.tmp_highest_usn >
->
2513 | s->schema_part.highwatermark.highest_usn) {
2514 |         becomeDC_drсуapi_pull_partition_send(s, &s->drsuapi2, &s->drsuapi3,
->
2515 | &s->schema_part,
2516 |                                             becomeDC_drсуapi3_pull_schema_recv);
2517 |     }
2518 |     return;
2519 | }
2520 |
2521 | static void becomeDC_drсуapi3_pull_config_recv(struct rpc_request *req);
2522 |
2523 | static void becomeDC_drсуapi3_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_drсуapi_pull_partition_send(s, &s->drsuapi2, &s->drsuapi3,
->
2541 | &s->config_part,
2542 |                                             becomeDC_drсуapi3_pull_config_recv);
2543 | }
2544 |
2545 | static void becomeDC_drсуapi3_pull_config_recv(struct rpc_request *req)
2546 | {
2547 |     struct libnet_BecomeDC_state *s = talloc_get_type(req->async.private,
2548 |                                                       struct libnet_BecomeDC_state);
2549 |     struct composite_context *c = s->creq;
2550 |     struct drsuapi_DsGetNCChanges *r = talloc_get_type(req->ndr.struct_ptr,
2551 |                                                       struct drsuapi_DsGetNCChanges);
2552 |     WERROR status;
2553 |
2554 |     bool print = false;
2555 |
2556 |     if (req->p->conn->flags & DCERPC_DEBUG_PRINT_OUT) {
2557 |         print = true;
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(drstuapi_DsGetNCChanges, r);
2564 |         }
2565 |
2566 |         status = becomeDC_drstuapi_pull_partition_recv(s, &s->drstuapi2, &s->drstuapi3,
-> &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_drstuapi_pull_partition_send(s, &s->drstuapi2, &s->drstuapi3,
-> &s->config_part,
->
2576 |
->
2577 |             becomeDC_drstuapi3_pull_config_recv);
2578 |         }
2579 |
2580 |         becomeDC_connect_ldap2(s);
2581 |     }
2582 |
2583 | static void becomeDC_drstuapi3_pull_domain_recv(struct rpc_request *req);
2584 |
2585 | static void becomeDC_drstuapi3_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->drstuapi2.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_drstuapi_pull_partition_send(s, &s->drstuapi2, &s->drstuapi3,
-> &s->domain_part,
2603 |
->
2604 |         becomeDC_drstuapi3_pull_domain_recv);
2605 | }
2606 |
2607 | static void becomeDC_drstuapi_update_refs_send(struct libnet_BecomeDC_state *s,
2608 | struct becomeDC_drstuapi *drstuapi,
2609 | struct libnet_BecomeDC_Partition
-> *partition,
2610 | void (*recv_fn)(struct rpc_request *req));
2611 |
2612 | static void becomeDC_drstuapi2_update_refs_schema_recv(struct rpc_request *req);
2613 |
2614 | static void becomeDC_drstuapi3_pull_domain_recv(struct rpc_request *req)
2615 | {
2616 |     struct libnet_BecomeDC_state *s = talloc_get_type(req->async.private,
2617 | struct libnet_BecomeDC_state);
2618 |     struct composite_context *c = s->creq;
2619 |     struct drstuapi_DsGetNCChanges *r = talloc_get_type(req->ndr.struct_ptr,
2620 | struct drstuapi_DsGetNCChanges);
2621 |
2622 |     WERROR status;
2623 |     bool print = false;
2624 |
2625 |     if (req->p->conn->flags & DCERPC_DEBUG_PRINT_OUT) {
2626 |         print = true;
2627 |     }
2628 |
2629 |     c->status = dcerpc_ndr_request_recv(req);
2630 |     if (!composite_is_ok(c)) return;
2631 |
2632 |     if (print) {
2633 |         NDR_PRINT_OUT_DEBUG(drstuapi_DsGetNCChanges, r);
2634 |     }
2635 |
2636 |     status = becomeDC_drstuapi_pull_partition_recv(s, &s->drstuapi2, &s->drstuapi3,
-> &s->domain_part, r);
2637 |     if (!W_ERROR_IS_OK(status)) {
2638 |         composite_error(c, werror_to_ntstatus(status));
2639 |         return;
2640 |     }

```

```

2638 |
2639 |     talloc_free(r);
2640 |
2641 |     if (s->domain_part.highwatermark.tmp_highest_usn >
-> s->domain_part.highwatermark.highest_usn) {
2642 |         becomeDC_drstuapi_pull_partition_send(s, &s->drstuapi2, &s->drstuapi3,
-> &s->domain_part,
2643 |                                             becomeDC_drstuapi3_pull_domain_recv);
->
2644 |         return;
2645 |     }
2646 |
2647 |     becomeDC_drstuapi_update_refs_send(s, &s->drstuapi2, &s->schema_part,
2648 |                                       becomeDC_drstuapi2_update_refs_schema_recv);
2649 | }
2650 |
2651 | static void becomeDC_drstuapi_update_refs_send(struct libnet_BecomeDC_state *s,
2652 |                                               struct becomeDC_drstuapi *drstuapi,
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 drstuapi_DsReplicaUpdateRefs *r;
2659 |     const char *ntds_guid_str;
2660 |     const char *ntds_dns_name;
2661 |
2662 |     r = talloc(s, struct drstuapi_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      = &drstuapi->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_drstuapi_DsReplicaUpdateRefs_send(drstuapi->pipe, r, r);
2683 |     composite_continue_rpc(c, req, recv_fn, s);
2684 | }
2685 |
2686 | static void becomeDC_drstuapi2_update_refs_config_recv(struct rpc_request *req);
2687 |
2688 | static void becomeDC_drstuapi2_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 drstuapi_DsReplicaUpdateRefs *r = talloc_get_type(req->ndr.struct_ptr,
2694 |                                                               struct drstuapi_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(drstuapi_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_drstuapi_update_refs_send(s, &s->drstuapi2, &s->config_part,
2716 |                                       becomeDC_drstuapi2_update_refs_config_recv);
2717 | }
2718 |
2719 | static void becomeDC_drstuapi2_update_refs_domain_recv(struct rpc_request *req);
2720 |

```

```

2721 static void becomeDC_drсуapi2_update_refs_config_rcv(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_rcv(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_drсуapi_update_refs_send(s, &s->drсуapi2, &s->domain_part,
2740     becomeDC_drсуapi2_update_refs_domain_rcv);
2741 }
2742
2743 static void becomeDC_drсуapi2_update_refs_domain_rcv(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_rcv(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 *_l_l_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=*)", _l_l_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<lc> 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_accou

```

```

->   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   *       userAccountControl:   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  *       userAccountControl:   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 *       userAccountControl:   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_drstuapi_connect_send(), unbecomeDC_drstuapi_connect_recv(),
169 | *       unbecomeDC_drstuapi_bind_send() and unbecomeDC_drstuapi_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_drstuapi_remove_ds_server_send() and unbecomeDC_drstuapi_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 drstuapi_DsBind bind_r;
207 |         struct GUID bind_guid;
208 |         struct drstuapi_DsBindInfoCtr bind_info_ctr;
209 |         struct drstuapi_DsBindInfo28 local_info28;
210 |         struct drstuapi_DsBindInfo28 remote_info28;
211 |         struct policy_handle bind_handle;
212 |         struct drstuapi_DsRemoveDSserver rm_ds_srv_r;
213 |     } drstuapi;
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 |         ret = ldb_search(s->ldap.ldb, basedn, LDB_SCOPE_BASE,
337 |             "(objectClass=*)", attrs, &r);
338 |         talloc_free(basedn);
339 |         if (ret != LDB_SUCCESS) {
340 |             return NT_STATUS_LDAP(ret);
341 |         } else if (r->count != 1) {
342 |             talloc_free(r);
343 |             return NT_STATUS_INVALID_NETWORK_RESPONSE;
344 |         }
345 |         talloc_steal(s, r);
346 |         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 *_l_l_attrs[] = {
454 |         "1.1",
455 |         NULL
456 |     };
457 |
458 |     basedn = ldb_dn_new_fmt(s, s->ldap.ldb, "<WKGUID=aa312825768811d1laded00c04fd8d5cd
->,%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=*)", _l_l_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_drstuapi_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_drstuapi_connect_send(s);
524 | }
525 |
526 | static void unbecomeDC_drstuapi_connect_recv(struct composite_context *creq);
527 |
528 | static void unbecomeDC_drstuapi_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->drstuapi.binding);
538 |     talloc_free(binding_str);
539 |     if (!composite_is_ok(c)) return;
540 |
541 |     creq = dcerpc_pipe_connect_b_send(s, s->drstuapi.binding, &dcerpc_table_drstuapi,
542 |                                     s->libnet->cred, s->libnet->event_ctx);
543 |     composite_continue(c, creq, unbecomeDC_drstuapi_connect_recv, s);
544 | }
545 |
546 | static void unbecomeDC_drstuapi_bind_send(struct libnet_UnbecomeDC_state *s);
547 |
548 | static void unbecomeDC_drstuapi_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->drstuapi.pipe);
555 |     if (!composite_is_ok(c)) return;
556 |
557 |     unbecomeDC_drstuapi_bind_send(s);
558 | }
559 |
560 | static void unbecomeDC_drstuapi_bind_recv(struct rpc_request *req);
561 |
562 | static void unbecomeDC_drstuapi_bind_send(struct libnet_UnbecomeDC_state *s)
563 | {
564 |     struct composite_context *c = s->creq;
565 |     struct rpc_request *req;
566 |     struct drstuapi_DsBindInfo28 *bind_info28;
567 |
568 |     GUID_from_string(DRSUAPI_DS_BIND_GUID, &s->drstuapi.bind_guid);
569 |
570 |     bind_info28
571 |     bind_info28->supported_extensions      = &s->drstuapi.local_info28;
572 |     bind_info28->site_guid                 = GUID_zero();
573 |     bind_info28->ul                         = 508;
574 |     bind_info28->repl_epoch                = 0;
575 |
576 |     s->drstuapi.bind_info_ctr.length       = 28;
577 |     s->drstuapi.bind_info_ctr.info.info28 = *bind_info28;
578 |
579 |     s->drstuapi.bind_r.in.bind_guid = &s->drstuapi.bind_guid;
580 |     s->drstuapi.bind_r.in.bind_info = &s->drstuapi.bind_info_ctr;
581 |     s->drstuapi.bind_r.out.bind_handle = &s->drstuapi.bind_handle;
582 |
583 |     req = dcerpc_drstuapi_DsBind_send(s->drstuapi.pipe, s, &s->drstuapi.bind_r);
584 |     composite_continue_rpc(c, req, unbecomeDC_drstuapi_bind_recv, s);
585 | }
586 |
587 | static void unbecomeDC_drstuapi_remove_ds_server_send(struct libnet_UnbecomeDC_state *s);
588 |

```



```

589 static void unbecomeDC_drstuapi_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->drstuapi.bind_r.out.result)) {
599         composite_error(c, werror_to_ntstatus(s->drstuapi.bind_r.out.result));
600         return;
601     }
602
603     ZERO_STRUCT(s->drstuapi.remote_info28);
604     if (s->drstuapi.bind_r.out.bind_info) {
605         switch (s->drstuapi.bind_r.out.bind_info->length) {
606             case 24: {
607                 struct drstuapi_DsBindInfo24 *info24;
608                 info24 = &s->drstuapi.bind_r.out.bind_info->info.info24;
609                 s->drstuapi.remote_info28.supported_extensions =
-> info24->supported_extensions;
610                 s->drstuapi.remote_info28.site_guid =
-> info24->site_guid;
611                 s->drstuapi.remote_info28.u1 = info24->u1;
612                 s->drstuapi.remote_info28.repl_epoch = 0;
613                 break;
614             }
615             case 28:
616                 s->drstuapi.remote_info28 = s->drstuapi.bind_r.out.bind_info->info.
-> info28;
617                 break;
618             }
619         }
620
621         unbecomeDC_drstuapi_remove_ds_server_send(s);
622     }
623
624 static void unbecomeDC_drstuapi_remove_ds_server_recv(struct rpc_request *req);
625
626 static void unbecomeDC_drstuapi_remove_ds_server_send(struct libnet_UnbecomeDC_state *s)
627 {
628     struct composite_context *c = s->creq;
629     struct rpc_request *req;
630     struct drstuapi_DsRemovedSServer *r = &s->drstuapi.rm_ds_srv_r;
631
632     r->in.bind_handle = &s->drstuapi.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_drstuapi_DsRemovedSServer_send(s->drstuapi.pipe, s, r);
639     composite_continue_rpc(c, req, unbecomeDC_drstuapi_remove_ds_server_recv, s);
640 }
641
642 static void unbecomeDC_drstuapi_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 drstuapi_DsRemovedSServer *r = &s->drstuapi.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_drstuapi.h"
29 #include "librpc/gen_ndr/ndr_drstblobs.h"
30
31 WERROR dsdb_load_oid_mappings_drstuapi(struct dsdb_schema *schema, const struct
-> drstuapi_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_strdup(ctr->mappings,
-> 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;

```



```

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_MSIDS_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_MSIDS_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)) {

```

```

403 |                                     DEBUG(0,("%s: '%s': unable to map attributeSyntax_ %s: %s\n",
404 |                                           __location__, attr->LDAPDisplayName,
-> | attr->attributeSyntax_oid,
405 |                                     win_errstr(status)));
406 |                                     return status;
407 |                                 }
408 |                             }
409 |                             GET_UINT32_LDB(msg, "oMSyntax", attr, oMSyntax);
410 |                             GET_BLOB_LDB(msg, "oObjectClass", mem_ctx, attr, oObjectClass);
411 |
412 |                             GET_BOOL_LDB(msg, "isSingleValued", attr, isSingleValued, True);
413 |                             GET_UINT32_LDB(msg, "rangeLower", attr, rangeLower);
414 |                             GET_UINT32_LDB(msg, "rangeUpper", attr, rangeUpper);
415 |                             GET_BOOL_LDB(msg, "extendedCharsAllowed", attr, extendedCharsAllowed, False);
416 |
417 |                             GET_UINT32_LDB(msg, "schemaFlagsEx", attr, schemaFlagsEx);
418 |                             GET_BLOB_LDB(msg, "msDs-Schema-Extensions", mem_ctx, attr,
-> | msDs_Schema_Extensions);
419 |
420 |                             GET_BOOL_LDB(msg, "showInAdvancedViewOnly", attr, showInAdvancedViewOnly,
-> | False);
421 |                             GET_STRING_LDB(msg, "adminDisplayName", mem_ctx, attr, adminDisplayName, False);
422 |                             GET_STRING_LDB(msg, "adminDescription", mem_ctx, attr, adminDescription, False);
423 |                             GET_STRING_LDB(msg, "classDisplayName", mem_ctx, attr, classDisplayName, False);
424 |                             GET_BOOL_LDB(msg, "isEphemeral", attr, isEphemeral, False);
425 |                             GET_BOOL_LDB(msg, "isDefunct", attr, isDefunct, False);
426 |                             GET_BOOL_LDB(msg, "systemOnly", attr, systemOnly, False);
427 |
428 |                             attr->syntax = dsdb_syntax_for_attribute(attr);
429 |                             if (!attr->syntax) {
430 |                                 return WERR_DS_ATT_SCHEMA_REQ_SYNTAX;
431 |                             }
432 |
433 |                             return WERR_OK;
434 |                         }
435 |
436 | WERROR dsdb_class_from_ldb(const struct dsdb_schema *schema,
437 |                           struct ldb_message *msg,
438 |                           TALLOC_CTX *mem_ctx,
439 |                           struct dsdb_class *obj)
440 | {
441 |     WERROR status;
442 |
443 |     GET_STRING_LDB(msg, "cn", mem_ctx, obj, cn, True);
444 |     GET_STRING_LDB(msg, "LDAPDisplayName", mem_ctx, obj, LDAPDisplayName, True);
445 |     GET_STRING_LDB(msg, "governsID", mem_ctx, obj, governsID_oid, True);
446 |     if (schema->num_prefixes == 0) {
447 |         /* set an invalid value */
448 |         obj->governsID_id = 0xFFFFFFFF;
449 |     } else {
450 |         status = dsdb_map_oid2int(schema, obj->governsID_oid,
-> | &obj->governsID_id);
451 |         if (!W_ERROR_IS_OK(status)) {
452 |             DEBUG(0,("%s: '%s': unable to map governsID %s: %s\n",
453 |                   __location__, obj->LDAPDisplayName, obj->governsID_oid,
454 |                   win_errstr(status)));
455 |             return status;
456 |         }
457 |     }
458 |     GET_GUID_LDB(msg, "schemaIDGUID", obj, schemaIDGUID);
459 |
460 |     GET_UINT32_LDB(msg, "objectClassCategory", obj, objectClassCategory);
461 |     GET_STRING_LDB(msg, "rDNAttID", mem_ctx, obj, rDNAttID, False);
462 |     GET_STRING_LDB(msg, "defaultObjectCategory", mem_ctx, obj, defaultObjectCategory,
-> | True);
463 |
464 |     GET_STRING_LDB(msg, "subClassOf", mem_ctx, obj, subClassOf, True);
465 |
466 |     obj->systemAuxiliaryClass      = NULL;
467 |     obj->systemPossSuperiors       = NULL;
468 |     obj->systemMustContain         = NULL;
469 |     obj->systemMayContain          = NULL;
470 |
471 |     obj->auxiliaryClass            = NULL;
472 |     obj->possSuperiors             = NULL;
473 |     obj->mustContain               = NULL;
474 |     obj->mayContain                = NULL;
475 |
476 |     GET_STRING_LDB(msg, "defaultSecurityDescriptor", mem_ctx, obj,
-> | defaultSecurityDescriptor, False);
477 |
478 |     GET_UINT32_LDB(msg, "schemaFlagsEx", obj, schemaFlagsEx);
479 |     GET_BLOB_LDB(msg, "msDs-Schema-Extensions", mem_ctx, obj,
-> | msDs_Schema_Extensions);
480 |
481 |     GET_BOOL_LDB(msg, "showInAdvancedViewOnly", obj, showInAdvancedViewOnly, False);
482 |     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
->
541 | *schema, struct
->
542 | drsuapi_DsReplicaObject *obj, 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, "oObjectClass", mem_ctx, attr, oObjectClass);
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_drstuapi(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_drstuapi.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_drstuapi_to_ldb(const struct dsdb_schema *schema,
31                                                 const struct dsdb_attribute *attr,
32                                                 const struct drstuapi_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_drstuapi(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 drstuapi_DsReplicaAttribute *out)
68 {
69     return WERR_FOOBAR;
70 }
71
72 static WERROR dsdb_syntax_BOOL_drstuapi_to_ldb(const struct dsdb_schema *schema,
73                                                 const struct dsdb_attribute *attr,
74                                                 const struct drstuapi_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_ldap_to_drstuapi(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 drstuapi_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 drstuapi_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_drstuapi_to_ldap(const struct dsdb_schema *schema,
158 |                                                const struct dsdb_attribute *attr,
159 |                                                const struct drstuapi_DsReplicaAttribute
->
160 |                                                *in,
161 |                                                TALLOC_CTX *mem_ctx,
162 |                                                struct ldb_message_element *out)
163 | {
164 |     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_drstuapi(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 drstuapi_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 drstuapi_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_drstuapi_to_ldb(const struct dsdb_schema *schema,
236 |                                               const struct dsdb_attribute *attr,
237 |                                               const struct drstuapi_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_drstuapi(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_drstuapi_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_lsec 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_drstuapi(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_drstuapi_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 *= 1000000;
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_drstuapi(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 /= 1000000;
482 |
483 |         SBVAL(blobs[i].data, 0, v);
484 |     }
485 |
486 |     return WERR_OK;
487 | }
488 |
489 | static WERROR dsdb_syntax_DATA_BLOB_drstuapi_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_drstuapi(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_drstuapi_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_drstuapi_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_drstuapi_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_drstuapi_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_drstuapi_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_drstuapi(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 drstuapi_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 DRSTUAPI_ATTRIBUTE_objectClass:
710 |     case DRSTUAPI_ATTRIBUTE_governsID:
711 |     case DRSTUAPI_ATTRIBUTE_attributeID:
712 |     case DRSTUAPI_ATTRIBUTE_attributeSyntax:
713 |         return dsdb_syntax_FOOBAR_ldb_to_drstuapi(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 drstuapi_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 = strtoul((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_drstuapi_to_ldb(const struct dsdb_schema *schema,
743 |                                                   const struct dsdb_attribute *attr,
744 |                                                   const struct drstuapi_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_drstuapi(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_drstuapi_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_drstuapi_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_drstuapi(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 drstuapi_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 drstuapi_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 drstuapi_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_drstuapi_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_drstuapi_to_ldb(const struct dsdb_schema *schema,
911 |                                                    const struct dsdb_attribute *attr,
912 |                                                    const struct
-> drstuapi_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 drstuapi_DsReplicaObjectIdentifier3Binary id3b;
928 |         char *binary;
929 |         char *str;

```



```

-> drsuapi_DsReplicaAttribute *in,
1011 |
1012 |                                     TALLOC_CTX *mem_ctx,
-> *out)                               struct ldb_message_element
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
1114         = "Boolean",
1115         .ldap_oid
1116         = "1.3.6.1.4.1.1466.115.121.1.7",
1117         .oMSyntax
1118         = 1,
1119         .attributeSyntax_oid
1120         = "2.5.5.8",
1121         .drsuapi_to_ldb
1122         = dsdb_syntax_BOOL_drsuapi_to_ldb,
1123         .ldb_to_drsuapi
1124         = dsdb_syntax_BOOL_ldb_to_drsuapi,
1125     }, {
1126         .name
1127         = "Integer",
1128         .ldap_oid
1129         = "1.3.6.1.4.1.1466.115.121.1.27",
1130         .oMSyntax
1131         = 2,
1132         .attributeSyntax_oid
1133         = "2.5.5.9",
1134         .drsuapi_to_ldb
1135         = dsdb_syntax_INT32_drsuapi_to_ldb,
1136         .ldb_to_drsuapi
1137         = dsdb_syntax_INT32_ldb_to_drsuapi,
1138     }, {
1139         .name
1140         = "String(Octet)",
1141         .ldap_oid
1142         = "1.3.6.1.4.1.1466.115.121.1.40",
1143         .oMSyntax
1144         = 4,
1145         .attributeSyntax_oid
1146         = "2.5.5.10",
1147         .drsuapi_to_ldb
1148         = dsdb_syntax_DATA_BLOB_drsuapi_to_ldb,
1149         .ldb_to_drsuapi
1150         = dsdb_syntax_DATA_BLOB_ldb_to_drsuapi,
1151     }, {
1152         .name
1153         = "String(Sid)",
1154         .ldap_oid
1155         = "1.3.6.1.4.1.1466.115.121.1.40",
1156         .oMSyntax
1157         = 4,
1158         .attributeSyntax_oid
1159         = "2.5.5.17",
1160         .drsuapi_to_ldb
1161         = dsdb_syntax_DATA_BLOB_drsuapi_to_ldb,
1162         .ldb_to_drsuapi
1163         = dsdb_syntax_DATA_BLOB_ldb_to_drsuapi,
1164     }, {
1165         .name
1166         = "String(Object-Identifier)",
1167         .ldap_oid
1168         = "1.3.6.1.4.1.1466.115.121.1.38",
1169         .oMSyntax
1170         = 6,
1171         .attributeSyntax_oid
1172         = "2.5.5.2",
1173         .drsuapi_to_ldb
1174         = dsdb_syntax_OID_drsuapi_to_ldb,
1175         .ldb_to_drsuapi
1176         = dsdb_syntax_OID_ldb_to_drsuapi,
1177     }, {
1178         .name
1179         = "Enumeration",
1180         .ldap_oid
1181         = "1.3.6.1.4.1.1466.115.121.1.27",
1182         .oMSyntax
1183         = 10,
1184         .attributeSyntax_oid
1185         = "2.5.5.9",
1186         .drsuapi_to_ldb
1187         = dsdb_syntax_INT32_drsuapi_to_ldb,
1188         .ldb_to_drsuapi
1189         = dsdb_syntax_INT32_ldb_to_drsuapi,
1190     }, {
1191         /* not used in w2k3 forest */
1192         .name
1193         = "String(Numeric)",
1194         .ldap_oid
1195         = "1.3.6.1.4.1.1466.115.121.1.36",
1196         .oMSyntax
1197         = 18,
1198         .attributeSyntax_oid
1199         = "2.5.5.6",
1200         .drsuapi_to_ldb
1201         = dsdb_syntax_DATA_BLOB_drsuapi_to_ldb,
1202         .ldb_to_drsuapi
1203         = dsdb_syntax_DATA_BLOB_ldb_to_drsuapi,
1204     }, {
1205         .name
1206         = "String(Printable)",
1207         .ldap_oid
1208         = "1.3.6.1.4.1.1466.115.121.1.44",
1209         .oMSyntax
1210         = 19,
1211         .attributeSyntax_oid
1212         = "2.5.5.5",
1213         .drsuapi_to_ldb
1214         = dsdb_syntax_DATA_BLOB_drsuapi_to_ldb,
1215         .ldb_to_drsuapi
1216         = dsdb_syntax_DATA_BLOB_ldb_to_drsuapi,
1217     }, {
1218         .name
1219         = "String(Teletex)",
1220         .ldap_oid
1221         = "1.2.840.113556.1.4.905",
1222         .oMSyntax
1223         = 20,
1224         .attributeSyntax_oid
1225         = "2.5.5.4",
1226         .drsuapi_to_ldb
1227         = dsdb_syntax_DATA_BLOB_drsuapi_to_ldb,
1228         .ldb_to_drsuapi
1229         = dsdb_syntax_DATA_BLOB_ldb_to_drsuapi,
1230     }, {
1231         .name
1232         = "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_drсуapi_to_ldb,
1182 |         .ldb_to_drсуapi    = dsdb_syntax_DATA_BLOB_ldb_to_drсуapi,
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_drсуapi_to_ldb,
1189 |         .ldb_to_drсуapi    = dsdb_syntax_NTTIME_UTC_ldb_to_drсуapi,
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_drсуapi_to_ldb,
1196 |         .ldb_to_drсуapi    = dsdb_syntax_NTTIME_ldb_to_drсуapi,
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_drсуapi_to_ldb,
1204 |         .ldb_to_drсуapi    = dsdb_syntax_FOOBAR_ldb_to_drсуapi,
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_drсуapi_to_ldb,
1211 |         .ldb_to_drсуapi    = dsdb_syntax_UNICODE_ldb_to_drсуapi,
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_drсуapi_to_ldb,
1218 |         .ldb_to_drсуapi    = dsdb_syntax_INT64_ldb_to_drсуapi,
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_drсуapi_to_ldb,
1225 |         .ldb_to_drсуapi    = dsdb_syntax_DATA_BLOB_ldb_to_drсуapi,
1226 |     }, {
1227 |         .name              = "Object(DS-DN)",
1228 |         .ldap_oid          = "1.3.6.1.4.1.1466.115.121.1.12",
1229 |         .oMSyntax         = 127,
1230 |         .oObjectClass      = OMOBJECTCLASS("x2b\x0c\x02\x87\x73\x1c\x00\x85
-> \x4a"),
1231 |         .attributeSyntax_oid = "2.5.5.1",
1232 |         .drsuapi_to_ldb    = dsdb_syntax_DN_drсуapi_to_ldb,
1233 |         .ldb_to_drсуapi    = dsdb_syntax_DN_ldb_to_drсуapi,
1234 |     }, {
1235 |         .name              = "Object(DN-Binary)",
1236 |         .ldap_oid          = "1.2.840.113556.1.4.903",
1237 |         .oMSyntax         = 127,
1238 |         .oObjectClass      = 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_drсуapi_to_ldb,
1241 |         .ldb_to_drсуapi    = dsdb_syntax_DN_BINARY_ldb_to_drсуapi,
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 |         .oObjectClass      = OMOBJECTCLASS("x56\x06\x01\x02\x05\x0b\x1D"),
1248 |         .attributeSyntax_oid = "2.5.5.7",
1249 |         .drsuapi_to_ldb    = dsdb_syntax_FOOBAR_drсуapi_to_ldb,
1250 |         .ldb_to_drсуapi    = dsdb_syntax_FOOBAR_ldb_to_drсуapi,
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 |         .oObjectClass      = 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_drstuapi_to_ldb,
1264 |         .ldb_to_drstuapi         = dsdb_syntax_DATA_BLOB_ldb_to_drstuapi,
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_drstuapi_to_ldb
->
1272 |         .ldb_to_drstuapi         = dsdb_syntax_PRESENTATION_ADDRESS_ldb_to_drstuapi
->
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_drstuapi_to_ldb,
1281 |         .ldb_to_drstuapi         = dsdb_syntax_FOOBAR_ldb_to_drstuapi,
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_drstuapi_to_ldb,
1290 |         .ldb_to_drstuapi         = dsdb_syntax_FOOBAR_ldb_to_drstuapi,
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_oid)
-> d) != 0) continue;
1312
1313         return &dsdb_syntaxes[i];
1314     }
1315
1316     return NULL;
1317 }
1318
1319 WERROR dsdb_attribute_drstuapi_to_ldb(const struct dsdb_schema *schema,
1320                                     const struct drstuapi_DsReplicaAttribute *in,
1321                                     TALLOCTX *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->drstuapi_to_ldb(schema, sa, in, mem_ctx, out);
1332 }
1333
1334 WERROR dsdb_attribute_ldb_to_drstuapi(const struct dsdb_schema *schema,
1335                                     const struct ldb_message_element *in,
1336                                     TALLOCTX *mem_ctx,
1337                                     struct drstuapi_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_drstuapi(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_drstuapi.h"
29 #include "librpc/gen_ndr/ndr_drstblobs.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_key,
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 |
183 |     W_ERROR_NOT_OK_RETURN(status);
184 |
185 |     talloc_free(attr->value_ctr.values[0].blob->data);
186 |     *attr->value_ctr.values[0].blob = plain_data;
187 |
188 |     return WERR_OK;
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         = calloc_array(msg, struct ldb_message_element,
255 |                                     msg->num_elements);
256 |     W_ERROR_HAVE_NO_MEMORY(msg->elements);
257 |
258 |     md = calloc(mem_ctx, struct replPropertyMetaDataBlob);
259 |     W_ERROR_HAVE_NO_MEMORY(md);
260 |
261 |     md->version           = 1;
262 |     md->reserved          = 0;
263 |     md->ctr.ctrl.count    = in->meta_data_ctr->count;
264 |     md->ctr.ctrl.reserved = 0;
265 |     md->ctr.ctrl.array    = calloc_array(mem_ctx,
266 |                                     struct replPropertyMetaDatum,
267 |                                     md->ctr.ctrl.count + 1); /* +1 because of
-> the RDN attribute */
268 |     W_ERROR_HAVE_NO_MEMORY(md->ctr.ctrl.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 replPropertyMetaDatum *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.ctrl.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.ctrl.array[md->ctr.ctrl.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.ctrl.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 repsFromTol *source_dsa,
346 |                                               const struct drsuapi_DsReplicaCursor2CtrEx
-> | *uptodateness_vector,
347 |                                               const DATA_BLOB *gensec_key,
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_key,
-> | 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_drstuapi.h"
50 #include "librpc/gen_ndr/ndr_drslblobs.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
->
74 |                                     *module,
75 |                                     struct ldb_request
->
76 |                                     *req,
77 |                                     struct
->
78 |                                     dsdb_extended_replicated_objects *objs)
79 |
80 |
81 |
82 |
83 |
84 |
85 |
86 |
87 |
88 |
89 |
90 |
91 |
92 |
93 |
94 |
95 |
96 |
97 |
98 |
99 |
100|

```

```

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);

```



```

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 invocationId */
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, sambd_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.ctrl.count = msg->num_elements;
472 |         nmd.ctr.ctrl.array = talloc_array(msg,
473 |                                           struct replPropertyMetaDatum,
474 |                                           nmd.ctr.ctrl.count);
475 |         if (!nmd.ctr.ctrl.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 replPropertyMetaDatum *m = &nmd.ctr.ctrl.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)) {

```



```

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.ctrl.count; i++) {
774 |             md->ctr.ctrl.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_replPropertyMetaDatum_conflict_compare(struct replPropertyMetaDatum
-> | *m1,
818 |                                                         struct replPropertyMetaDatum
-> | *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.ctrl.count = omd.ctr.ctrl.count + rmd->ctr.ctrl.count;
918     nmd.ctr.ctrl.array = talloc_array(ar->sub.mem_ctx,
919                                     struct replPropertyMetaData1,
920                                     nmd.ctr.ctrl.count);
921     if (!nmd.ctr.ctrl.array) return replmd_replicated_request_werror(ar,
-> WERR_NOMEM);
922
923     /* first copy the old meta data */
924     for (i=0; i < omd.ctr.ctrl.count; i++) {
925         nmd.ctr.ctrl.array[ni] = omd.ctr.ctrl.array[i];
926         ni++;
927     }
928
929     /* now merge in the new meta data */
930     for (i=0; i < rmd->ctr.ctrl.count; i++) {
931         bool found = false;
932
933         rmd->ctr.ctrl.array[i].local_usn = seq_num;
934
935         for (j=0; j < ni; j++) {
936             int cmp;
937
938             if (rmd->ctr.ctrl.array[i].attid != nmd.ctr.ctrl.array[j].attid)
-> {
939                 continue;
940             }
941
942             cmp = replmd_replPropertyMetaData1_conflict_compare(&rmd->ctr.ctr
-> l.array[i],
943 |
-> &nmd.ctr.ctrl.array[j]);
944             if (cmp > 0) {
945                 /* replace the entry */
946                 nmd.ctr.ctrl.array[j] = rmd->ctr.ctrl.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.ctrl.array[ni] = rmd->ctr.ctrl.array[i];
962         ni++;
963     }
964
965     /*
966     * finally correct the size of the meta_data array
967     */
968     nmd.ctr.ctrl.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.ctrl.count - 1;
980
981         rdn_p = &nmd.ctr.ctrl.array[rdn_idx];
982         replmd_replPropertyMetaDataCtrl_sort(&nmd.ctr.ctrl, &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 replPropertyMetaData 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", &nm_d_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]
->
1102 |     .guid_value);
1103 |     if (!tmp_str) return replmd_replicated_request_werror(ar, WERR_NOMEM);
1104 |
1105 |     filter = talloc_asprintf(ar->sub.mem_ctx, "(objectGUID=%s)", tmp_str);
1106 |     if (!filter) return replmd_replicated_request_werror(ar, WERR_NOMEM);
1107 |     talloc_free(tmp_str);
1108 |
1109 |     ret = ldb_build_search_req(&ar->sub.search_req,
1110 |                               ar->module->ldb,
1111 |                               ar->sub.mem_ctx,
1112 |                               ar->objs->partition_dn,
1113 |                               LDB_SCOPE_SUBTREE,
1114 |                               filter,
1115 |                               NULL,
1116 |                               NULL,
1117 |                               ar,
1118 |                               replmd_replicated_apply_search_callback);
1119 |     if (ret != LDB_SUCCESS) return replmd_replicated_request_error(ar, ret);
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_drstuapi_DsReplicaCursor2_compare(const struct drstuapi_DsReplicaCursor2
-> *c1,
1175 |                                                    const struct drstuapi_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 drstuapi_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
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.ctrl                = *ar->objs->source_dsa;
1366 |         /* and fix some values... */
1367 |         nrf.ctr.ctrl.consecutive_sync_failures = 0;
1368 |         nrf.ctr.ctrl.last_success            = now;
1369 |         nrf.ctr.ctrl.last_attempt            = now;
1370 |         nrf.ctr.ctrl.result_last_attempt     = WERR_OK;
1371 |         nrf.ctr.ctrl.highwatermark.highest_usn = nrf.ctr.ctrl.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,
->
1504 |             WERR_DS_DRA_INTERNAL_ERROR);
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
->
1560 |     ldb_request *req)
1561 | {
1562 |     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 |     int ret;
1630 |
1631 |     while (handle->state != LDB_ASYNC_DONE) {
1632 |         ret = replmd_wait_none(handle);
1633 |         if (ret != LDB_SUCCESS) {
1634 |             return ret;
1635 |         }
1636 |     }
1637 |
1638 |     return handle->status;
1639 | }
1640 |
1641 | static int replmd_wait(struct ldb_handle *handle, enum ldb_wait_type type)
1642 | {
1643 |     if (type == LDB_WAIT_ALL) {
1644 |

```

```
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_drstuapi.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 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("FF00000000000000000000000000000000
-> 00000000");
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 it 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 whether 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=$HOME/samba4
24 |thesis@tempo4:~/SAMBA_4_0_rev_21859/source> make && make install
25 |thesis@tempo4:~/SAMBA_4_0_rev_21859/source> cd ~/samba4/
26 |thesis@tempo4:~/samba4> /sbin/ifconfig vmnet24
27 |vmnet24  Protokoll:Ethernet  Hardware Adresse 00:50:56:C0:00:18
28 |         inet Adresse:172.31.24.1  Bcast:172.31.24.255  Maske:255.255.255.0
29 |         inet6 Adresse: fe80::250:56ff:fec0:18/64  Gültigkeitsbereich:Verbindung
30 |         UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
31 |         RX packets:127758 errors:0 dropped:0 overruns:0 frame:0
32 |         TX packets:32501 errors:0 dropped:0 overruns:0 carrier:0
33 |         collisions:0 Sendewarteschlangenlänge:1000
34 |         RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
35 |
36 |
37 |Die folgenden Konfigurationen müssen angepasst werden!
38 |Achtung: Die krb5.conf und hosts sind sehr wichtig!
39 |
40 |
41 |thesis@tempo4:~/samba4> cat ~/samba4/etc/smb.conf
42 |[globals]
43 |     netbios name             = smbtoruredc
44 |     workgroup                = W2K3-THESIS
45 |     realm                    = W2K3-THESIS.VMNET24.VM.BASE
46 |     sam database              = smbtoruredc_samdb.ldb
47 |     secrets database          = smbtoruredc_secrets.ldb
48 |
49 |     server role               = domain controller
50 |
51 |     bind interfaces only     = yes
52 |     interfaces                = vmnet24
53 |
54 |[c$]
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.vmmnet24.vmm.base -U
-> administrator%test -W W2K3-THESIS --realm w2k3-thesis.vmmnet24.vmm.base NET-API-BECOME-DC
92 | Using seed 1174111058
93 | Become DC [smbtorturedc] of Domain[W2K3-THESIS]/[w2k3-thesis.vmmnet24.vmm.base]
94 | Promotion Partner is Server[w2k3-21.w2k3-thesis.vmmnet24.vmm.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.vmmnet24.vmm.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=vmmnet24,DC=vm,DC=base]
101 |         objectGUID[f1bb4021-522f-4267-adcd-7326a86009b9]
102 |         invocationId[26c94114-cfc2-429b-b8f7-c45ae89938f0]
103 | Pathes under PRIVATE_DIR[/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=vmmnet24,DC=vm,DC=base =>
-> smb_torturedc_schema.ldb]
-> Config Partition[CN=Configuration,DC=w2k3-thesis,DC=vmmnet24,DC=vm,DC=base =>
-> smb_torturedc_config.ldb]
107 | Domain Partition[DC=w2k3-thesis,DC=vmmnet24,DC=vm,DC=base => smb_torturedc_domain.ldb]
108 | Setting up smb_torturedc_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 smb_torturedc_samdb.ldb attributes
113 | Setting up smb_torturedc_samdb.ldb rootDSE
114 | Erasing data from partitions
115 | Setting up smb_torturedc_samdb.ldb indexes
116 | Setting up smb_torturedc_samdb.ldb templates
117 | Setting up smb_torturedc_secrets.ldb
118 | WARNING: probable memory leak in ldb /home/thesis/samba4/private/smb_torturedc_samdb.ldb -
-> 837 blocks (startup 247) 14819 bytes
119 | Open the SAM LDB with system credentials: smb_torturedc_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=vmmnet24,DC=vm,DC=base]
-> objects[133]
124 | Schema-DN[CN=Schema,CN=Configuration,DC=w2k3-thesis,DC=vmmnet24,DC=vm,DC=base]
-> objects[133]
125 | Schema-DN[CN=Schema,CN=Configuration,DC=w2k3-thesis,DC=vmmnet24,DC=vm,DC=base]
-> objects[133]
126 | Schema-DN[CN=Schema,CN=Configuration,DC=w2k3-thesis,DC=vmmnet24,DC=vm,DC=base]
-> objects[133]
127 | Schema-DN[CN=Schema,CN=Configuration,DC=w2k3-thesis,DC=vmmnet24,DC=vm,DC=base]
-> objects[133]
128 | Schema-DN[CN=Schema,CN=Configuration,DC=w2k3-thesis,DC=vmmnet24,DC=vm,DC=base]
-> objects[133]
129 | Schema-DN[CN=Schema,CN=Configuration,DC=w2k3-thesis,DC=vmmnet24,DC=vm,DC=base]
-> objects[133]
130 | Schema-DN[CN=Schema,CN=Configuration,DC=w2k3-thesis,DC=vmmnet24,DC=vm,DC=base]
-> objects[133]
131 | Schema-DN[CN=Schema,CN=Configuration,DC=w2k3-thesis,DC=vmmnet24,DC=vm,DC=base]
-> objects[133]
132 | Schema-DN[CN=Schema,CN=Configuration,DC=w2k3-thesis,DC=vmmnet24,DC=vm,DC=base]
-> objects[75]
133 | Analyze and apply schema objects
134 | WARNING: probable memory leak in ldb /home/thesis/samba4/private/smb_torturedc_samdb.ldb -
-> 2243 blocks (startup 908) 200331 bytes
135 | Reopen the SAM LDB with system credentials and a already stored schema:
-> smb_torturedc_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=vmmnet24,DC=vm,DC=base] objects[133]
139 | Partition[CN=Configuration,DC=w2k3-thesis,DC=vmmnet24,DC=vm,DC=base] objects[133]
140 | Partition[CN=Configuration,DC=w2k3-thesis,DC=vmmnet24,DC=vm,DC=base] objects[133]
141 | Partition[CN=Configuration,DC=w2k3-thesis,DC=vmmnet24,DC=vm,DC=base] objects[133]
142 | Partition[CN=Configuration,DC=w2k3-thesis,DC=vmmnet24,DC=vm,DC=base] objects[133]
143 | Partition[CN=Configuration,DC=w2k3-thesis,DC=vmmnet24,DC=vm,DC=base] objects[133]
144 | Partition[CN=Configuration,DC=w2k3-thesis,DC=vmmnet24,DC=vm,DC=base] objects[133]

```

```

145 | Partition[CN=Configuration,DC=w2k3-thesis,DC=vmnet24,DC=vm,DC=base] objects[133]
146 | Partition[CN=Configuration,DC=w2k3-thesis,DC=vmnet24,DC=vm,DC=base] objects[133]
147 | Partition[CN=Configuration,DC=w2k3-thesis,DC=vmnet24,DC=vm,DC=base] objects[133]
148 | Partition[CN=Configuration,DC=w2k3-thesis,DC=vmnet24,DC=vm,DC=base] objects[133]
149 | Partition[CN=Configuration,DC=w2k3-thesis,DC=vmnet24,DC=vm,DC=base] objects[84]
150 | Partition[DC=w2k3-thesis,DC=vmnet24,DC=vm,DC=base] objects[133]
151 | Partition[DC=w2k3-thesis,DC=vmnet24,DC=vm,DC=base] objects[82]
152 | mark ROOTDSE with isSynchronized=TRUE
153 | Reopen the SAM LDB with system credentials and all replicated data:
-> | smbtoruredc_samdb.ldb
154 | Delete of machine account was successful.
155 |
156 |
157 | thesis@tempo4:~/samba4> ls -l ~/samba4/private/
158 | insgesamt 10736
159 | -rw----- 1 thesis thesis 5070848 2007-03-17 02:14 smbtoruredc_config.ldb
160 | -rw----- 1 thesis thesis 1204224 2007-03-17 02:14 smbtoruredc_domain.ldb
161 | -rw----- 1 thesis thesis 147456 2007-03-17 02:14 smbtoruredc_samdb.ldb
162 | -rw----- 1 thesis thesis 4435968 2007-03-17 02:14 smbtoruredc_schema.ldb
163 | -rw----- 1 thesis thesis 674 2007-03-17 02:14 smbtoruredc_secrets.keytab
164 | -rw----- 1 thesis thesis 131072 2007-03-17 02:14 smbtoruredc_secrets.ldb
165 | drwxr-sr-x 2 thesis thesis 6 2007-03-15 15:27 tls
166 |
167 |
168 | Die Daten lokal überprüfen:
169 |
170 |
171 | thesis@tempo4:~/samba4> bin/ldbsearch -H ~/samba4/private/smbtoruredc_samdb.ldb
-> | "(name=Administrator)"
172 | # record 1
173 | dn: CN=Administrator,CN=Users,DC=w2k3-thesis,DC=vmnet24,DC=vm,DC=base
174 | objectClass: top
175 | objectClass: person
176 | objectClass: organizationalPerson
177 | objectClass: user
178 | cn: Administrator
179 | description: Built-in account for administering the computer/domain
180 | instanceType: 4
181 | whenCreated: 20061217150534.0Z
182 | whenChanged: 20061217153251.0Z
183 | uSNCreated: 9065
184 | uSNChanged: 9065
185 | nTSecurityDescriptor: O:S-1-5-21-2283420010-3047391439-292721624-512G:S-1-5-21-
186 | -2283420010-3047391439-292721624-512D:PAI(A; ;RPLCLORC; ;AU)(A; ;RPWPCRCDCCLCLO
187 | RCWOWSDSW; ;BA)(A; ;RPWPCRCDCCLCLORCWOWSDSW; ;S-1-5-21-2283420010-3047391439-2
188 | 92721624-519)(A; ;RPWPCRCDCCLCLORCWOWSDSW; ;S-1-5-21-2283420010-3047391439-2927
189 | 21624-512)(A; ;RPWPCRCDCCLCLORCWOWSDSDTSW; ;SY)(OA; ;RP;037088f8-0ae1-11d2-b422
190 | -00a0c968f939;bf967aba-0de6-11d0-a285-00aa003049e2;RU)(OA; ;RP;59ba2f42-79a2-1
191 | 1d0-9020-00c04fc2d3cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)(OA; ;RP;bc0ac24
192 | 0-79a9-11d0-9020-00c04fc2d4cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)(OA; ;RP
193 | ;4c164200-20c0-11d0-a768-00aa006e0529;bf967aba-0de6-11d0-a285-00aa003049e2;RU
194 | )(OA; ;RP;5f202010-79a5-11d0-9020-00c04fc2d4cf;bf967aba-0de6-11d0-a285-00aa003
195 | 049e2;RU)(OA; ;RPLCLORC; ;bf967aba-0de6-11d0-a285-00aa003049e2;RU)(OA; ;CR;ab721
196 | a53-1e2f-11d0-9819-00aa0040529b; ;WD)(OA; ;CR;ab721a53-1e2f-11d0-9819-00aa00405
197 | 29b; ;PS)(OA; ;RPWP;bf967a7f-0de6-11d0-a285-00aa003049e2; ;S-1-5-21-2283420010-3
198 | 047391439-292721624-517)(OA; ;RP;037088f8-0ae1-11d2-b422-00a0c968f939;4828cc14
199 | -1437-45bc-9b07-ad6f015e5f28;RU)(OA; ;RP;59ba2f42-79a2-11d0-9020-00c04fc2d3cf;
200 | 4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)(OA; ;RP;bc0ac240-79a9-11d0-9020-00c04
201 | fc2d4cf;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)(OA; ;RP;4c164200-20c0-11d0-a7
202 | 68-00aa006e0529;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)(OA; ;RP;5f202010-79a5
203 | -11d0-9020-00c04fc2d4cf;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)(OA; ;RPLCLORC
204 | ; ;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)(OA; ;RP;46a9b11d-60ae-405a-b7e8-ff8
205 | a58d456d2; ;S-1-5-32-560)(OA; ;RPWP;6db69a1c-9422-11d1-aebd-0000f80367c1; ;S-1-5
206 | -32-561)S:AI(AU;SA;WPWOWD; ;WD)(OU;CIIOIDSA;WP;f30e3bbe-9ff0-11d1-b603-0000f8
207 | 0367c1;bf967aa5-0de6-11d0-a285-00aa003049e2;WD)(OU;CIIOIDSA;WP;f30e3bbf-9ff0-
208 | 11d1-b603-0000f80367c1;bf967aa5-0de6-11d0-a285-00aa003049e2;WD)
209 | name: Administrator
210 | objectGUID: 299cb4b3-cd36-419d-8f15-1b030bc02b80
211 | replPropertyMetaData:: AQAAAAAAAAAgAAAAAAAAAAAAAAAAABAAAAvu2V+wIAAAAUk0NCfVjRR6E
212 | ga9/Rb0HWAiAAAAAAAAABpIwAAAAAAAAA0AAAABAAAAvu2V+wIAAAAUk0NCfVjRR6Ega9/Rb0HWAiAA
213 | AAAAAABpIwAAAAAAAAEAAgABAAAAvu2V+wIAAAAUk0NCfVjRR6Ega9/Rb0HWAiAAAAAAAAABpIwAAA
214 | AAAAAIAAgABAAAAvu2V+wIAAAAUk0NCfVjRR6Ega9/Rb0HWAiAAAAAAAAABpIwAAAAAAAAA0AAGABAA
215 | AAvu2V+wIAAAAUk0NCfVjRR6Ega9/Rb0HWAiAAAAAAAAABpIwAAAAAAAAABkBAgACAAAAI/SV+wIAAAAU
216 | k0NCfVjRR6Ega9/Rb0HWA4TAAAAAAAAABpIwAAAAAAAAEACQABAAAAvu2V+wIAAAAUk0NCfVjRR6Eg
217 | a9/Rb0HWAiAAAAAAAAABpIwAAAAAAAAAgACQABAAAAvu2V+wIAAAAUk0NCfVjRR6Ega9/Rb0HWAiAAA
218 | AAAAAABpIwAAAAAAAAABACQABAAAAvu2V+wIAAAAUk0NCfVjRR6Ega9/Rb0HWAiAAAAAAAAABpIwAAAA
219 | AABkACQABAAAAvu2V+wIAAAAUk0NCfVjRR6Ega9/Rb0HWAiAAAAAAAAABpIwAAAAAAAAACQABAAAA
220 | Avu2V+wIAAAAUk0NCfVjRR6Ega9/Rb0HWAiAAAAAAAAABpIwAAAAAAAAAC0ACQABAAAAvu2V+wIAAAAU
221 | k0NCfVjRR6Ega9/Rb0HWAiAAAAAAAAABpIwAAAAAAAAADcACQABAAAAvu2V+wIAAAAUk0NCfVjRR6Ega
222 | 9/Rb0HWAiAAAAAAAAABpIwAAAAAAAAAD4ACQABAAAAvu2V+wIAAAAUk0NCfVjRR6Ega9/Rb0HWAiAAAA
223 | AAAABpIwAAAAAAAAEACQABAAAAvu2V+wIAAAAUk0NCfVjRR6Ega9/Rb0HWAiAAAAAAAAABpIwAAAAA
224 | AAFYACQABAAAAvu2V+wIAAAAUk0NCfVjRR6Ega9/Rb0HWAiAAAAAAAAABpIwAAAAAAAFoACQABAAAA
225 | vu2V+wIAAAAUk0NCfVjRR6Ega9/Rb0HWAiAAAAAAAAABpIwAAAAAAAF4ACQABAAAAvu2V+wIAAAAUk
226 | 0NCfVjRR6Ega9/Rb0HWAiAAAAAAAAABpIwAAAAAAAGAACQABAAAAvu2V+wIAAAAUk0NCfVjRR6Ega9
227 | /Rb0HWAiAAAAAAAAABpIwAAAAAAAGIACQABAAAAvu2V+wIAAAAUk0NCfVjRR6Ega9/Rb0HWAiAAAAA
228 | AABpIwAAAAAAAIoACQABAAAAvu2V+wIAAAAUk0NCfVjRR6Ega9/Rb0HWAiAAAAAAAAABpIwAAAAAA
229 | AIsACQABAAAAvu2V+wIAAAAUk0NCfVjRR6Ega9/Rb0HWAiAAAAAAAAABpIwAAAAAAAJIACQABAAAAv

```

```

230 | u2V+wIAAAAUk0NCfVjRR6Ega9/Rb0HWaiAAAAAABpIwAAAAAAJYACQABAAAAI/SV+wIAAAAUk0
231 | NcfVjRR6Ega9/Rb0HW4TAAAAAABpIwAAAAAAJwACQABAAAAVU2V+wIAAAAUk0NCfVjRR6Ega9/
232 | Rb0HWaiAAAAAABpIwAAAAAAJ8ACQABAAAAVU2V+wIAAAAUk0NCfVjRR6Ega9/Rb0HWaiAAAAA
233 | AABpIwAAAAAAKACQABAAAAVU2V+wIAAAAUk0NCfVjRR6Ega9/Rb0HWaiAAAAAABpIwAAAAA
234 | N0ACQABAAAAVU2V+wIAAAAUk0NCfVjRR6Ega9/Rb0HWaiAAAAAABpIwAAAAAAC4BCQABAAAAVU
235 | 2V+wIAAAAUk0NCfVjRR6Ega9/Rb0HWaiAAAAAABpIwAAAAAA4DCQABAAAAVU2V+wIAAAAUk0N
236 | CfvjRR6Ega9/Rb0HWaiAAAAAABpIwAAAAAAGQDCQABAAAAVU2V+wIAAAAUk0NCfVjRR6Ega9/R
237 | b0HWaiAAAAAABpIwAAAAAAMAAAABAAAAVU2V+wIAAAAUk0NCfVjRR6Ega9/Rb0HWaiAAAAA
238 | ABpIwAAAAAA==
239 | userAccountControl: 66048
240 | codePage: 0
241 | countryCode: 0
242 | dBCSPwd:: Afxaa+e8aSmq07Q1tRQE7g==
243 | unicodePwd:: DLauIAX3178qgoB5c7iVNw==
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,DC
255 | =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: 0:S-1-5-21-2283420010-3047391439-292721624-512G:S-1-5-21
294 | -2283420010-3047391439-292721624-512D:PAI(A;;RPLCLORC;;;AU)(A;;RPWPCCRDCLCLO
295 | RCWOWSDSW;;;BA)(A;;RPWPCCRDCLCLORCWOWSDSW;;;S-1-5-21-2283420010-3047391439-2
296 | 92721624-519)(A;;RPWPCCRDCLCLORCWOWSDSW;;;S-1-5-21-2283420010-3047391439-2927
297 | 21624-512)(A;;RPWPCCRDCLCLORCWOWSDSW;;;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;CIIODISA;WP;f30e3bbe-9ff0-11d1-b603-0000f8
315 | 0367c1;bf967aa5-0de6-11d0-a285-00aa003049e2;WD)(OU;CIIODISA;WP;f30e3bbf-9ff0-

```

```
316 | 11d1-b603-0000f80367c1;bf967aa5-0de6-11d0-a285-00aa003049e2;WD)
317 | name: Administrator
318 | objectGUID: 299cb4b3-cd36-419d-8f15-1b030bc02b80
319 | replPropertyMetaData:: AAAAAAAAAAAGAAAAAAAAAAAAAAAAABAAAAvu2V+wIAAAAUk0NcfVjRR6E
320 | ga9/Rb0HWAiAAAAAAAAABpIwAAAAAAAA0AAAABAAAAvu2V+wIAAAAUk0NcfVjRR6Ega9/Rb0HWAiAA
321 | AAAAAABpIwAAAAAAAAEAAGABAAAAvu2V+wIAAAAUk0NcfVjRR6Ega9/Rb0HWAiAAAAAAAAABpIwAAA
322 | AAAAAIAAGABAAAAvu2V+wIAAAAUk0NcfVjRR6Ega9/Rb0HWAiAAAAAAAAABpIwAAAAAAAA0AAGABAA
323 | AAvu2V+wIAAAAUk0NcfVjRR6Ega9/Rb0HWAiAAAAAAAAABpIwAAAAAAAAABkBAgACAAAAI/SV+wIAAAA
324 | uK0NcfVjRR6Ega9/Rb0HW4TAAAAAAAAABpIwAAAAAAAAEACQABAAAAvu2V+wIAAAAUk0NcfVjRR6E
325 | a9/Rb0HWAiAAAAAAAAABpIwAAAAAAAAAGACQABAAAAvu2V+wIAAAAUk0NcfVjRR6Ega9/Rb0HWAiAAA
326 | AAAAAABpIwAAAAAAAAABACQABAAAAvu2V+wIAAAAUk0NcfVjRR6Ega9/Rb0HWAiAAAAAAAAABpIwAAA
327 | AABkACQABAAAAvu2V+wIAAAAUk0NcfVjRR6Ega9/Rb0HWAiAAAAAAAAABpIwAAAAAAAAcWACQABAAA
328 | Avu2V+wIAAAAUk0NcfVjRR6Ega9/Rb0HWAiAAAAAAAAABpIwAAAAAAAAAC0ACQABAAAAvu2V+wIAAAAU
329 | K0NcfVjRR6Ega9/Rb0HWAiAAAAAAAAABpIwAAAAAADcACQABAAAAvu2V+wIAAAAUk0NcfVjRR6Ega
330 | 9/Rb0HWAiAAAAAAAAABpIwAAAAAAD4ACQABAAAAvu2V+wIAAAAUk0NcfVjRR6Ega9/Rb0HWAiAAAA
331 | AAAAAABpIwAAAAAAAEACQABAAAAvu2V+wIAAAAUk0NcfVjRR6Ega9/Rb0HWAiAAAAAAAAABpIwAAAA
332 | AAFYACQABAAAAvu2V+wIAAAAUk0NcfVjRR6Ega9/Rb0HWAiAAAAAAAAABpIwAAAAAAAFoACQABAAA
333 | vu2V+wIAAAAUk0NcfVjRR6Ega9/Rb0HWAiAAAAAAAAABpIwAAAAAAAF4ACQABAAAAvu2V+wIAAAAUk
334 | 0NcfVjRR6Ega9/Rb0HWAiAAAAAAAAABpIwAAAAAAGACQABAAAAvu2V+wIAAAAUk0NcfVjRR6Ega9
335 | /Rb0HWAiAAAAAAAAABpIwAAAAAAGIACQABAAAAvu2V+wIAAAAUk0NcfVjRR6Ega9/Rb0HWAiAAAA
336 | AABpIwAAAAAAAIoACQABAAAAvu2V+wIAAAAUk0NcfVjRR6Ega9/Rb0HWAiAAAAAAAAABpIwAAAAAA
337 | AIsACQABAAAAvu2V+wIAAAAUk0NcfVjRR6Ega9/Rb0HWAiAAAAAAAAABpIwAAAAAAAJIACQABAAAAv
338 | u2V+wIAAAAUk0NcfVjRR6Ega9/Rb0HWAiAAAAAAAAABpIwAAAAAAAJYACQABAAAAI/SV+wIAAAAUk0
339 | NcfVjRR6Ega9/Rb0HW4TAAAAAAAAABpIwAAAAAAJwACQABAAAAvu2V+wIAAAAUk0NcfVjRR6Ega9/
340 | Rb0HWAiAAAAAAAAABpIwAAAAAAJ8ACQABAAAAvu2V+wIAAAAUk0NcfVjRR6Ega9/Rb0HWAiAAAA
341 | AABpIwAAAAAAKAACQABAAAAvu2V+wIAAAAUk0NcfVjRR6Ega9/Rb0HWAiAAAAAAAAABpIwAAAAAA
342 | N0ACQABAAAAvu2V+wIAAAAUk0NcfVjRR6Ega9/Rb0HWAiAAAAAAAAABpIwAAAAAAAC4BCQABAAAAvu
343 | 2V+wIAAAAUk0NcfVjRR6Ega9/Rb0HWAiAAAAAAAAABpIwAAAAAAAD4DCQABAAAAvu2V+wIAAAAUk0N
344 | CfvjRR6Ega9/Rb0HWAiAAAAAAAAABpIwAAAAAAGQDCQABAAAAvu2V+wIAAAAUk0NcfVjRR6Ega9/R
345 | b0HWAiAAAAAAAAABpIwAAAAAAMAAAABAAAAvu2V+wIAAAAUk0NcfVjRR6Ega9/Rb0HWAiAAAA
346 | ABpIwAAAAAAA==
347 | userAccountControl: 66048
348 | codePage: 0
349 | countryCode: 0
350 | dBCSPwd:: Afxaa+e8aSmq07Q1tRQE7g==
351 | unicodePwd:: DLauIX3178ggoB5c7iVNW==
352 | pwdLastSet: 128085090543125000
353 | primaryGroupID: 513
354 | objectSid: S-1-5-21-2283420010-3047391439-292721624-500
355 | adminCount: 1
356 | accountExpires: 9223372036854775807
357 | sAMAccountName: Administrator
358 | sAMAccountType: 805306368
359 | objectCategory: CN=Person,CN=Schema,CN=Configuration,DC=w2k3-thesis,DC=vmnet24
360 | ,DC=vm,DC=base
361 | isCriticalSystemObject: TRUE
362 | distinguishedName: CN=Administrator,CN=Users,DC=w2k3-thesis,DC=vmnet24,DC=vm,DC
363 | =base
364 |
365 | # returned 1 records
366 | # 1 entries
367 | # 0 referrals
368 |
```