МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Казанский национальный исследовательский технический университет**

**им. А.Н. Туполева-КАИ»**

**(КНИТУ-КАИ)**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

им. Ю.В. Кожевникова

Выпускная квалификационная работа

на тему:

«Управление схемой службы каталогов Microsoft»

направление подготовки 09.03.01 "Информатика и вычислительная техника"

профиль "Программное обеспечение средств вычислительной техники и

автоматизированных систем"

Студент группы 4410

Исмагилов И.Ф.\_\_\_\_\_\_

Руководитель:

Сотников С.В.\_\_\_\_\_\_\_

к.т.н., доцент.

Казань - 2016

MINISTRY OF EDUCATION AND SCIENCE OF THE RUSSIAN FEDERATION

**Federal State Educational Institution of Higher Education**

**«Kazan national research technical university**

**named after A.N. Tupolev-KAI»**

**(KNRTU-KAI)**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

DEPARTMENT APPLIED MATHEMATICS AND INFORMATICS

named after U.V. Kozhevnikov

Graduate qualification work

on the theme:

«Microsoft Directory Service’s Schema Management»

specialization 09.03.01 "Computer and Information Science"

specialized "Computer and Automated Systems’ Software"

Student group 4410

Ismagilov I.F.\_\_\_\_\_\_\_\_

Research advisor:

Sotnikov S.V.\_\_\_\_\_\_\_

Ph.D., assistant professor

Kazan – 2016

**АННОТАЦИЯ**

**к выпускной квалификационной работе**

**на тему**

**«Управление схемой службы каталогов Microsoft»**

Выполнил:

студент группы 4410

Института Компьютерных Технологий и Защиты Информации

КНИТУ-КАИ им. А.Н.Туполева

Исмагилов Ильшат Фархатович.

Цель проекта - разработка программного комплекса для управления схемой службы каталогов Microsoft.

Для достижения указанной цели были поставлены следующие задачи:

1. Разработать пользовательский интерфейс с возможностью управления схемой.
2. Протестировать взаимодействие программы с Windows Server 2003 R2, Windows Server 2008 R2, Windows Server 2012 R2.

Для решения поставленных задач использовались следующие технологии:

1. Среда разработки Borland Developer Studio 2006.
2. Язык программирования Object Pascal.
3. Lightweight Directory Access Protocol (LDAP).
4. Средство виртуализации VMware Workstation.

Объем дипломной работы 148 страниц, на которых размещены 50 рисунков и 3 таблицы. При написании диплома использовалось 8 источников.

**ANNOTATION**

**on graduate qualification work**

**on the theme**

**«Microsoft Directory Service’s Schema Management»**

Done by

Ismagilov Ilshat Farkhatovich

student of group 4410

Institute of Computer Technology and Information Protection

KNRTU-KAI named after A.N.Tupolev.

Aim of the project is development of software for managing the Microsoft Directory Service’s Schema.

To achieve this goal following tasks were set:

1. Develop the user interface with ability to control directory service’s schema.
2. Test the interaction of the program with Windows Server 2003 R2, Windows Server 2008 R2 and Windows Server 2012 R2.

To solve these tasks following technologies were used:

1. Borland Developer Studio 2006 environment.
2. Object Pascal programming language.
3. Lightweight Directory Access Protocol (LDAP).
4. VMware Workstation for PC virtualization.

The volume of this work is 152 pages, 50 pictures and 3 tables where placed. While writing the diploma eight sources have been used.

Содержание

[Введение 7](#_Toc453013762)

[1. Постановка задачи 9](#_Toc453013763)

[2. Управление схемой службы каталогов 11](#_Toc453013764)

[2.1. Служба каталогов 11](#_Toc453013765)

[2.1.1.Описание службы каталогов 11](#_Toc453013766)

[2.1.2. Настройка Active Directory 17](#_Toc453013767)

[2.2. Схема службы каталогов 23](#_Toc453013768)

[2.2.1. Описание схемы 23](#_Toc453013769)

[2.2.2. Классы 25](#_Toc453013770)

[2.2.3. Атрибуты 31](#_Toc453013771)

[2.3. Обзор методов управления схемой Active Directory 35](#_Toc453013772)

[2.3.1. Lightweight Directory Access Protocol 35](#_Toc453013773)

[2.3.2. Active Directory Service Interfaces 40](#_Toc453013774)

[2.3.3. Определение способа управления схемой 41](#_Toc453013775)

[2.4. Обзор существующих решений для управления схемой 42](#_Toc453013776)

[2.4.1. Стандартные утилиты Microsoft 42](#_Toc453013777)

[2.4.2. Программа LdapAdmin 48](#_Toc453013778)

[3. Описание программного комплекса 49](#_Toc453013779)

[3.1. Основные технические требования 49](#_Toc453013780)

[3.2. Содержимое рабочего каталога программы 50](#_Toc453013781)

[3.3. Руководство для пользователя 50](#_Toc453013782)

[3.4. Руководство для программиста 60](#_Toc453013783)

[3.4.1. Структура программного комплекса 60](#_Toc453013784)

[3.4.2. Описание используемых функций LDAP API 60](#_Toc453013785)

[3.4.3. Описание разработанных модулей 68](#_Toc453013786)

[4. Результаты тестирования программного комплекса 76](#_Toc453013787)

[4.1. Тестирование на Windows Server 2003 R2 76](#_Toc453013788)

[4.2. Тестирование на Windows Server 2008 R2 78](#_Toc453013789)

[4.3. Тестирование на Windows Server 2012 R2 80](#_Toc453013790)

[Заключение 83](#_Toc453013791)

[Conclusion 85](#_Toc453013792)

[Литература 87](#_Toc453013793)

[Приложение 1. Листинг программы 88](#_Toc453013794)

[Приложение 2. Экранные формы программы 152](#_Toc453013795)

**Содержания(англ)**

# Введение

Служба каталогов является фундаментом ИТ-инфраструктуры компании любого размера и любого направления деятельности. Она представляет средства для централизованного управления ресурсами сети. Под ресурсами подразумеваются объекты службы каталогов. Сюда входят все компоненты сетевой инфраструктуры. Например, пользователи, принтеры и т.д.

Служба каталогов Microsoft – это Active Directory. В лесу этого сервиса может быть создано более 2-х миллиардов объектов, что позволяет внедрять службу каталогов в компаниях с сотнями тысяч компьютеров и пользователей.

Каждый объект относится к определенному классу, который описан в схеме службы каталогов. Например, все пользователи – это объекты класса user.

Обычно стандартных классов Active Directory вполне достаточно. Однако все меняется, особенно в такой отрасли как IT, и наступает момент, когда необходимо расширить функционал службы каталогов, т.е. модифицировать схему.

Актуальностью данной темы является возможность расширения схемы администраторами службы каталогов и программистов, разрабатывающих программы, которые предусматривают взаимодействие с Active Directory.

Целью данного проекта является разработка программного комплекса, предоставляющий необходимый функционал для управления схемой службы каталогов Microsoft.

Практическая ценность данного комплекса заключается не только в возможности использования графического интерфейса программы для изменения схемы, а также внедрения самого модуля управления схемой в существующую информационную систему.

Представленная пояснительная записка дипломной работы состоит из введения, 4 разделов основной части, заключения, списка литературы и приложения.

В первом разделе формулируются задачи и основные требования.

Во втором разделе подробно описывается служба каталогов Active Directory и ее схема, рассматриваются технологии перепрограммирования схемы и готовые решения.

В третьем разделе приводится описание разработанного программного комплекса.

В четвертом разделе приводятся результаты тестирования на разных серверных операционных системах фирмы Microsoft.

Приложение включает в себя листинг программы и экранные формы.

# Постановка задачи

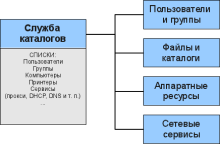
1. Разработать пользовательский интерфейс, аналогичный соответствующей оснастки MS Management Console (Snap-in Active Directory Schema) с изменениями, удовлетворяющими требованиям к указанному программному комплексу, которые приведены ниже.
2. Требования к программному комплексу.
   1. Пользовательский интерфейс должен быть интуитивно понятным и быть подобным интерфейсу оснастки MS Management Console (Snap-in Active Directory Schema)
   2. Возможности, которые должен реализовать данный программный комплекс:
      1. Подключение к владельцу служб каталогов.
      2. Управление схемой службы каталогов.
         1. Вывод содержимое схемы.
         2. Вывод дерева классов.
         3. Создание новых классов и атрибутов.
         4. Активация и деактивация классов и атрибутов.
         5. Изменение существующих классов и атрибутов.
         6. Добавление, удаление и изменение атрибутов в существующих объектах.
      3. Взаимодействие с модулем управления схемой должно быть реализовано в отдельном потоке.
   3. Программный комплекс должен быть реализован в виде модульной структуры.
   4. Программный комплекс должен быть выполнен в среде **Borland Developer Studio 2006**, с использованием языка программирования **Object Pascal**.
   5. Программный комплекс должен работать со следующими владельцами службы каталогов:
      1. Windows Server 2003 R2.
      2. Windows Server 2008 R2.
      3. Windows Server 2012 R2.
3. Привести подробное описание программного комплекса с точки зрения пользователя (руководство для пользователя) и с точки зрения программиста (руководство для программиста), а также сформулировать требования к аппаратному и программному обеспечению рабочей станции, на которой будут производиться необходимые действия с созданным программным комплексом.

# 2. Управление схемой службы каталогов

## 2.1. Служба каталогов

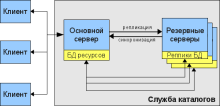
### 2.1.1.Описание службы каталогов

**Служба каталогов** — это сетевой сервис, представляющий централизованные средства управления ресурсами автоматизированной системы. Под ресурсами подразумеваются все компоненты сетевой инфраструктуры, которые используются для выполнения функций АСУ: пользователи, файлы и каталоги, устройства, сетевые сервисы и т.д. (рис.2.1.1).

**

**Рис. 2.1.1. Служба каталогов**

Как правило, служба каталогов состоит из базы данных, в которой размещены сведения о сетевых ресурсах и серверного ПО, представляющего механизмы доступа к этой базе. Как база данных сервиса каталогов, так и ее управляющая программа могут быть распределены на несколько серверов (рис. 2.1.2).

**

**Рис. 2.1.2. Распределенная служба каталогов**

Основными функциями службы каталогов являются следующие:

* Управление пользователями и группами (создание/удаление, настройка прав доступа).
* Управление ресурсами (представление в общий доступ, установка ограничений, удаленное администрирование и т.п.).
* Разграничение прав доступа (как правило, на уровне пользователей, групп и отдельных ресурсов).

Среди дополнительных функций сервиса каталогов можно указать, например, такие:

* поиск ресурсов;
* распространение сетевых политик;
* интеграция с другими сервисами.

**Сетевая политика** — совокупность правил, определяющих методы и средства взаимодействия с общими ресурсами в корпоративной сети.

Сетевые ресурсы в службе каталогов обычно представлены в виде иерархической структуры. Такой способ наиболее близок к реальной организационной модели подавляющего большинства предприятий и организаций. Корень иерархии описывает предприятие в целом, нижележащие уровни — подразделения и отдельные элементы. Для единообразного обращения к любому элементу иерархии протокол взаимодействия представляет унифицированную схему адресации — либо собственную, либо совместимую со стандартными схемами.

Технология Active Directory (AD) является службой каталогов, созданной компанией Microsoft. Служба каталогов содержит данные в организованном формате и предоставляет к ним упорядоченный доступ. Служба Active Directory — это не изобретение компании Microsoft, а реализация существующей индустриальной модели (а именно X.500), коммуникационного протокола (LDAP — Lightweight Directory Access Protocol) и технологии поиска данных (службы DNS).

Active Directory предназначен для хранения информации о всех сетевых ресурсах. Клиенты имеют возможность отправлять запросы Active Directory для получения информации о любом объекте сети. В список возможностей Active Directory входят следующие функции.

* Безопасное хранилище данных. Каждый объект в Active Directory имеет собственный список управления доступом (ACL), который содержит список ресурсов, получивших право доступа к объекту, а также предопределенный уровень доступа к этому объекту.
* Многофункциональный механизм запросов, основанный на созданном Active Directory глобальном каталоге (GC). Все клиенты, поддерживающие Active Directory, могут обращаться к этому каталогу.
* Репликация данных каталога на все контроллеры домена упрощает доступ к информации, повышает степень ее доступности и увеличивает надежность всей службы.
* Концепция модульного расширения, которая позволяет добавлять новые типы объектов или дополнять существующие объекты. Например, к объекту “пользователь” можно добавить атрибут “зарплата”.
* Сетевое взаимодействие с использованием нескольких протоколов. Служба Active Directory основана на модели X.500, благодаря чему поддерживаются различные сетевые протоколы, например, LDAP 2, LDAP 3 и HTTP.
* Для реализации службы имен контроллеров доменов и поиска сетевых адресов вместо NetBIOS используется служба DNS.

Информация каталога распределяется по всему домену, тем самым позволяя избежать чрезмерного дублирования данных.

Хотя Active Directory распределяет информацию каталога по различным хранилищам, пользователи имеют возможность запросить Active Directory на получение информации о других доменах. Глобальный каталог содержит сведения о всех объектах леса предприятия, помогая осуществлять поиск данных в рамках всего леса.

При запуске утилиты DCPROMO (программы повышения обычного сервера до контроллера домена) на компьютере под управлением Windows для создания нового домена, утилита создает домен на сервере DNS. Затем клиент связывается с сервером DNS для получения информации о своем домене. Сервер DNS предоставляет информацию не только о домене, но и о ближайшем контроллере домена. Клиентская система, в свою очередь, подключается к базе данных домена Active Directory на ближайшем контроллере домена с целью нахождения необходимых объектов (принтеров, файловых серверов, пользователей, групп, организационных подразделений), входящих в домен. Поскольку каждый контроллер домена хранит ссылки на другие домены в дереве, клиент может выполнять поиск во всем дереве домена.

Контролер домена — это компьютер с установленной службой Active Directory, который управляет доступом пользователей к ресурсам сети. Он хранит ресурсы (например, принтеры, папки с общим доступом), службы (например, электронная почта), людей (учетные записи пользователей и групп пользователей), компьютеры (учетные записи компьютеров). Число таких сохраненных ресурсов может достигать миллионов объектов. В качестве контролера домена могут выступать следующие версии MS Windows: Windows Server 2000/2003/2008/2012 кроме редакций Web-Edition.

Контролер домена помимо того, что является центром аутентификации сети, также является центром управления всеми компьютерами. Сразу после включения компьютер начинает обращаться к контролеру домена, задолго до появления окна аутентификации, таким образом выполняется аутентификация не только пользователя, вводящего логин и пароль но и аутентификация клиентского компьютера.

Разновидность Active Directory, которая перечисляет все объекты в лесу доменов, доступна для тех случаев, когда необходимо найти данные за пределами дерева доменов клиента. Подобная версия называется глобальный каталог. Глобальный каталог можно хранить на любом контроллере домена в лесу AD.

Глобальный каталог предоставляет быстрый доступ к каждому объекту, который располагаться в лесу доменов, но при этом содержит только некоторые параметры объектов. Для получения всех атрибутов следует обратиться к службе Active Directory целевого домена (контроллеру интересующего домена). Глобальный каталог можно настроить на предоставление необходимых свойств объектов.

Для упрощения процесса создания объектов Active Directory контроллер домена содержит копию и иерархию классов для всего леса. Служба Active Directory содержит структуры классов в расширяемой схеме, в которую можно добавить новые классы.

Схема (schema) — это часть конфигурационного пространства имен Windows, которое поддерживается всеми контроллерами доменов в лесу. Конфигурационное пространство имен Windows состоит из нескольких структурных элементов, таких как физическое расположение, сайты Windows и подсети.

Сайт (site) содержится внутри леса и может объединять компьютеры из любого домена, причем все компьютеры сайта должны иметь быстрые и надежные сетевые соединения для резервирования данных контроллера домена.

Подсеть (subnet) — это группа IP-адресов, выделенная сайту. Подсети позволяют ускорить репликацию данных Active Directory между контроллерами доменов.

**Именование объектов**

Active Directory поддерживает несколько способов именования объектов:

* характерное LDAP-имя;
* глобальный уникальный идентификатор (Global Unique Identifier, GUID);
* идентификатор безопасности (Security ID, SID);
* имя NetBIOS;
* основное имя субъекта системы безопасности (Security principal name).

В Active Directory имя объекта может быть представлено в одной из двух форм: *характерное имя* (DN, Distinguished Name), либо *относительное характерное имя* (RDN, Relative Distinguished Name). Характерное имя однозначно определяет объект в дереве. Можно провести аналогию между характерным именем в Active Directory и полным путем в файловой системе. Характерное имя включает информацию обо всех узлах дерева, которые необходимо пройти, чтобы прийти от данного объекта к корню дерева. В записи характерного имени используются сокращения, имеющие следующий смысл:

* **CN (Common Name)** - общее имя;
* **DC (Domain Component)** - составная часть доменного имени;
* **OU (Organizational Unit)** - организационная единица.

В Active Directory не может быть определено несколько объектов, имеющих одинаковое характерное имя, поскольку это привело бы к неоднозначности выбора конкретного объекта.

Большинство утилит, работающих с Active Directory, позволяют использовать другой способ именования объектов - канонические имена. В этом случае опускаются сокращения, обозначающие тип объекта или контейнера, и просто перечисляются их имена. Для обозначения имени домена используется соглашение о доменных именах. В характерном имени объекта присутствует компонент, идентифицирующий непосредственно сам объект, а не содержащие его контейнеры. Этот компонент называется относительным характерным именем объекта. Контейнер, который непосредственно содержит данный объект, называется родителем объекта.

Относительное характерное имя не обязано быть уникальным. Однако в силу того, что каждый объект должен однозначно идентифицироваться в Active Directory, относительное характерное имя объекта должно быть уникально в пределах того контейнера, что является его родителем.

С целью обеспечения уникальности объектов и облегчения их поиска каждому объекту при создании ставится в соответствие 128-разрядное число - *глобальный уникальный идентификатор* (GUID). Глобальный идентификатор является обязательным атрибутом любого объекта, который не может быть изменен ни при каких обстоятельствах. Существует и другая форма именования объектов, которая используется для обеспечения безопасности системы. Эти имена используются при входе пользователя в систему, позволяя провести его аутентификацию. Для этого можно использовать:

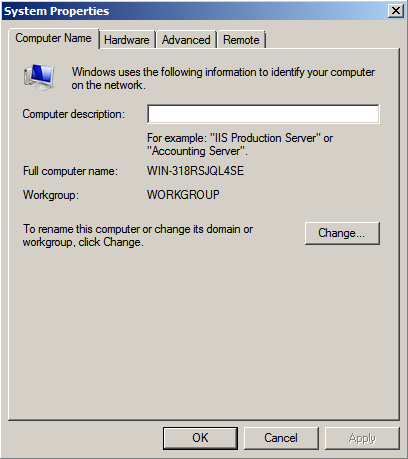
* *имена NetBIOS*;
* *основное имя пользователя* (user principal names). В Active Directory каждый пользователь имеет основное имя пользователя в формате <пользователь>@<DNS имя домена>. Основное имя образуется из имени пользователя и имени домена, к которому этот пользователь принадлежит.

### 2.1.2. Настройка Active Directory

В этом разделе приводится пример установки Active Directory на Windows Server 2012 R2.

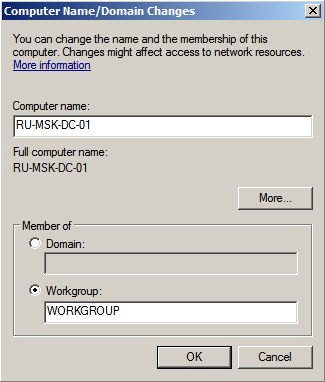
Перед началом установки роли **Active Directory Domain Services** необходимо присвоить серверу корректное имя, а затем указать статический IP-адрес в настройках сетевого подключения.

Для изменения имени сервера необходимо в окне «*System Properties*» на вкладке «*Computer Name*» нажать на кнопку «*Change*».

[](http://www.vmkh.net/wp-content/uploads/ustanovka-active-directory-domain-services-na-windows-server-2008-r2-3.jpg)

**Рис. 2.1.3. Свойства системы**

Далее в соответствующем поле необходимо указать новое имя сервера.

[](http://www.vmkh.net/wp-content/uploads/ustanovka-active-directory-domain-services-na-windows-server-2008-r2-4.jpg)

**Рис.2.1.4. Окно изменения имени компьютера**

Система предупредит о том, что для применения новых настроек необходимо перезагрузить сервер.

Для изменения IP-адреса необходимо открыть окно «*Local Area Connection Properties*». Затем выбрать «*Internet Protocol Version 4*» и нажать на кнопку «*Properties*».

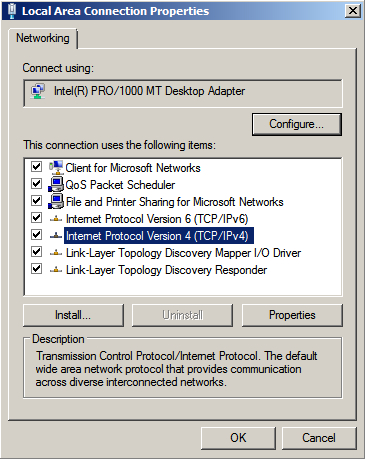
[](http://www.vmkh.net/wp-content/uploads/ustanovka-active-directory-domain-services-na-windows-server-2008-r2-12.jpg)

Рис. 2.1.5. Локальные подключения

Далее выбрать пункт «*Use the following IP address*» и указывается свободный IP-адрес, маска подсети и шлюз.

В поле «*Preferred DNS server*» указывается IP-адрес этого сервера, так как на сервере будет присутствовать роль DNS Server, которая устанавливается вместе с ролью «*Active Directory Domain Services*».

Для сохранения необходимо нажать на кнопку «OK».

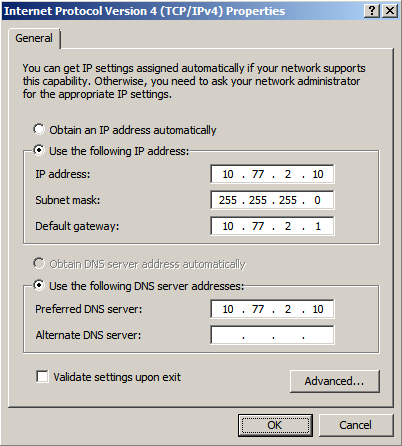
[](http://www.vmkh.net/wp-content/uploads/ustanovka-active-directory-domain-services-na-windows-server-2008-r2-13.jpg)

Рис. 2.1.6. Свойства TCP/IPv4

Далее необходимо открыть «*Server Manager*» и в пункте «*Roles*» выбрать «*Add Roles*».

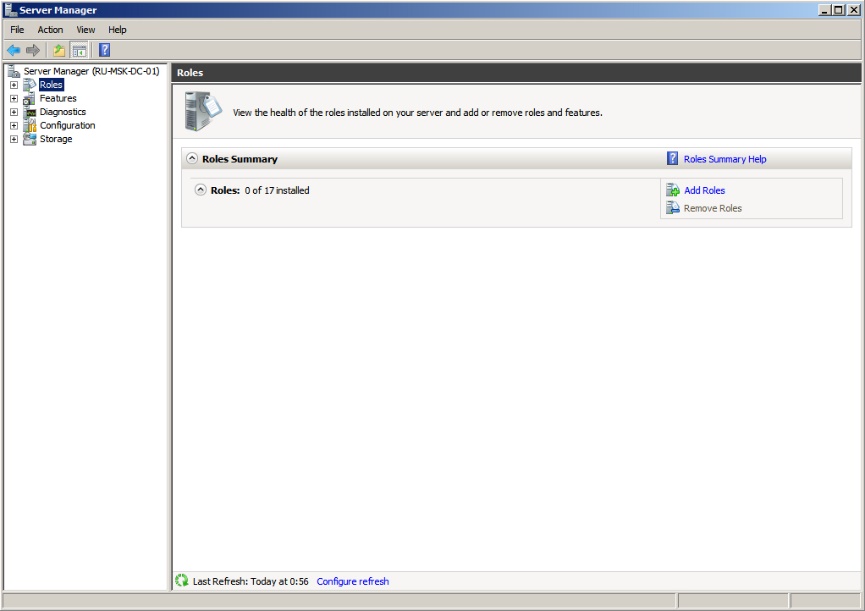
[](http://www.vmkh.net/wp-content/uploads/ustanovka-active-directory-domain-services-na-windows-server-2008-r2-15.jpg)

Рис. 2.1.7. Диспетчер серверов

В нашем случае выбирается роль «*Active Directory Domain Services*».

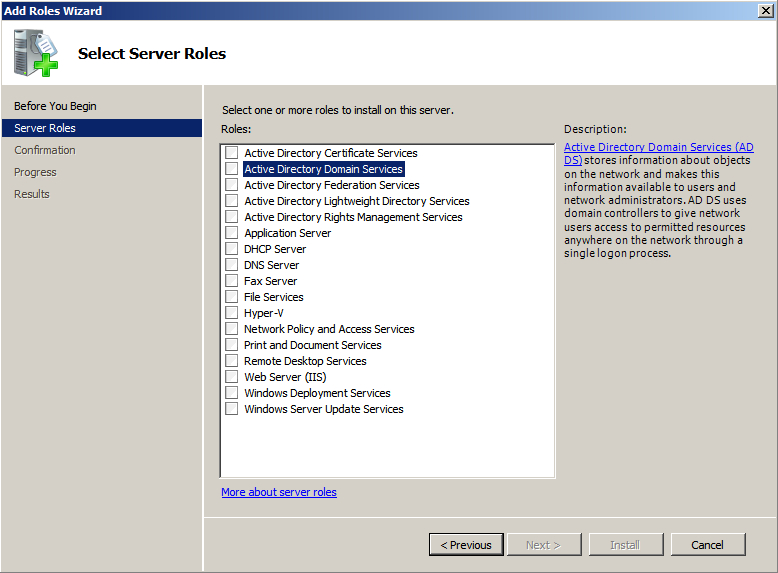
[](http://www.vmkh.net/wp-content/uploads/ustanovka-active-directory-domain-services-na-windows-server-2008-r2-17.jpg)

Рис. 2.1.8. Мастер добавления ролей

Далее необходимо нажимать «*Next*» и установить роль.

Следующим шагом необходимо нажать на кнопку «*Close this wizard and launch the Active Directory Domain Services Installation Wizard (****dcpromo.exe****)*», для того чтобы повысить роль вашего сервера до уровня контроллера домена.

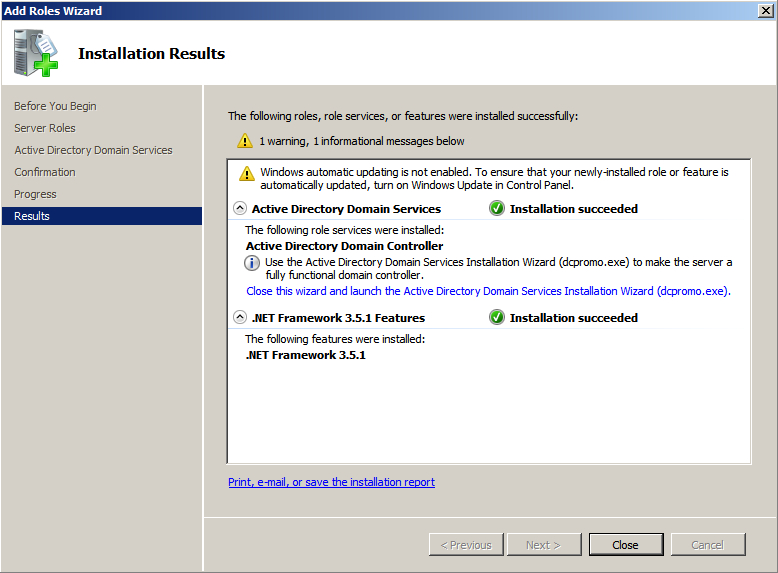
[](http://www.vmkh.net/wp-content/uploads/ustanovka-active-directory-domain-services-na-windows-server-2008-r2-23.jpg)

Рис. 2.1.9. Результаты работы мастера

[](http://www.vmkh.net/wp-content/uploads/ustanovka-active-directory-domain-services-na-windows-server-2008-r2-24.jpg)

Рис. 2.1.10. Мастер установки AD DS

Далее с помощью мастера установки проводятся настройка и установка Active Directory.

## 2.2. Схема службы каталогов

### 2.2.1. Описание схемы

Схема Active Directory представляет собой набор определений всех типов объектов каталога и связанных с ними атрибутов. Схема определяет способ хранения и конфигурирования в AD всех пользователей, компьютеров и других данных объек­тов, стандартизуя всю структуру Active Directory. Схема, защищенная с помощью спи­сков разграничительного контроля доступа (Discretionary Access Control List — DACL), управляет возможными атрибутами каждого объекта из Active Directory. По существу, схема представляет собой базовое определение самого каталога и является основой функциональности среды домена. Делегировать управление схемой следует осторож­но и лишь избранной группе администраторов, поскольку любые изменения схемы влияют на всю среду AD.

**Объекты схемы**

Объекты структуры Active Directory — User (пользователь), Printer (принтер), Computer (компьютер) и Site (сайт) — определены в схеме как объекты. Каждый объ­ект имеет список атрибутов, которые определяют его и могут быть использованы для поиска этого объекта. Например, объект User для работника с именем Вера Сердючка будет иметь атрибут FirstName (имя) Вера и атрибут LastName (фамилия) Сердючка. Кроме этих, можно назначить и другие атрибуты: название подразделения, адрес электронной почты и целый ряд других. Пользователи, выполняя поиск информации в Active Directory, могут делать запросы на основе этой информации, например, искать всех пользователей, работающих в отделе сбыта. Представление о том, сколько атрибутов имеется в Active Directory, дает тот факт, что сразу после инсталляции каж­дому объекту можно присвоить более 1000 атрибутов.

**Расширение схемы**

Одним из основных достоинств структуры Active Directory является возможность непосредственной модификации и расширения схемы для включения пользователь­ских атрибутов. Обычное расширение атрибутов происходит во время инсталляции последней версии Microsoft Exchange, которая расширяет схему, увеличивая ее размер в два раза. Модернизация Active Directory Windows 2000 до Active Directory Windows Server 2003 также расширяет схему, включая в нее атрибуты, специфические для Win­dows Server 2003.

Схема Active Directory расположена в контейнере с характерным именем (distinguished name) «*CN=Schema,CN=Configuration,DC=<ForestRoot>*»

**Модификация схемы с помощью интерфейса службы Active Directory**

Есть интересный метод преставления всех подробностей схемы Active Directory — с помощью утилиты интерфейса службы Active Directory (Active Directory Service Interfaces — ADSI). Эта утилита предназначена для упрощения доступа к Active Directory, и с ее помощью можно просмотреть любой внешний совместимый каталог LDAP. Ути­лита ADSI, позволяет просматривать, удалять и модифицировать атрибуты схемы. При модифицировании схемы необходимо соблюдать особую осто­рожность, поскольку проблемы, возникающие в схеме, исправлять очень трудно.

Для модификации схемы необходимо выполнение следующих условий:

* 1. Пользователь должен входить в группу «***Администраторы схемы***».
  2. Необходимо получить идентификатор для нового объекта (**object identifier – OID**). OID – это уникальный номер создаваемого объекта, который выделяет его в Active Directory. Для получения номера можно воспользоваться скриптом от Microsoft.

Каждый класс AD и каждый атрибут должны быть уникальны, поэтому каждому определению класса и атрибута нужно назначать уникальный идентификатор OID. Идентификатор представляет собой строку из десятичных чисел, разделенных точками - так же, как в IP-адресах. OID объекта **classSchema** помещается в его атрибуте **governsID**, а идентификатор OID объекта **attributeSchema** - в атрибуте **attributeID**.

Эти идентификаторы назначаются центром регистрации имен (Name Registration Authority) Международной организации по стандартизации (International Organization for Standardizatin, ISO).

Используемая в идентификаторах OID цифровая система обозначений имеет древовидную структуру. За отдельные участки дерева именования отвечают конкретные инстанции с четко разграниченными полномочиями, которые могут выделять "*подветви*" (arcs) другим инстанциям более низкого уровня, а те, в свою очередь, выделять "*подветви подветвей*" инстанциям еще более низкого уровня.

Кстати, не путайте идентификаторы OID с глобальными уникальными идентификаторами типа GUID (globally unique ID). OID содержит уникальное обозначение класса, а GUID - уникальное обозначение экземпляра объекта. Например, OID класса user - это 1.2.840.113556.1.5.9; каждый объект User, создаваемый из этого класса, будет иметь уникальный идентификатор GID, но являться потомком одного и того же класса (т.е. класса, OID которого равен 1.2.840.113556.1.5.9).

### 2.2.2. Классы

Каждый объект AD в контексте именования домена или в контексте именования конфигурации представляет собой пример, или экземпляр объектного класса (object class). Так, объект User представляет собой экземпляр класса user; а объект Computer, соответственно, экземпляр класса computer. Класс описывает объект AD и ассоциированные с ним свойства, или атрибуты, необходимые для формирования экземпляра объекта данного класса (т.е. обязательные атрибуты), либо те, которые могут присутствовать в экземпляре класса (т.е. факультативные атрибуты). Контекст именования схемы содержит определения всех классов и атрибутов, имеющихся в других контекстах именования AD. Но дело осложняется тем, что служба AD предусматривает хранение этих определений, которые в данном случае именуются объектами.

Таким образом, следует постоянно иметь в виду различие между объектом, который является экземпляром класса (это может быть, скажем, экземпляр объекта контекста именования конфигурации или контекста именования домена), и объектом, который является определением класса или атрибута (это может быть объект схемы в контексте именования схемы). Объекты схемы делятся на две категории: объекты classSchema определяют классы, а объекты attributeSchema - атрибуты. Каждый атрибут имеет лишь одно определение, но может использоваться с несколькими различными классами.

Представьте себе, что вы создаете объект User с именем James Bond. Этот объект является экземпляром класса user, и контекст именования схемы определяет его с помощью особого объекта classSchema, именуемого user. Служба AD сохраняет регистрационное имя объекта James Bond (пусть это будет имя BondJ), в атрибуте, именуемом SAMAccountName, определяемом контекстом именования схемы через особый объект attributeSchema, называемый SAMAccountName. Объект user категории classSchema определяет связь между объектом SAMAccountName и классом user. Вернее, объект SAMAccoutName представляет собой имя объекта attributeSchema, отображаемое при использовании протокола Lightweight Directory Access Protocol, или LDAP. Фактическое же имя объекта хранится в его атрибуте common name (cn) именуемом SAM-Account-Name.

#### **Определение объектов: объекты категории classSchema**

Объекты контекста именования схемы, относящиеся к категории classSchema, содержат определения классов. В этих определениях заключены следующие сведения:

правила, определяющие, где можно создавать объекты AD (например, в организационной единице (organizational unit, или OU), взаимосвязи, или производные классы, между объектами classchema (например, родительский класс данного класса, именуемый суперклассом (superclass)), отношения между атрибутами и объектами категории classSchema, в частности, обязательные и факультативные атрибуты класса.

Каждый объект classSchema относится к одной из трех категорий классов: абстрактная (abstract), вспомогательная (auxiliary) и структурная (structural).

**Абстрактные классы.** Объекты classSchema (иначе говоря, классы), относящиеся к абстрактной категории, выступают в качестве формальных параметров-заполнителей; эти классы нельзя использовать для создания экземпляра объекта в другом контексте именования AD. В службе AD абстрактные классы используются главным образом как родительские классы других классов, или суперклассы, так что администратор может создавать иерархическую структуру из объектных классов и определять систему наследования атрибутов между классами (в атрибуте subClasOf каждого класса хранится имя его суперкласса). Хотя наследование и не является прерогативой абстрактных классов, это именно то качество, из-за которого данные классы используются. Значение атрибута objectClassCategory абстрактных объектов classSchema равно 2.

**Вспомогательный класс.** Вспомогательные объекты classSchema подобны абстрактным классам в том, что они тоже не используются для создания экземпляров объектов AD. Вспомогательные классы напоминают файлы include и заголовочные файлы; с помощью этой категории классов можно включать набор из нескольких атрибутов в определение другого класса. Атрибуты systemAuxiliaryClass и auxiliaryClass являются атрибутами, которые способны содержать несколько значений (multivalue attributes). Иными словами, одному объекту classSchema можно поставить в соответствие множество вспомогательных классов. Атрибут с одним значением (single-value attribute) может содержать только одно значение, тогда как атрибут со множеством значений может содержать несколько значений.

При установке пакета Exchange 2000 в лесу AD программа установки Exchange 2000 Setup дополняет схему AD множеством новых классов. Некоторые из них относятся к категории вспомогательных классов и определяют атрибуты, которые используются только в Exchange 2000. Принадлежащие объекту classSchema атрибуты systemAuxiliaryClass или auxiliaryClass определяют объекты любого применимого вспомогательного класса. Значение атрибута objectClassCategory вспомогательных классов равно 3.

**Структурный класс.** Это единственный тип класса, с помощью которого можно создавать экземпляры объектов в других контекстах именования службы AD. Определения классов, внутри которых могут располагаться созданные экземпляры класса, определяемого объектами classSchema, содержатся в атрибутах systemPossSuperiors и possSuperiors. Так, принадлежащий классу user атрибут systemPossSuperiors определяет класс organizationalUnit в качестве возможного контейнера для экземпляров класса user, поэтому объект User можно создавать в экземпляре класса organizationalUnit. Эти определения известны как правила реализации (instantiation rules). Значение атрибута objectClassCategory структурных объектов classSchema равно 3.

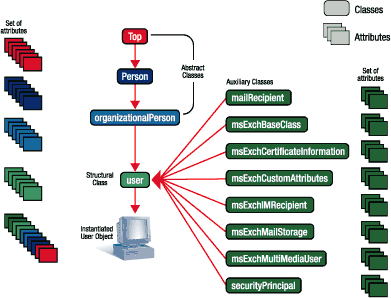


Рис. 2.2.1. Взаимодействие классов

На Рисунке 2.2.1 показано, как взаимодействуют три описанных типа классов. Вверху расположен абстрактный объект категории classSchema, который является суперклассом по отношению к абстрактному объекту Person категории classSchema. Person, в свою очередь, выступает в роли суперкласса по отношению к абстрактному объекту organizationaPerson, принадлежащему категории classSchema. Последний объект является суперклассом класса user (т.е. объекта user категории clssSchema). Наконец, объект user категории classSchema представляет собой структурный объект класса, определяющий объекты User (иными словами, экземпляры класса user).

Класс user наследует атрибуты своих суперклассов (см. Рисунок), равно как и различных вспомогательных классов. Каждый объект имеет набор обязательных атрибутов и набор факультативных атрибутов. Атрибуты systemMustContain и mustContain объекта категории classScema определяют его обязательные атрибуты, тогда как в атрибутах systemMayContain и mayContain объекта категории classSchema записаны факультативные атрибуты данного объекта. Оснастка ADSI Edit не дает возможности увидеть определенное значение для атрибутов systemMustContain и mustContain класса user. Это не означает, что у класса user нет обязательных атрибутов. Класс user наследует обязательные атрибуты своих суперклассов (скажем, класса Person) и вспомогательных классов (например, класса mailRecipient).

В нижеприведенной таблице перечислены ключевые атрибуты класса «*classSchema*».

Табл. 2.2.1. Атрибуты класса classSchema

|  |  |
| --- | --- |
| **lDAPDisplayName** | **Описание** |
| **cn** | У каждого объекта есть атрибут именования, из которого формируется относительное характерное имя (RDN). У класса classSchema это атрибут «*cn*» (Common-Name). |
| **lDAPDisplayName** | Имя, используемое LDAP клиентами, чтобы обращаться к классу. Это имя должно быть уникальным. |
| **schemaIDGUID** | Один из уникальных идентификаторов класса. Сервер сам генерирует значение этого атрибута, если он не был указан при создании класса или атрибута. |
| **adminDisplayName** | Имя, отображаемое в средствах администрирования. |
| **governsID** | Уникальный идентификатор класса. |
| **rDnAttId** | Определяет атрибут именования. Рекомендуется использовать «cn» в качестве значения этого атрибута. |
| **mustContain**, **systemMustContain** | Обязательные атрибуты. Могут быть заданы только при создании класса. |
| **mayContain**, **systemMayContain** | Необязательные атрибуты (возможные). |
| **possSuperiors**, **systemPossSuperiors** | Атрибуты, которые определяют какие классы могут быть законными родителями экземпляров данного класса. |
| **objectClassCategory** | Номер, указывающий тип класса:  Структурный – можно создавать экземпляры.  Абстрактный – класс предоставляет базовое определение для дочерних структурных классов.  Вспомогательный – может расширять другие типы классов. |
| **subClassOf** | Атрибут определяет родительский класс. |
| **auxiliaryClass**, **systemAuxiliaryClass** | Вспомогательные классы. |
| **defaultObjectCategory** | Устанавливается характерное имя суперкласса. |
| **defaultHidingValue** | TRUE, чтобы скрыть ненужный класс в средствах администрирования. |
| **systemFlags** | Числовое значение, которое содержит флаг для дополнительных свойств класса. |
| **systemOnly** | Логическое значение. При TRUE класс может изменять только система. |
| **defaultSecurityDescriptor** | Задает дескриптор безопасности по умолчанию для новых объектов данного класса |
| **isDefunct** | True – класс активный, false – класс деактивирован. |
| **description** | Описание класса. |
| **objectClass** | Определяет экземпляром какого класса является данный класс. |

### 2.2.3. Атрибуты

Объекты attributeSchema контекста именования схемы определяют атрибуты, предусмотренные в службе AD. Так же, как и объекты категории classSchema, все объекты типа attributeSchema имеют наборы атрибутов, в которых определены их характеристики.

Каждый атрибут схемы AD имеет заранее заданный синтаксис, определяющий тип информации, который может храниться в данном атрибуте. Например, телефонные номера хранятся в атрибутах иначе, нежели двоичные значения или строки. Синтаксис атрибута atributeSyntax объекта attributeSchema определяется с помощью идентификатора объектов Object Identifier (OID), а для определения синтаксиса атрибута oMSyntax того же объекта применяется обозначение объектной модели X/Open (X/Open Object Model). Атрибут объекта oMObjectClass тоже описывает его синтаксис, правда, в другой форме.

Значение атрибута oMSyntax всегда равно 127 (это значение свидетельствует о том, что синтаксис указывает на тип данных «объект»). Необходимо следить за тем, чтобы атрибут oMObjectClass объекта отображал корректное значение для избранного синтаксиса.

В нижеприведенной таблице перечислены ключевые атрибуты класса «*attributeSchema*».

Табл. 2.2.2. Атрибуты класса

|  |  |
| --- | --- |
| **lDAPDisplayName** | **Description** |
| **cn** | Атрибут контекста именования. |
| [**lDAPDisplayName**](https://msdn.microsoft.com/en-us/library/windows/desktop/ms676828(v=vs.85).aspx) | Имя, используемое LDAP клиентами, чтобы обращаться к классу. Это имя должно быть уникальным. |
| [**schemaIDGUID**](https://msdn.microsoft.com/en-us/library/windows/desktop/ms679645(v=vs.85).aspx) | Один из уникальных идентификаторов класса. Сервер сам генерирует значение этого атрибута, если он не был указан при создании класса или атрибута. |
| [**adminDisplayName**](https://msdn.microsoft.com/en-us/library/windows/desktop/ms675214(v=vs.85).aspx) | Имя, отображаемое в средствах администрирования. |
| [**attributeID**](https://msdn.microsoft.com/en-us/library/windows/desktop/ms675234(v=vs.85).aspx) | Уникальный идентификатор атрибута. |
| [**attributeSecurityGUID**](https://msdn.microsoft.com/en-us/library/windows/desktop/ms675235(v=vs.85).aspx) | Атрибут для управления доступом. |
| [**attributeSyntax**](https://msdn.microsoft.com/en-us/library/windows/desktop/ms675236(v=vs.85).aspx) | Синтаксис атрибута. |
| [**oMSyntax**](https://msdn.microsoft.com/en-us/library/windows/desktop/ms679072(v=vs.85).aspx) | XDS представление синтаксиса атрибута. |
| [**oMObjectClass**](https://msdn.microsoft.com/en-us/library/windows/desktop/ms679071(v=vs.85).aspx) | Если oMSyntax равен 127, используется этот атрибут для определения объекта по умолчанию. |
| [**rangeLower**](https://msdn.microsoft.com/en-us/library/windows/desktop/ms679437(v=vs.85).aspx),[**rangeUpper**](https://msdn.microsoft.com/en-us/library/windows/desktop/ms679438(v=vs.85).aspx) | Определяют границы значения атрибута. |
| [**isSingleValued**](https://msdn.microsoft.com/en-us/library/windows/desktop/ms676811(v=vs.85).aspx) | Одно значение или несколько у атрибута. |
| [**searchFlags**](https://msdn.microsoft.com/en-us/library/windows/desktop/ms679765(v=vs.85).aspx) | Определяют поисковую информацию атрибута. |
| [**isMemberOfPartialAttributeSet**](https://msdn.microsoft.com/en-us/library/windows/desktop/ms676807(v=vs.85).aspx) | Определяет находится ли атрибут в глобальном каталоге. |
| [**linkID**](https://msdn.microsoft.com/en-us/library/windows/desktop/ms676831(v=vs.85).aspx) | Числовое значение, определяющее является ли атрибут связанным. Четное число – переднее связывание, нечетное – заднее. |
| [**systemFlags**](https://msdn.microsoft.com/en-us/library/windows/desktop/ms680022(v=vs.85).aspx) | Определяет дополнительные параметры атрибута. |
| [**systemOnly**](https://msdn.microsoft.com/en-us/library/windows/desktop/ms680025(v=vs.85).aspx) | Может изменяться только системой. |
| [**mAPIID**](https://msdn.microsoft.com/en-us/library/windows/desktop/ms676860(v=vs.85).aspx) | Числовое значение для клиентов MAPI. |
| [**isDefunct**](https://msdn.microsoft.com/en-us/library/windows/desktop/ms676800(v=vs.85).aspx) | Определяет активен или неактивен атрибут. |
| [**description**](https://msdn.microsoft.com/en-us/library/windows/desktop/ms675492(v=vs.85).aspx) | Описание. |
| [**objectClass**](https://msdn.microsoft.com/en-us/library/windows/desktop/ms679012(v=vs.85).aspx) | Определяет экземпляром какого класса является данный атрибут. |

В Active Directory определено всего 23 типа синтаксиса атрибутов. Ниже перечислены основные (значения SyntaxID и OM ID необходимы для модифицирования схемы, а именно добавления новых атрибутов):

* Boolean: Syntax ID - 2.5.5.8, OM ID - 1
* Enumeration: Syntax ID - 2.5.5.9, OM ID - 10
* Integer: Syntax ID - 2.5.5.9, OM ID - 2
* Interval: Syntax ID - 2.5.5.16, OM ID - 65
* Object(Access-Point): Syntax ID - 2.5.5.14, OM ID - 127
* Object(DN-Binary): Syntax ID - 2.5.5.7, OM ID - 127
* Object(DN-String): Syntax ID - 2.5.5.14, OM ID - 127
* Object(DS-DN): Syntax ID - 2.5.5.1, OM ID - 127
* Object(OR-Name): Syntax ID - 2.5.5.7, OM ID - 127
* String(Case Sensitive) : Syntax ID - 2.5.5.3, OM ID - 27
* String(Generalized-Time) : Syntax ID - 2.5.5.11, OM ID - 24
* String(IA5) : Syntax ID - 2.5.5.5, OM ID - 22
* String(NT-Sec-Desc) : Syntax ID - 2.5.5.15, OM ID - 66
* String(Numeric) : Syntax ID - 2.5.5.6 18
* String(Object-Identifier) : Syntax ID - 2.5.5.2, OM ID - 6
* String(Octet) : Syntax ID - 2.5.5.10, OM ID - 4
* String(Printable) : Syntax ID - 2.5.5.5, OM ID - 19
* String(Sid) : Syntax ID - 2.5.5.17, OM ID - 4
* String(Teletex) : Syntax ID - 2.5.5.4, OM ID - 20
* String(Unicode) : Syntax ID - 2.5.5.12, OM ID - 64
* String(UTC-Time) : Syntax ID - 2.5.5.11, OM ID – 23

## 2.3. Обзор методов управления схемой Active Directory

### 2.3.1. Lightweight Directory Access Protocol

**LDAP** (англ. *Lightweight Directory Access Protocol* — «*облегчённый протокол доступа к каталогам*») — протокол прикладного уровня для доступа к службе каталогов X.500, разработанный IETF как облегчённый вариант разработанного ITU-T протокола DAP. LDAP — относительно простой протокол, использующий TCP/IP и позволяющий производить операции аутентификации (*bind*), поиска (*search*) и сравнения (*compare*), а также операции добавления, изменения или удаления записей. Обычно LDAP-сервер принимает входящие соединения на порт 389 по протоколам TCP или UDP. Для LDAP-сеансов, инкапсулированных в SSL, обычно используется порт 636.

Всякая запись в каталоге LDAP состоит из одного или нескольких *атрибутов* и обладает *уникальным именем* (DN — англ. *Distinguished Name*). Уникальное имя может выглядеть, например, следующим образом: «*cn=Иван Петров, ou=Сотрудники, dc=example, dc=com*». Уникальное имя состоит из одного или нескольких *относительных уникальных имен* (RDN — англ. *Relative Distinguished Name*), разделённых запятой. Относительное уникальное имя имеет вид «*ИмяАтрибута=значение*». На одном уровне каталога не может существовать двух записей с одинаковыми относительными уникальными именами. В силу такой структуры уникального имени записи в каталоге LDAP можно легко представить в виде дерева.

Запись может состоять только из тех атрибутов, которые определены в описании класса записи (*object class*), которые, в свою очередь, объединены в схемы (*schema*). В схеме определено, какие атрибуты являются для данного класса обязательными, а какие — необязательными. Также схема определяет тип и правила сравнения атрибутов. Каждый атрибут записи может хранить несколько значений.

Возможности LDAP описаны в 4 моделях:

* Информационная модель определят, каким образом информация или данные представлены в системе LDAP.
* Модель именования определяет, каким образом данные организованы.
* Функциональная модель определяет, что можно сделать (чтение, поиск, запись и т.д.).
* Модель безопасности определяет то, как информация защищена.

Стандартизация поведения LDAP с помощью этих моделей делает LDAP предпочтительным протоколом доступа к службам каталогов.

В протоколе LDAP определены следующие операции для работы с Каталогом:

* Операции подключения/отключения
  + Подключение (*bind*) — позволяет ассоциировать клиента с определённым объектом Каталога (фактическим или виртуальным) для осуществления контроля доступа для всех прочих операций чтения/записи. Для того, чтобы работать с Каталогом, клиент обязан пройти аутентификацию как объект, отличительное имя (Distinguished Name) которого находится в пространстве имён, описываемом Каталогом. В запросе операции *bind* клиент может не указывать отличительное имя, в таком случае будет осуществлено подключение под специальным псевдонимом *anonymous* (обычно это что-то наподобие гостевой учетной записи с минимальными правами)
  + Отключение (*unbind*) — позволяет клиенту в рамках сеанса соединения с LDAP-сервером переключиться на аутентификацию с новым отличительным именем. Команда unbind возможна только после аутентификации на сервере с использованием bind, в противном случае вызов unbind возвращает ошибку
  + Поиск (*search*) — чтение данных из Каталога. Операция сложная, на вход принимает множество параметров, среди которых основными являются:
  + База поиска (*baseDN*) — ветка DIT, от которой начинается поиск данных
  + Глубина поиска (*scope*) — может иметь значения (в порядке увеличения охватываемой области): base, one, sub
    - base — поиск непосредственно в узле — базе поиска
    - one — поиск по всем узлам, являющимся прямыми потомками базового в иерархии, то есть лежащим на один уровень ниже него
    - sub — поиск по всей области, нижележащей относительно базы поиска (baseDN)
  + Фильтр поиска (*searchFilter*) — это выражение, определяющее критерии отбора объектов каталога, попадающих в область поиска, задаваемую параметром scope. Выражение фильтра поиска записывается в польской (префиксной) нотации, состоящей из логических (булевых) операторов и операндов, в свою очередь являющихся внутренними операторами сопоставления значений атрибутов LDAP (в левой части) с выражениями (в правой части) с использованием знака равенства.

Логические операторы представлены стандартным «набором»: **&** (логическое «И»), **|** (логическое «ИЛИ») и **!** (логическое «НЕ»).

Пример фильтра поиска:

(&(!(entryDN:dnSubtreeMatch:=dc=Piter,dc=Russia,ou=People,dc=example,dc=com))(objectClass=sambaSamAccount)  
(|(sn=Lazar\*)(uid=Nakhims\*)))

* Операции модификации — позволяют изменять данные в Каталоге, при этом в понятие модификации входит как добавление, удаление и перемещение записей целиком, так и редактирование записей на уровне их атрибутов. Подтипы модификации:
  + Добавление (*add*) — добавление новой записи
  + Удаление (*delete*) — удаление записи
  + Модификация RDN (*modrdn*) — перемещение/копирование записи
  + Модификация записи (*modify*) — позволяет редактировать запись на уровне её атрибутов,
    - добавляя новый атрибут или новое значение многозначного атрибута (add)
    - удаляя атрибут со всеми его значениями (delete)
    - заменяя одно значение атрибута на другое (replace)
    - а также увеличивая (уменьшая) значение атрибута в рамках атомарной операции (increment)
  + Операция сравнения (*compare*) — позволяет для определённого отличительного имени сравнить выбранный атрибут с заданным значением

В стандарте LDAP определена специальная операция, позволяющая клиентам получать информацию о поддерживаемых сервером версиях протокола и возможностях LDAP-сервера. Эта команда является надстройкой (расширением) для операции *search* и выполняется при следующем сочетании параметров последней:

* BIND анонимный
* База поиска *baseDN* указана как **""** (пустая строка)
* Глубина поиска *scope* указана как **base**
* Фильтр поиска: **(objectClass=\*)**
* Перечень запрашиваемых атрибутов: либо явное перечисление, либо «**+**».

Active Directory поддерживает LDAP v2 и LDAP v3. Ниже перечислены особенности 3 версии:

* Можно указать элементы управления (как на сервере, так и на клиенте), которые расширяют функциональные возможности операции LDAP.
* Можно сделать запрос сервер для выполнения расширенных операций (помимо стандартных операций LDAP).
* Повышенная безопасность при использовании Simple Authentication Security Layer (SASL)
* Возможность модифицирования схемы.
* И клиентская, и серверная часть поддерживают формат UTF-8. Можно отправлять и принимать данные на любом языке. Active Directory посылает все ответы в этом формате.
* Операционные атрибуты, которые служба каталогов использует для собственных нужд.
* Сервер может направить клиента на другие сервера.

LDAP v3 совместим с LDAP v2. Есть только одно условие: клиент LDAP v2 должен иметь возможность подключиться к серверу.

### 2.3.2. Active Directory Service Interfaces

**ADSI** (от англ. *Active Directory Service Interfaces* — «интерфейсы службы Active Directory») — интерфейс программирования приложений, разработанный компанией Microsoft и предназначенный для доступа к различным службам каталогов, в первую очередь к Active Directory. ADSI позволяет создавать, изменять и удалять объекты в каталогах, выполнять поиск и т. д.

Интерфейсы Microsoft Active Directory Service Interfaces (ADSI) представляют собой модель службы каталогов на основе модели компонентных объектов (COM), позволяющую клиентским приложениям, совместимым с ADSI, получать доступ к широкому кругу разных протоколов каталогов, включая службу каталогов Windows®, LDAP и NDS, используя всего один стандартный набор интерфейсов. ADSI ограждает клиентское приложение от реализации и вмешательства в работу базового хранилища данных или протокола.

Приложение, называемое поставщиком ADSI, делает себя доступным для клиентских приложений ADSI. Данные, предоставляемые поставщиком, организованы в специальное пространство имен, определенное поставщиком. Вдобавок к реализации интерфейсов, определенных ADSI, поставщик может реализовывать схему ADSI. Схема используется для предоставления метаданных о структуре пространства имен и объектов, предоставляемых поставщиком ADSI.

Имена объектов ADSI называются строками связывания (Binding String) или строками ADsPath, которые состоят из двух частей. Первая часть имени определяет, к какой именно службе каталогов (или провайдеру ADSI) мы обращаемся:

Табл. 2.3.1. Описание провайдеров

|  |  |
| --- | --- |
| **Обращение** | **Описание** |
| LDAP:// | Для службы каталогов, созданной на основе протокола LDAP (Lightweight Directory Access Protocol, упрощённый протокол для доступа к каталогу), в том числе для Active Directory в Windows 2000/2003. |
| WinNT:// | Для службы каталогов в сети Windows NT 4.0 или на локальной рабочей станции Windows XP/2000. |
| NDS:// | Для службы каталогов NetWare NDS (Novell Directory Service). |
| NWCOMPAT:// | Для службы каталогов NetWare Bindery. |

Вторая часть строки ADsPath определяет расположение объекта в конкретном каталоге (для каждого провайдера ADSI - по-своему).

ADSI может выступать в роли провайдера OLE DB, что позволяет с помощью ADO (ActiveX Data Object) выполнять "естественные" запросы к пространству имён службы каталога (провайдер "*ADsDSOObject*"). Использовать ADSI в качестве провайдера OLE DB можно в запросах к пространствам имён LDAP и NDS.

### 2.3.3. Определение способа управления схемой

Как написано выше, протокол LDAP является провайдером ADSI. Это означает, что ADSI является «оболочкой» LDAP API. Их зависимость показана ниже (рис.2.3.1.).

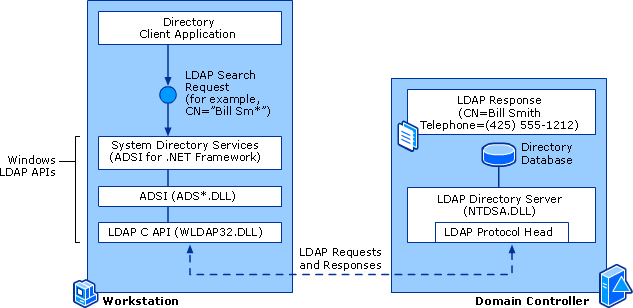


Рис. 2.3.1. Windows LDAP APIs

При написании программы мы будем использовать родной LDAP API (wldap32.dll). Все остальные технологии работают именно с этим протоколом и являются более тяжеловесными и избыточными, так как они были разработаны, чтобы упростить процесс программного взаимодействия со службами каталогов.

## 2.4. Обзор существующих решений для управления схемой

### 2.4.1. Стандартные утилиты Microsoft

**CSVDE**

Средство CSVDE позволяет импортировать новые объекты в Active Directory, используя исходный CSV-файл; оно также дает возможность экспортировать существующие объекты в файл CSV. CSVDE нельзя использовать для изменения существующих объектов; при использовании этого средства в режиме импорта можно лишь создавать новые объекты.

Экспорт списка существующих объектов с помощью CSVDE довольно прост. Ниже показано, как экспортировать объекты Active Directory в файл под названием ad.csv:

**csvde –f ad.csv**

Параметр –f указывает, что за ним следует имя выходного файла. Но следует понимать, что, в зависимости от среды, этот базовый синтаксис может привести к выводу огромного и неудобного файла. Чтобы ограничить средство экспортом лишь объектов внутри определенного структурного подразделения (OU), команду можно изменить следующим образом:

**csvde –f UsersOU.csv –d ou=Users,dc=contoso,dc=com**

Предположим далее, что необходимо экспортировать лишь объекты пользователя в файл CSV. В таком случае можно добавить параметр –r, позволяющий указать фильтр протокола LDAP для данного поиска, который ограничит число экспортируемых атрибутов:

**сsvde –f UsersOnly.csv –d ou=Users,dc=contoso,dc=com –r  
“(&(objectcategory=person)(objectclass=user))” –l  
DN,objectClass,description**

Параметр –i позволяет импортировать объекты в Active Directory из исходного файла CSV. Однако создание объектов пользователя с помощью CSVDE имеет один важный недостаток: с помощью этого средства нельзя устанавливать пароли пользователей.

**LDIFDE**

Active Directory предоставляет второе встроенное средство для пакетных операций пользователей, именуемое LDIFDE и обладающее более широкими и гибкими возможностями, чем CSVDE. Помимо создания новых объектов, LDIFDE позволяет модифицировать и удалять существующие объекты и даже расширять схему Active Directory. Платой за гибкость LDIFDE является то, что необходимый входной файл (файл LDIF) с расширением .ldf использует более сложный формат, чем простой файл CSV.

Начнем с простого примера — экспорта пользователей в структурном подразделении в файл LDF:

**ldifde -f users.ldf -s DC1.contoso.com -d “ou=UsersOU,dc=contoso,dc=com”  
–r “(&(objectcategory=person)(objectclass=user))”**

Как и в случае большинства средств командной строки, полное описание параметров LDIFDE можно получить, запустив команду LDIFDE /?. На Рис. 2.4.1 показаны те, что были использованы для составления запроса.

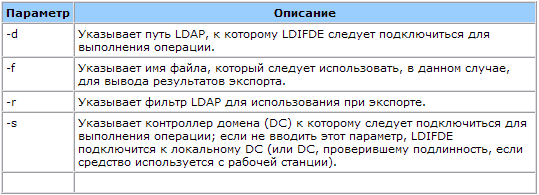


Рис. 2.4.. Список параметров и их описание

По-настоящему возможности LDIFDE раскрываются при создании объектов и управлении ими. Однако перед этим необходимо создать входной файл. Нижеследующий код создает две новых учетных записи пользователя — afuller и rking; для создания входного файла введите текст в блокноте (или другом редакторе открытого текста) и сохраните его как NewUsers.ldf:

**dn**: CN=afuller, OU=UsersOU, DC=contoso, DC=com  
**changetype: add**  
**cn: afuller**  
**objectClass: user**  
**samAccountName: afuller**

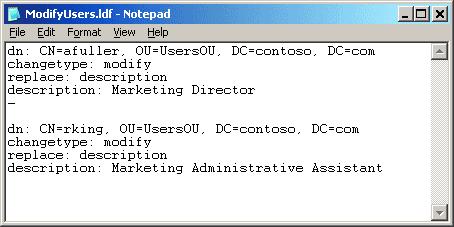
**dn: CN=rking, OU=UsersOU, DC=contoso, DC=com**  
**changetype: add**  
**cn: rking**  
**objectClass: user**  
**samAccountName: rking**

После того как создание файла завершено, запустите следующую команду:

**ldifde –i –f NewUsers.ldf –s DC1.contoso.com**

Единственный новый параметр здесь — это -i, который, как несложно догадаться, указывает, что выполняется операция импорта, а не экспорта.

При модификации или удалении существующих объектов синтаксис команды LDIFDE не меняется; вместо этого изменяется содержимое файла LDF. Для изменения поля описания учетных записей пользователей создайте текстовый файл, именуемый ModifyUsers.ldf, такой как показано на Рис. 2.4.2.



**Рис. 2.4.2. Файл LDF ModifyUsers**

Изменения импортируются путем запуска того же синтаксиса команды LDIFDE, что и раньше, с указанием нового файла LDF после параметров -f. Формат LDF для удаления объектов еще проще; для удаления пользователей создается файл, именуемый DeleteUsers.ldf, и вводится следующее:

**dn: CN=afuller OU=UsersOU, DC=contoso, DC=com**  
**changetype: delete**

**dn: CN=rking, OU=UsersOU, DC=contoso, DC=com**  
**changetype: delete**

В отличие от CSVDE, LDIFDE может настраивать пароли пользователей. Однако перед настройкой атрибута unicodePWD для учетной записи пользователя необходимо настроить шифрование SSL/TLS на контроллерах домена.

Вдобавок, LDIFDE может создавать и модифицировать любые объекты Active Directory, а не только учетные записи пользователей. Например, нижеследующий файл LDF создаст новое расширение схемы, именуемое EmployeeID-example, в схеме леса contoso.com:

**dn: cn=EmployeeID-example,cn=Schema,**  
**cn=Configuration,dc=contoso,dc=com  
changetype: add  
adminDisplayName: EmployeeID-Example  
attributeID: 1.2.3.4.5.6.6.6.7  
attributeSyntax: 2.5.5.6  
cn: Employee-ID  
instanceType: 4  
isSingleValued: True  
lDAPDisplayName: employeeID-example**

Поскольку в файлах LDIFDE используется стандартный отраслевой формат файла LDAP, приложения от сторонних производителей, которым необходимо модифицировать схему Active Directory, часто поставляют файлы LDF, с помощью которых можно изучить и одобрить изменения, прежде чем применять их к производственной среде.

**Оснастка «Схема Active Directory»**

Оснастка консоли управления (MMC) "*Схема Active Directory*" используется для просмотра объектов схемы доменных служб Active Directory (AD DS) и управления ими. Требования к установке и использованию оснастки "Схема Active Directory" более строгие по сравнению с другими средствами администрирования. Для внесения изменений в оснастку "Схема Active Directory" необходимо быть членом группы безопасности "Администраторы схемы" в корневом домене леса Active Directory. По умолчанию в группу "Администраторы схемы" входит только учетная запись администратора для корневого домена леса.

Оснастка "*Схема Active Directory*" служит для просмотра определений классов и объектов, обеспечения административного доступа к схеме и позволяет изменять классы и атрибуты схемы.

Оснастка "*Схема Active Directory*" также может быть использована для просмотра объектов схемы служб Active Directory, облегченного доступа к каталогам (AD LDS) и управления ими.

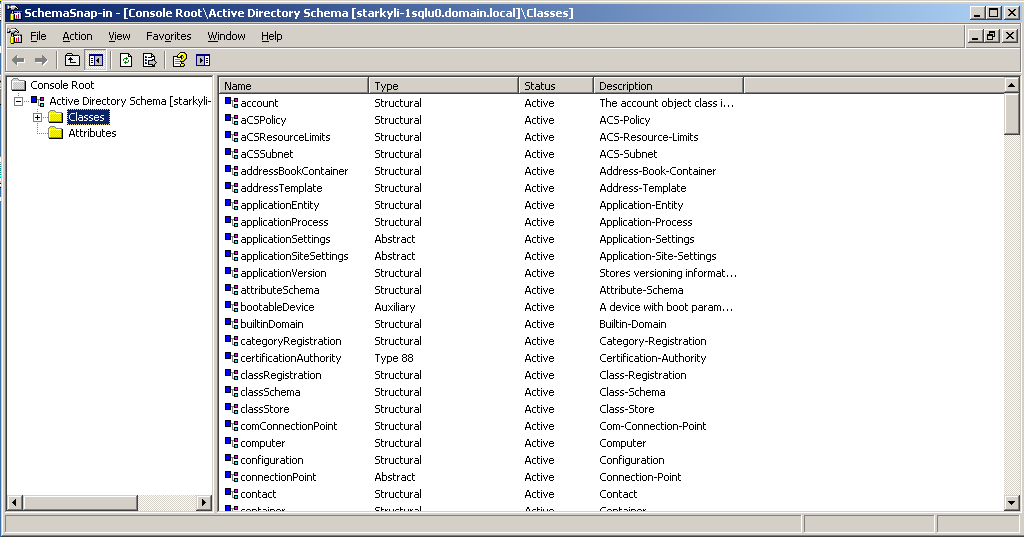


Рис. 2.4.3. Интерфейс оснастки

Однако, у схемы есть следующие недостатки:

1. Она может быть запущена только на сервере.

2. Можно изменить только заданные в свойствах атрибуты (рис.2.4.4.), а список всех атрибутов доступен только для чтения и то только у объектов классов схемы (рис.2.4.5.).

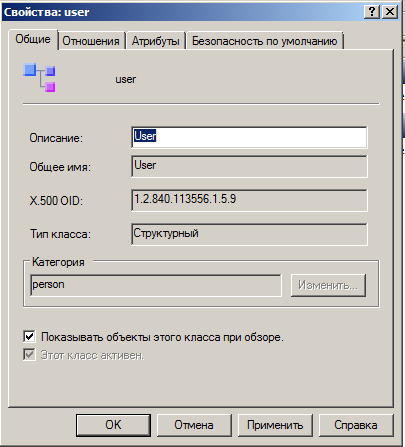


Рис. 2.4.4. Свойства класса user

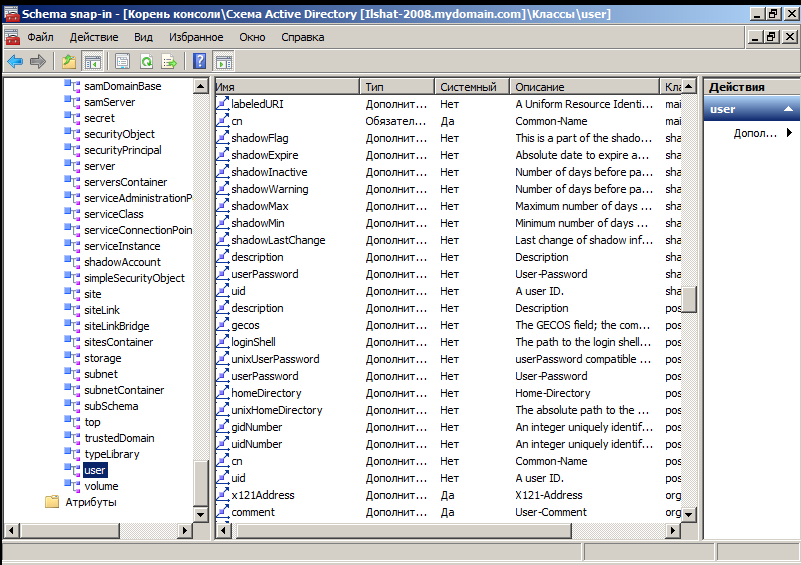


Рис. 2.4.5. Список атрибутов у класса user

### 2.4.2. Программа LdapAdmin

**LdapAdmin** – это бесплатный административный инструмент для управления службой каталогов. Приложение позволяет смотреть, искать, изменять, создавать и удалять объекты LDAP сервера. Программа также может выполнять более трудоемкие операции, такие как копирование каталога и перемещение между удаленными серверами.

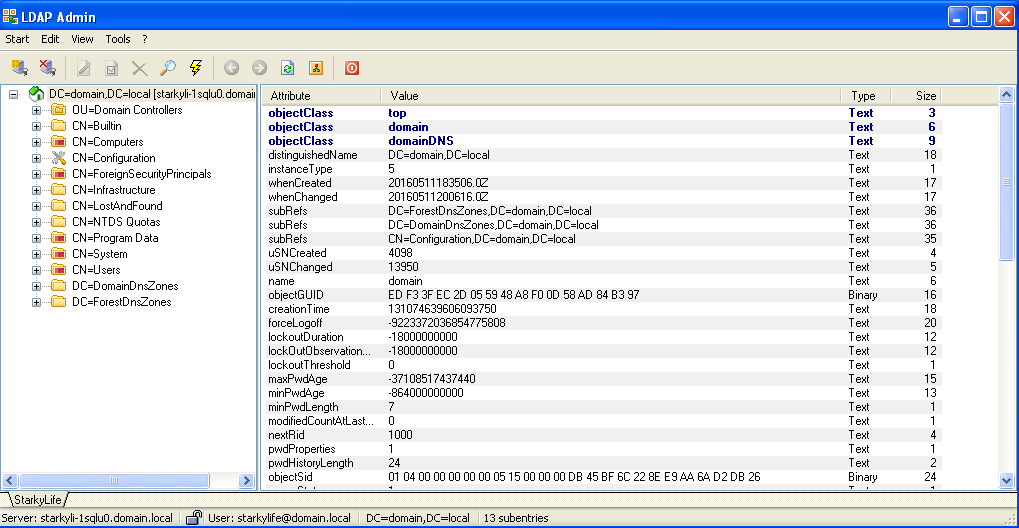


Рис. 2.4.6. Интерфейс LdapAdmin

LdapAdmin является LDAP-браузером. Это означает, что он предназначен для работы со всеми элементами LDAP-каталога. Чтобы получить представление об объекте, необходимо открыть его список атрибутов и понимать, что к чему относится.

# Описание программного комплекса

Конечными пользователями продукта являются администраторы службы каталогов. Программа предоставляет им интуитивно-понятный графический интерфейс и необходимый функционал для управления схемой Active Directory.

В виду того, что программный комплекс модульный, разработчики программного обеспечения, которым при разработке программ приходится работать со службами каталогов Microsoft, могут использовать готовые классы для взаимодействия со схемой.

Преимуществом программного продукта является то, что он не занимает много места и потребляет малый объем оперативной памяти.

## Основные технические требования

Требования относятся к машине, на которой запускается программный комплекс.

**Требования к оперативной памяти**. Размер используемой оперативной памяти зависит от размеров схемы службы каталогов.

Минимальные требования к оперативной памяти – 32 МБ. Рекомендуемые – 128 МБ.

**Требования к операционной системе**. Windows XP, Windows Server 2003, Windows Vista, Windows 7 x32.

**Требования к операционной системе владельца службы каталогов.** Windows Server 2000, Windows Server 2003 R2, Windows Server 2008 R2, Windows Server 2012 R2.

**Аппаратные требования**.

Минимальные требования к аппаратной конфигурации компьютера: Intel Pentium IV c тактовой частотой не менее 2 Гц.

Требования к свободному месту на жестком диске несущественны, так как папка с программой весит всего 1.62 МБ и процессе работы не создает дополнительные файлы.

**Необходимые условия для работы с программой**.

Для работы с программой рабочая станция должна быть подключена к серверу доменных служб.

Подключенный пользователь должен входить в группу «*Администраторы схемы*», иначе программа не сможет работать со схемой.

## Содержимое рабочего каталога программы

Рабочим каталогом называется каталог, в котором находится программа. В этом каталоге содержится пусковой файл SchemaAdmin.exe и директория с внешними ресурсами Icons, в котором содержится весь графический материал для интерфейса.

## Руководство для пользователя

При запуске программы открывается главное окно (рис. 3.3.1.).

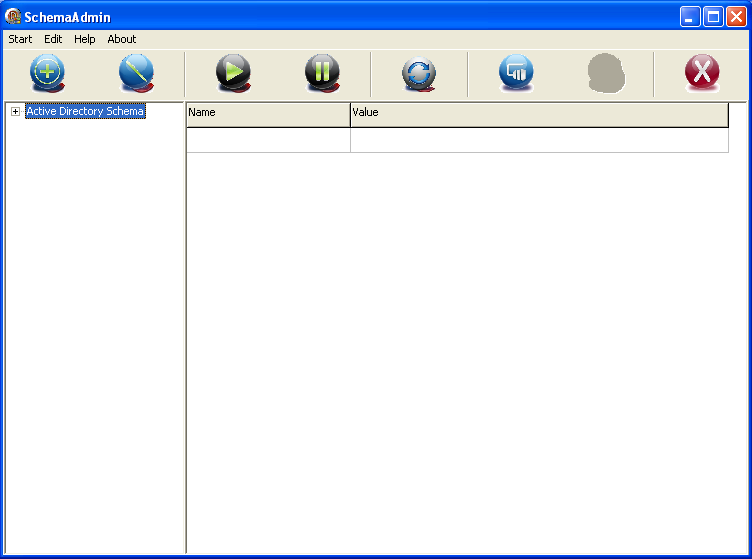


Рис. 3.3.. Главное окно

Чтобы подключиться нужно нажать кнопку . Откроется окно подключения (рис. 3.3.2.).

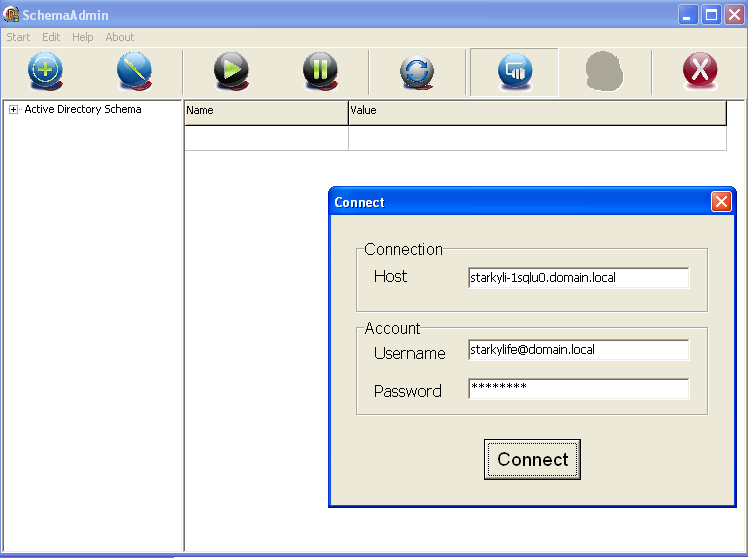


Рис. 3.3.. Окно соединения

В открывшемся окне необходимо ввести имя хоста или DNS адрес владельца доменных служб.

После нажатия «*Connect*» программа загрузит содержимое схемы.

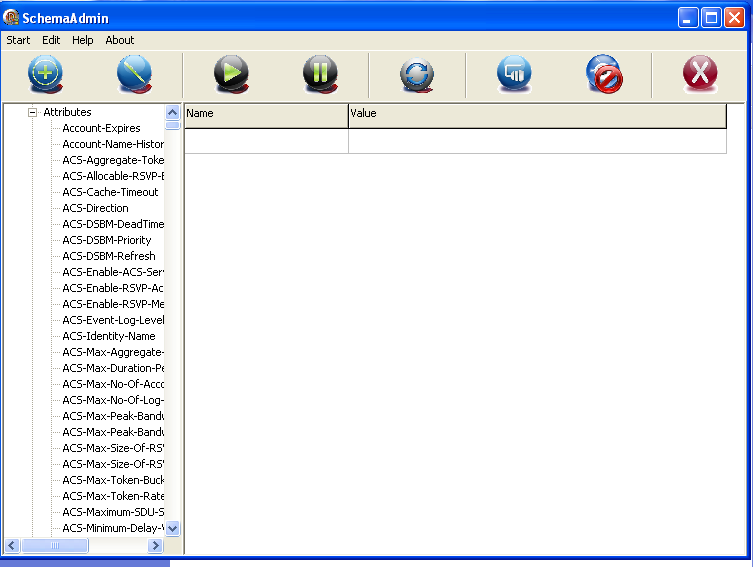


Рис. 3.3.. Главное окно с загруженными данными

Кнопка «Disconnect»  станет активной.

Наряду со списками всех атрибутов и классов сформируется и дерево классов, показывающее их структуру взаимодействия.

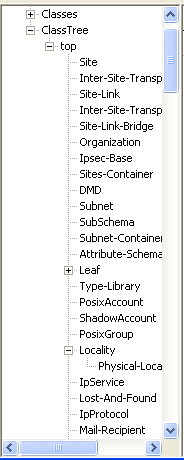


Рис. 3.3.. Дерево классов

При выборе элемента элемента «*Attributes*» таблица слева заполнится всеми атрибутами и их главными свойствами.

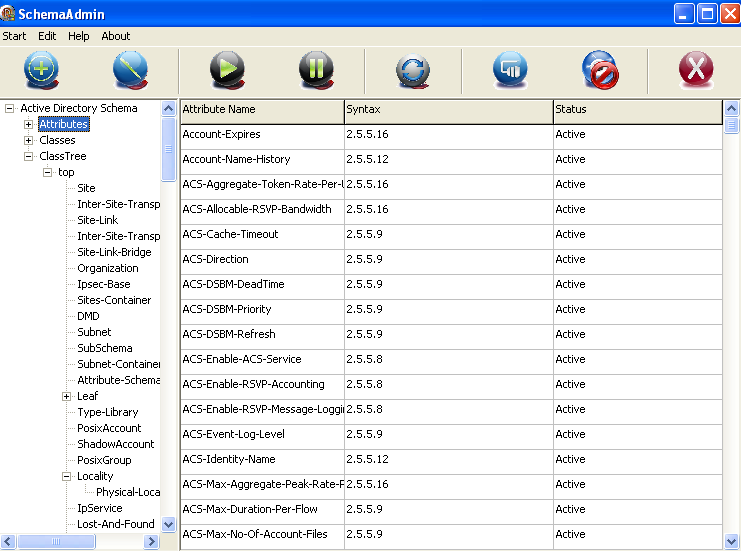


Рис. 3.3.. Вывод атрибутов

При выборе поддерева «*Classes*» таблица заполнится всеми доступными классами.

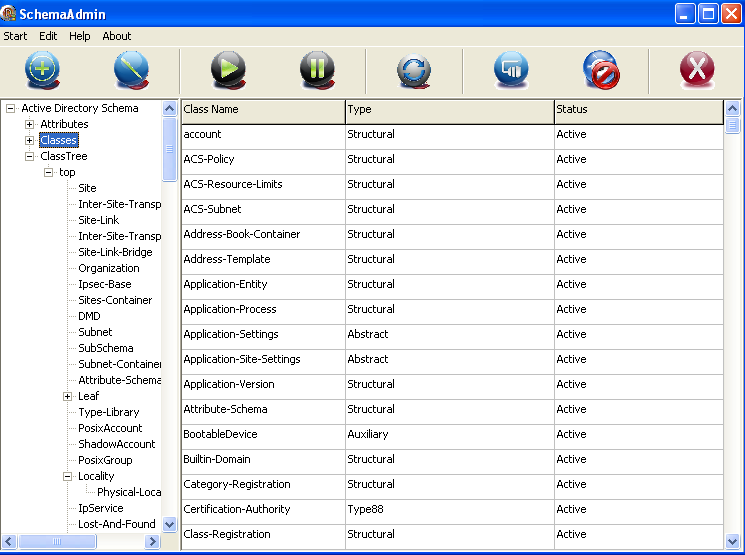


Рис. 3.3.. Вывод классов

Для просмотра и редактирования содержимого объекта класса или атрибута необходимо выбрать его.

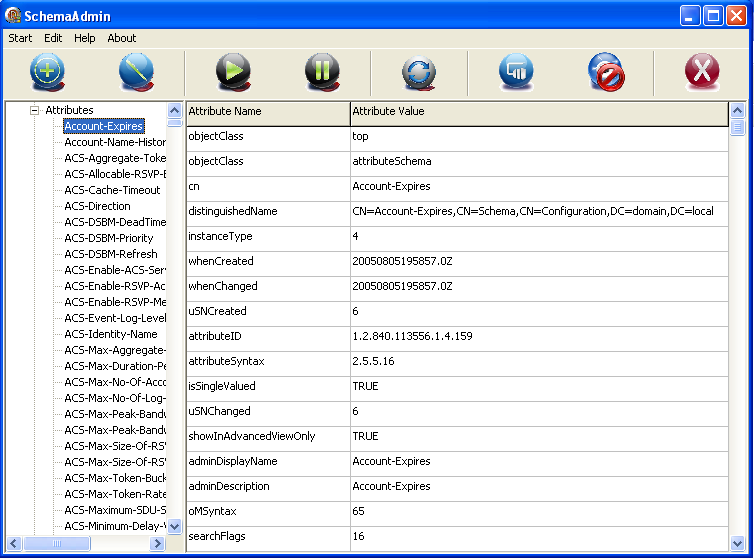


Рис. 3.3.. Сведения выбранного объекта

Столбец «*Attribute Value*» можно редактировать. Для добавления или удаления атрибутов необходимо использовать кнопку Modify.

При нажатии кнопки добавить в пункте меню или на верхней панели, откроется окно добавления (рис. 3.3.8.).

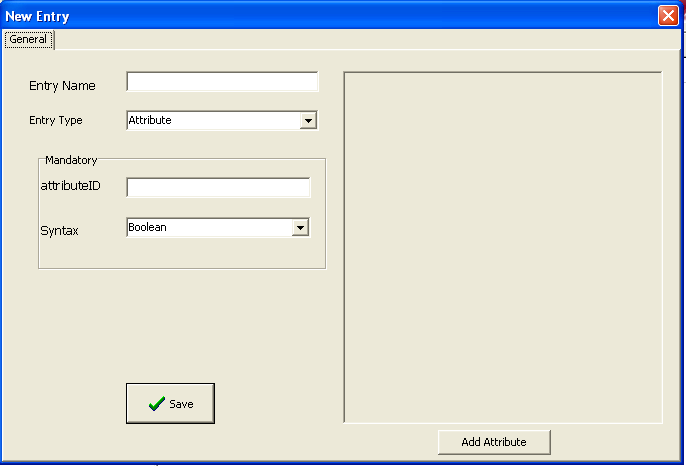


Рис. 3.3.. Окно добавления атрибута

Так выглядит окно добавления атрибута (рис.3.3.8). Если вместо атрибута выбрать класс, то изменится список обязательных атрибутов и добавится дополнительная вкладка, где можно проставить, какие атрибуты можно добавить в объекты этого класса (рис.3.3.9).

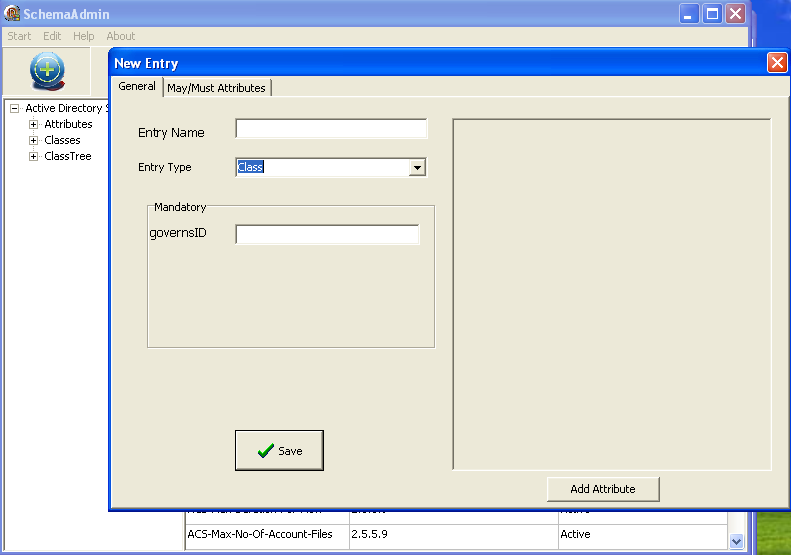


Рис. 3.3.. Окно добавления класса

При нажатии Add Attribute добавится строчка для нового атрибута.

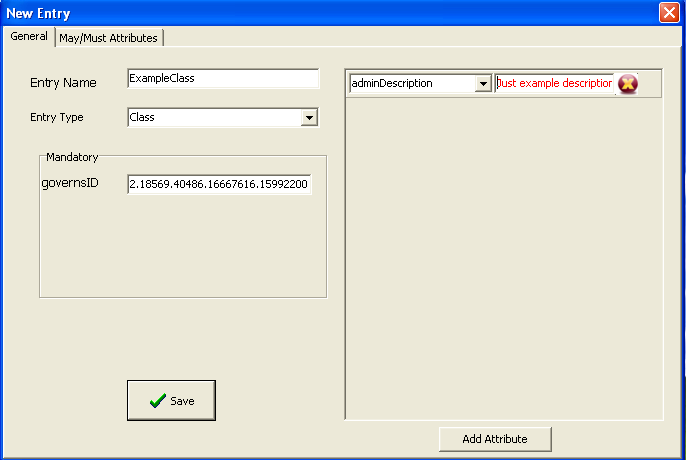


Рис. 3.3.. Пример добавления дополнительного атрибута

При нажатии на рисунок удаления, строчка удалится.

Во вкладке May/Must Attributes можно ввести обязательные и необязательные атрибуты для объектов этого класса (рис. 3.3.11.).

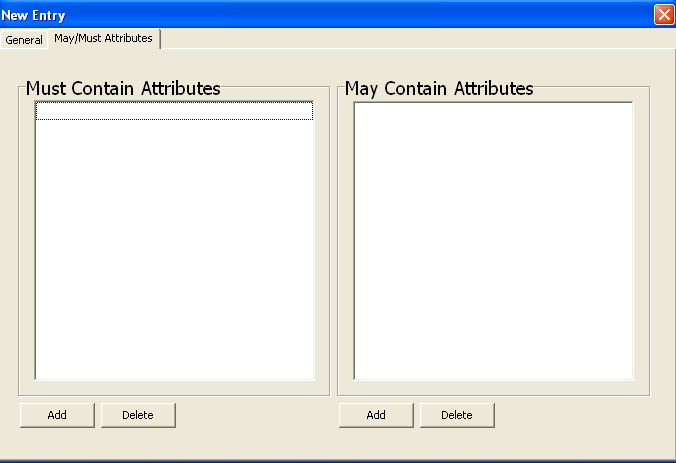


Рис. 3.3.. Вкладка возможных атрибутов

Для добавления новых элементов нажмите кнопку «Add», для удаления «Delete».

При нажатии кнопки добавления откроется окно с выбором всех доступных атрибутов (рис. 3.3.12.).

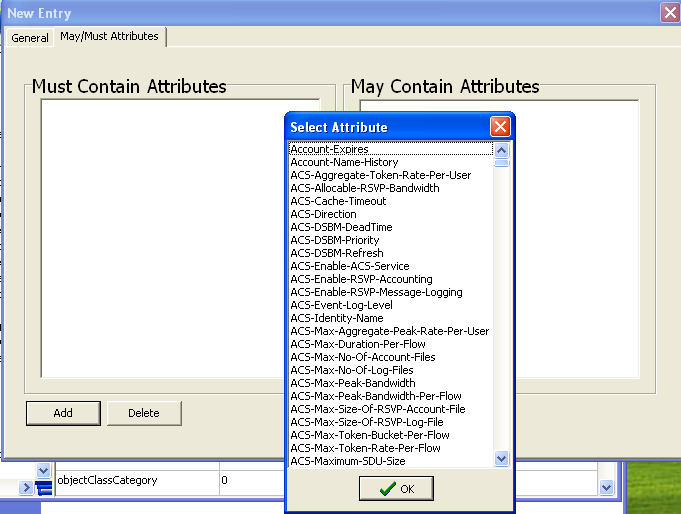


Рис. 3.3.. Окно выбора атрибута

Для создания нового класса или атрибута, необходимо заполнить все обязательные поля и нажать Save.

После объект появится в списке в главном окне.

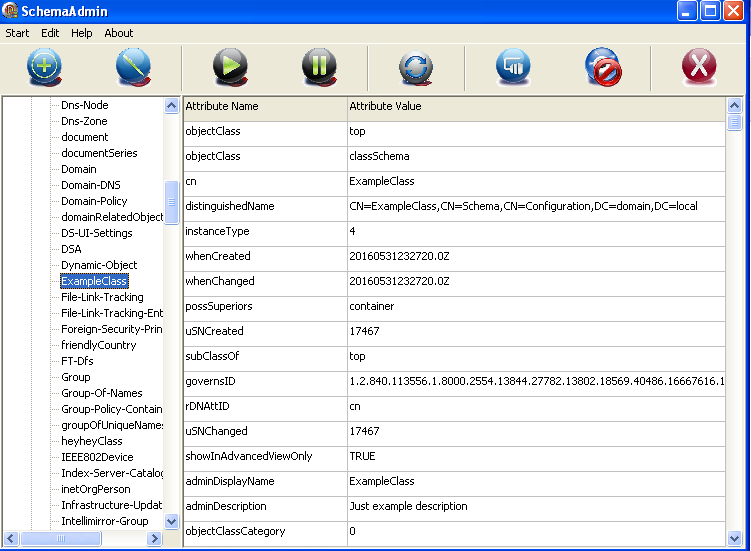


Рис. 3.3.. Вывод добавленного объекта

Для редактирования элементов схемы необходимо нажать кнопку Modify. Откроется окно изменения объекта (рис. 3.3.14.).

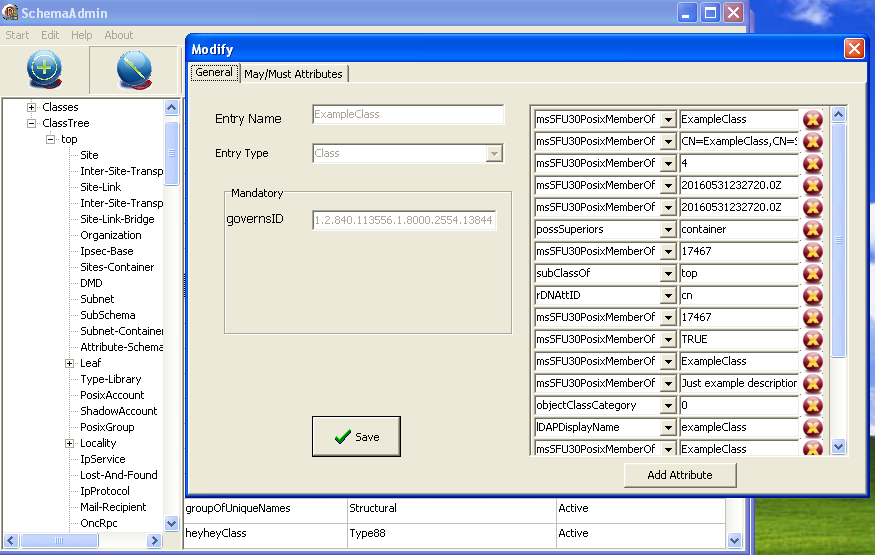


Рис. 3.3.. Окно изменения объекта

Данные выделенные сером изменять нельзя.

Для активации выбранного элемента схемы необходимо нажать кнопку  или выбрать соответствующий пункт меню. Для деактивации - .

Посмотреть статус можно в таблице в главном окне.

До нажатия кнопки деактивации.

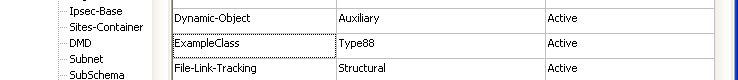


Рис. 3.3.. Пример активации/деактивации элемента схемы (до)

После нажатия кнопки деактивации.



Рис. 3.3.. Пример активации/деактивации элемента схемы (после)

Кнопка  обновляет содержимое схемы.

Чтобы закрыть программу необходимо нажать кнопку .

Все функции верхней панели продублированы в меню программы (рис.3.3.17-3.3.18.).

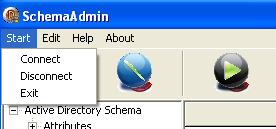


Рис. 3.3.. Меню «Start»

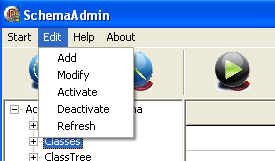


Рис. 3.3.. Меню «Edit»

## Руководство для программиста

В данном разделе приводится описание программного комплекса с точки зрения руководства для программиста.

### Структура программного комплекса

Программа была разработана в среде разработки Borland Developer Studio 2006 на языке программирования Object Pascal.

Для разработки графического интерфейса использовались VCL формы.

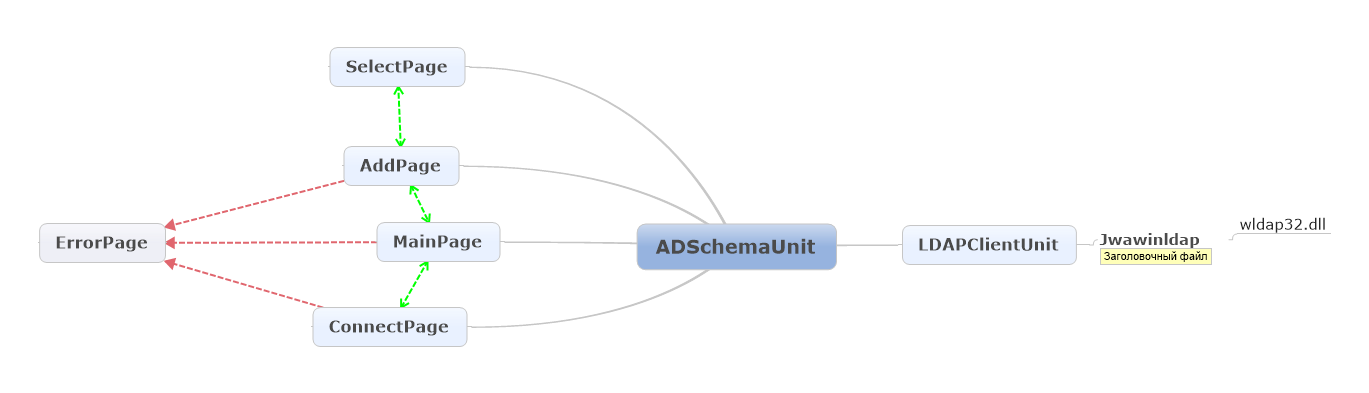
****

Рис. 3.4.. Структура программы

### Описание используемых функций LDAP API

**Функция ldap\_init**

Функция ldap\_init [1] инициализирует сеанс с сервером LDAP.

function ldap\_init(

HostName : PCHAR;

PortNumber : ULONG

):PLDAP;

**ПАРАМЕТРЫ:**

**HostName**

Имя домена или имя/IP адрес хоста, с которым должно произойти соединение;

**PortNumber**

Содержит TCP порт, по которому происходит соединение (по умолчанию 389). Этот параметр будет проигнорирован, если имя хоста включает номер порта.

**ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ:**

В том случае, если функция выполнилась успешно, то в результате будет возвращен указатель на структуру LDAP.

Если функция неуспешна, то будет возвращен nil. Для получения кода ошибки используется LdapGetLastError,

**Функция ldap\_set\_option**

Функция ldap\_set\_option устанавливает опции в блоках связи.

function ldap\_set\_option(

ld : pLdap;

option : integer;

invalue : Pointer

):ULONG;

**ПАРАМЕТРЫ:**

**ld**

Указатель на сессию.

**option**

Имя устанавливаемой опции.

**invalue**

Указатель на значение опции, которое должно быть установлено. Константы LDAP\_OPT\_ON и LDAP\_OPT\_OFF предоставляют возможность включить или выключить установки опции.

**ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ:**

Если функция успешна, то она возвращает LDAP\_SUCCESS. В противном случае возвращается код ошибки.

**Функция ldap\_simple\_bind\_s**

Функция ldap\_simple\_bind\_s производит аутентификацию клиента на сервере с помощью пароля.

function ldap\_simple\_bind\_s(

ld : PLDAP;

dn : PCHAR;

passwd : PCHAR

):ULONG;

**ПАРАМЕТРЫ:**

**ld**

Указатель на сессию.

**dn**

Distinguished name пользователя. Операция bind использует параметры *dn* и *passwd* для аутентификации пользователя.

**passwd**

Пароль пользователя, определенного в параметре *dn*.

**ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ:**

Если функция успешна, то будет возвращено сообщение ID запрашиваемой операции.

Если функция неуспешна, она возвращает -1 и устанавливает сеансовые параметры ошибки в структуре данных LDAP.

**Функция ldap\_unbind**

Функция ldap\_unbind освобождает все ресурсы связанные c сеансом LDAP.

function ldap\_unbind(

ld:PLDAP

):ULONG;

**ПАРАМЕТРЫ:**

**ld**

Указатель на сессию.

**ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ:**

Если функция успешна, то она возвращает LDAP\_SUCCESS. В противном случае возвращается код ошибки.

**Функция ldap\_count\_entries**

Функция ldap\_count\_entries возвращает количество записей, которые были возвращены сервером в результате поиска.

function ldap\_count\_entries(

ld : PLDAP;

res : PLDAPMessage

):ULONG;

**ПАРАМЕТРЫ:**

**ld**

Указатель на сессию.

**res**

Результат поиска, полученный при вызове одной из синхронных команд поиска.

**ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ:**

Если функция успешна, то она возвращает количество записей, полученных в результате поиска.

Если функция неуспешна, то возвращается –1, и функция устанавливает параметры ошибки сеанса в структуре LDAP.Ldap\_first\_entry

**Функция ldap\_first\_entry**

Функция ldap\_first\_entry возвращает первый элемент коллекции найденных объектов. Если коллекция пустая, возвращает nil.

function ldap\_first\_entry(

ld : PLDAP;

res : PLDAPMessage

):PLDAPMessage;

**ПАРАМЕТРЫ:**

**ld**

Указатель на сессию.

**res**

Результат поиска, полученный при вызове одной из синхронных команд поиска.

**ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ:**

Если поиск удачен, то функция возвращает указатель на первый элемент коллекции найденных объектов. В противном случае (если коллекция пустая), возвращает nil.

**Функция ldap\_next\_entry**

Функция ldap\_next\_entry возвращает элемент коллекции, следующий после заданного.

function ldap\_next\_entry(

ld : PLDAP;

entry : PLDAPMessage

):PLDAPMessage;

**ПАРАМЕТРЫ:**

**ld**

Указатель на сессию.

**entry**

Результат вызова функции ldap\_first\_entry.

**ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ:**

Если поиск удачен, то функция возвращает указатель на первый элемент коллекции найденных объектов. В противном случае (если коллекция пустая), возвращает nil.

**Функция ldap\_first\_attribute**

Функция ldap\_first\_attribute возвращает первый атрибут данного объекта каталога.

function ldap\_first\_attribute(

ld : PLDAP;

entry : PLDAPMessage;

var ptr : PPBerElement

):PCHAR;

**ПАРАМЕТРЫ:**

**ld**

Указатель на сессию

**entry**

Объект, атрибуты которого должны быть получены. Результат вызова функций ldap\_first\_entry или ldap\_next\_entry.

**ptr**

Используется для получения следующего атрибута на странице.

**ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ:**

Указатель на нуль-терминированную строку. Если функция успешна, то возвращается указатель на буфер, содержащий имя атрибута. Функция возвращает nil, если больше нет атрибутов.

Если функция неуспешна, то возвращается nil и устанавливается параметр ошибки сеанса в структуре LDAP.

**Функция ldap\_next\_attribute**

Функция ldap\_next\_attribute возвращает следующий атрибут данного объекта каталога.

function ldap\_next\_attribute(

ld : PLDAP;

entry : PLDAPMessage;

var ptr : PberElement

**ПАРАМЕТРЫ:**

**ld**

Указатель на сессию

**entry**

Объект, атрибуты которого должны быть получены. Результат вызова функций ldap\_first\_entry или ldap\_next\_entry.

**ptr**

Используется для получения следующего атрибута на странице.

**ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ:**

Если функция успешна, то возвращается указатель на буфер, содержащий имя атрибута. Функция возвращает nil, если больше нет атрибутов.

Если функция неуспешна, то возвращается nil и устанавливается параметр ошибки сеанса в структуре LDAP.

**Функция ldap\_get\_values**

Функция ldap\_get\_values возвращает список значений указанного атрибута объекта.

function ldap\_get\_values(

ld : PLDAP;

entry : PLDAPMessage;

attr : PCHAR

):pointer;

**ПАРАМЕТРЫ:**

**ld**

Указатель на сессию.

**entry**

Элемент коллекции, полученный в результате вызова функции ldap\_next\_entry.

**attr**

Указатель на нуль-терминированную строку, содержащую атрибут, значения которого должны быть найдены.

**ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ:**

Если функция успешна, то она возвращает список указателей на значения. Если никакие значения атрибута не найдены, то возвращает nil.

**Функция ldap\_add\_s**

Синхронная функция ldap\_add\_s осуществляет операцию добавления элемента к дереву. Для успешного выполнения операции родитель добавляемого элемента должен уже существовать или быть пустым (равным корню distinguished name).

function ldap\_add\_s(

ld : PLDAP;

dn : PCHAR;

attrs : array of PLDAPMod

):ULONG;

**ПАРАМЕТРЫ:**

**ld**

Указатель на сессию.

**dn**

Distinguished name добавляемого элемента.

**attrs**

Массив указателей на LDAPMod структуры. Каждая структура определяет единственный атрибут.

**ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ:**

Если функция успешна, то она возвращает значение LDAP\_SUCCESS.

Если функция неуспешна, то она возвращает код ошибки.

**Функция ldap\_modify\_s**

Функция ldap\_modify\_s модифицирует существующую запись. Формат аналогичен предыдущей функции.

function ldap\_modify\_s(

ld : PLDAP;

dn : PCHAR;

mods : pointer

):ULONG;

**ПАРАМЕТРЫ:**

**ld -**Указатель на сессию.

**dn -**Имя модифицируемого элемента.

**mods**

Нуль-терминированный массив модификаций (массив атрибутов и их значений) элемента.

**ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ:**

Если функция успешна, то она возвращает значение LDAP\_SUCCESS.

Если функция неуспешна, то она возвращает код ошибки.

**Функция ldap\_delete\_s**

Синхронная функция ldap\_delete\_s удаляет объект из каталога.

function ldap\_delete\_s(ld: PLDAP; dn: PCHAR):ULONG;

**ПАРАМЕТРЫ:**

**ld -** Указатель на сессию.

**dn -** Distinguished name удаляемого объекта.

**ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ:**

Если функция успешна, то она возвращает значение LDAP\_SUCCESS.

Если функция неуспешна, то она возвращает код ошибки.

### Описание разработанных модулей

**LDAPClientUnit.pas**

Модуль предоставляет удобный программный интерфейс для работы с wldap32.dll. Использование указателей скрыто внутри этого модуля, освобождая остальные части программы от работы с этим типом данных.

Структура LDAPClientStatus используется для возвращения статуса выполнения текущей операции.

Типы LDAPEntryArray, LDAPEntry и класс LDAPAttribute используется для возвращения результатов поиска.

Класс LDAPClient – основной класс модуля. Через него другие части программы взаимодействует с LDAP.

Основные функции LDAPClient:

* **Init**. Инициализурует LDAP сессию.

function Init(hostName : string; portNumber : integer) : LDAPClientStatus;

ПАРАМЕТРЫ:

hostname – имя хоста.

portNumber – номер порта (по умолчанию, 389).

ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ:

LDAPClientStatus – статус выполнения операции.

* **ConnectSimple**. Соединяет клиента с сервером.

function ConnectSimple(userDN : string; password : string) : LDAPClientStatus;

ПАРАМЕТРЫ:

userDN – имя пользователя.

Password – пароль.

ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ:

LDAPClientStatus – статус выполнения операции.

* **Disconnect**. Разрывает соединение.

procedure Disconnect();

* **Search**. Функция поиска.

function Search(catalogDN, filter : string;

attr : array of LDAPAttribute;

var searchResult : LDAPEntryArray) : LDAPClientStatus;

ПАРАМЕТРЫ:

catalogDN – характерное имя каталога, в котором будет идти поиск.

Filter – фильтр поиска.

Attr – массив названий атрибутов, с которыми должны вернуться записи.

ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ:

searchResult – массив результатов поиска.

LDAPClientStatus – статус выполнения операции.

* **AddEntry**. Функция добавления элемента.

function AddEntry(entryDN : string;

attr : array of LDAPAttribute) : LDAPClientStatus;

ПАРАМЕТРЫ:

entryDN – имя записи.

Attr – массив атрибутов и их значений.

ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ:

LDAPClientStatus – статус выполнения операции.

* **ModifyEntry**. Функция модифицирования заданного объекта.

function ModifyEntry(entryDN : string;

attrToModify : array of LDAPAttribute;

modifyType : integer) : LDAPCLientStatus;

ПАРАМЕТРЫ:

entryDN – имя записи.

attrToModify – массив атрибутов и их значений, которые нужно изменить.

modifyType – целое число (0 – изменить значения атрибутов, 1 – удалить заданные атрибуты)

ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ:

LDAPClientStatus – статус выполнения операции.

* **DeleteEntry**. Функция удаления.

function DeleteEntry(entryDN : string) : LDAPClientStatus;

ПАРАМЕТРЫ:

entryDN – имя записи.

ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ:

LDAPClientStatus – статус выполнения операции.

* **CompareAttributeValue**. Проверяет на совпадение значения заданного атрибута объекта.

function CompareAttributeValue(entryDN,

attrName,

attrValueToCompare : string) : boolean;

ПАРАМЕТРЫ:

entryDN – имя записи.

attrName – наименование атрибута.

attrValueToCompare – значение, с которым надо сравнить текущее значение атрибута.

ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ:

Если совпадают, то true.

Если не совпадают, то false.

**ADSchemaUnit.pas**

Модуль отвечает за взаимодействия других частей программы со схемой. Класс ADSchema находится на самом верхнем уровне абстракции модуля управления схемой. Внутри этого класса скрыто взаимодействие c LDAPClient.

Основные функции ADSchema:

* **Create**. В зависимости от входных параметров или сразу соединяется с сервером, или просто создает объект, предоставляя программисту самому решать, когда соединиться.

constructor Create();overload;

constructor Create(hostName, userName, password: string; portNumber : integer;

var status : ADSchemaStatus); overload;

ПАРАМЕТРЫ:

hostname – имя хоста.

username – имя пользователя.

Password – пароль пользователя.

portNumber – номер порта (по умолчанию, 389).

ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ:

Status – статус выполнения операции.

* **Destroy**. Уничтожает объект и освобождает всю память, которую использовал объект.

destructor Destroy(); override;

* **Connect**. Функция соединения. Используется, если при создании объекта был использован конструктор без параметров или если была вызвана функция Disconnect до этого.

procedure Connect(hostName, userName, password: string; portNumber : integer;

var status : ADSchemaStatus);

ПАРАМЕТРЫ:

hostname – имя хоста.

username – имя пользователя.

Password – пароль пользователя.

portNumber – номер порта (по умолчанию, 389).

ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ:

Status – статус выполнения операции.

* **Disconnect**. Функция разрыва соединения. Вызывается из деструктора, если соединение с сервером все еще активна.

procedure Disconnect();

* **GetEntry**. Ищет и возвращает данные, которые были заданы в параметрах.

function GetEntry(CNname : string;

entrType : EntryType;

var status : ADSchemaStatus) : ADEntry; overload;

function GetEntry(CNname : string;

entrType : EntryType;

withAttr : array of string;

var status : ADSchemaStatus) : ADEntry; overload;

ПАРАМЕТРЫ:

CNname – имя класса или атрибута.

entrType – тип записи.

withAttr – массив наименований запрашиваемых атрибутов у записи.

ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ:

Status – статус выполнения операции.

Объект класса ADEntry, содержащее данные запрашиваемой записи.

* **GetAll**. Возвращает все объекты выбранного типа: атрибуты или классы, с заданными атрибутами.

function GetAll(entrType : EntryType;

var status : ADSchemaStatus) : ADEntryList; overload;

function GetAll(entrType : EntryType;

withAttr : array of string;

var status : ADSchemaStatus) : ADEntryList; overload;

ПАРАМЕТРЫ:

entrType – тип записи.

withAttr – массив наименований запрашиваемых атрибутов у записей.

ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ:

Status – статус выполнения операции.

Объект класса ADEntryList, содержащее данные всех возвращаемые записей.

* **GetEntries**. Также возвращает данные, но позволяет задавать фильтр поиска.

function GetEntries(filter : string;

withAttr : array of string;

var status : ADSchemaStatus) : ADEntryList;

ПАРАМЕТРЫ:

Filter – LDAP-фильтр поиска.

withAttr – массив наименований запрашиваемых атрибутов у записей

ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ:

Status – статус выполнения операции.

Объект класса ADEntryList, содержащее данные всех запрашиваемых записей.

* **AddEntry**. Добавляет новый класс или атрибут в схему.

function AddEntry(newEntry : ADEntry) : ADSchemaStatus;

ПАРАМЕТРЫ:

newEntry – объект класса ADEntry с данными о добавляемой записи.

ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ:

Статус выполнения операции.

* **ModifyEntryAttributes**. Изменяет атрибуты заданного объекта.

function ModifyEntryAttributes(modifiedEntry : ADEntry) : ADSchemaStatus;

ПАРАМЕТРЫ:

modifiedEntry – объект класса ADEntry с данными записи, которые необходимо изменить.

ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ:

Статус выполнения операции.

* **DeleteEntryAttributes**. Удаляет заданные атрибуты объекта.

function DeleteEntryAttributes(entryWithDeleteAttributes : ADEntry)

: ADSchemaStatus; overload;

ПАРАМЕТРЫ:

entryWithDeleteAttributes – объект класса ADEntry с данными об атрибутах, которые надо удалить.

ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ:

Статус выполнения операции.

* **ActivateEntry**. Активирует объект схемы, если он был неактивен.

function ActivateEntry(defunctEntryName : string) : ADSchemaStatus;

ПАРАМЕТРЫ:

defunctEntryName – имя неактивного класса или атрибута.

ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ:

Статус выполнения операции.

* **DeactivateEntry**. Уничтожает (деактивирует) объект схемы.

function DeactivateEntry(activeEntryName : string) : ADSchemaStatus;

ПАРАМЕТРЫ:

activeEntryName – имя активного класса или атрибута.

ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ:

Статус выполнения операции.

Структуры возвращаемых данных описаны в модуле ADSchemaTypes.

**ADSchemaTypes.pas**

Модуль описывает классы, которые структурируют возвращаемые и принимаемые данные класса ADSchema.

* Класс ADSchemaStatus описывает структуру статуса выполнения операции. В полях этого класса хранятся номер, тип и сообщение ошибки.
* Класс ADAttribute описывает структуру данных атрибута объекта и предоставляет функции для контролируемого изменения содержимого объектов этого класса.
* Класс ADEntry описывает структуру объекта схемы. Внутри себя хранит список объектов ADAttribute.
* Класс ADEntryList описывает список объектов типа ADEntry. Предоставляет функции контролируемого изменения своих полей.

Описанные выше классы упрощают взаимодействие с модулем управления схемой, так как точно определяют формат возвращаемых данных и предоставляют удобный программный интерфейс для манипулирования ими.

**MainPage.pas**

Модуль главной страницы программного комплекса. На этой странице реализован вывод содержимое схемы, вывод дерева классов, частичное редактирование атрибутов выбранного объекта, активация и деактивация объекта класса или атрибута схема.

Внутри модуля содержится основной функционал программного комплекса.

**ConnectPage.pas**

Модуль страницы соединения. При успехе создает объект класса ADSchema и отправляет на главную страницу.

**AddPage.pas**

Модуль реализует добавление новых и изменение старых элементов схемы, предоставляя удобный графический интерфейс.

**SelectPage.pas**

Вспомогательная форма для модуля AddPage. Выводит список всех доступных атрибутов, которые затем добавляются к обязательным или необязательным типам.

**ErrorPage.pas**

Модуль вывода ошибки. Если программа перехватила ошибку запускается эта страница, где указывается номер и описание ошибки.

# Результаты тестирования программного комплекса

Тестирование программного комплекса проводилось на 3 серверных операционных системах, являющихся владельцами службы каталогов Active Directory.

## Тестирование на Windows Server 2003 R2

Тестирование взаимодействия с Windows Server 2003 R2.

1. Запустить программу на клиенте и ввести доменное имя рабочей станции с операционной системой Windows Server 2003 R2.

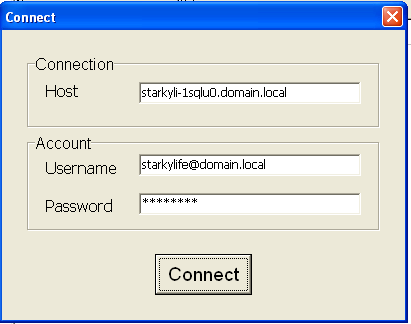


Рис. 4.1.. Соединение с сервером

1. Подключение и вывод содержимое схемы прошло успешно (рис.4.1.2.).

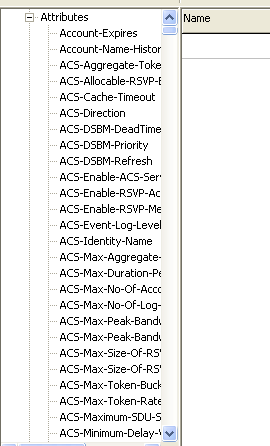


Рис. 4.1.. Главное окно с загруженными данными

1. Тест добавления в схему нового класса.

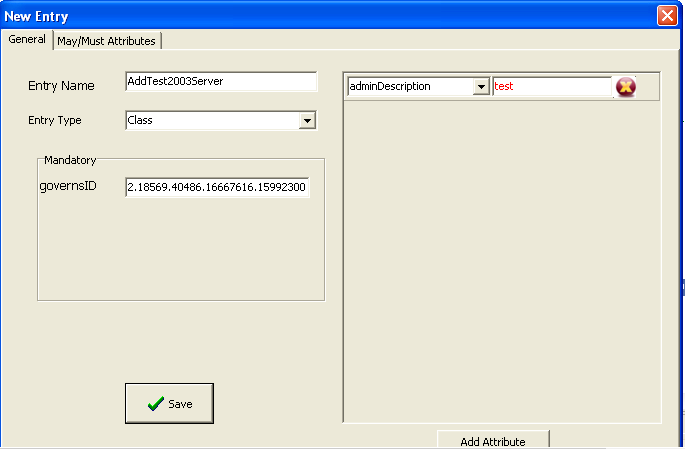


Рис. 4.1.. Окно добавления

1. Проверка на сервере.

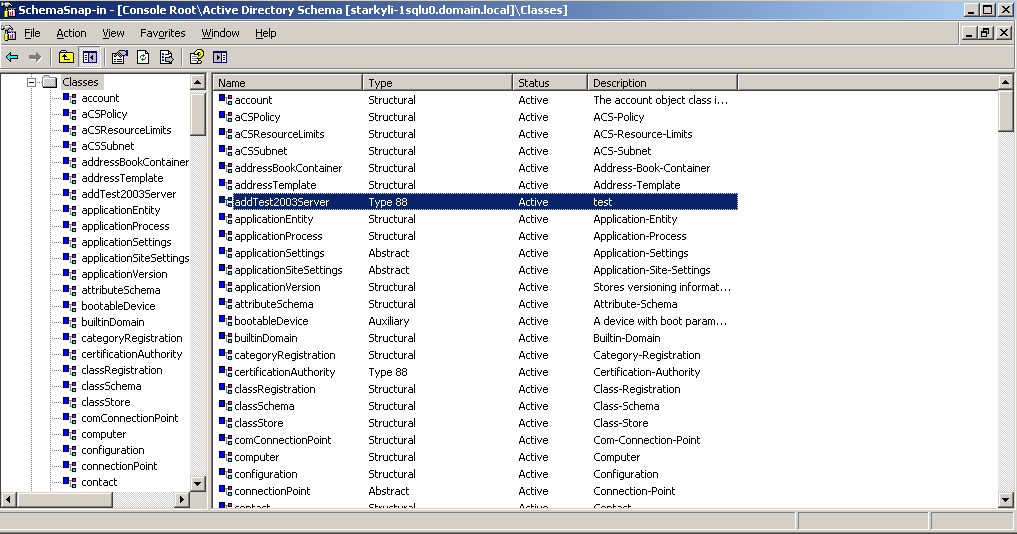


Рис. 4.1.4. Оснастка «Схема Active Directory» на Windows Server 2003 R2

Тестирование прошло успешно.

## Тестирование на Windows Server 2008 R2

Тестирование взаимодействия с Windows Server 2008 R2.

1. Запустить программу на клиенте и ввести доменное имя рабочей станции с операционной системой Windows Server 2008 R2.

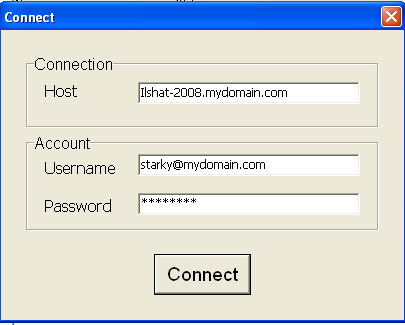


Рис. 4.2.. Окно соединения

1. Подключение и вывод содержимое схемы прошло успешно.

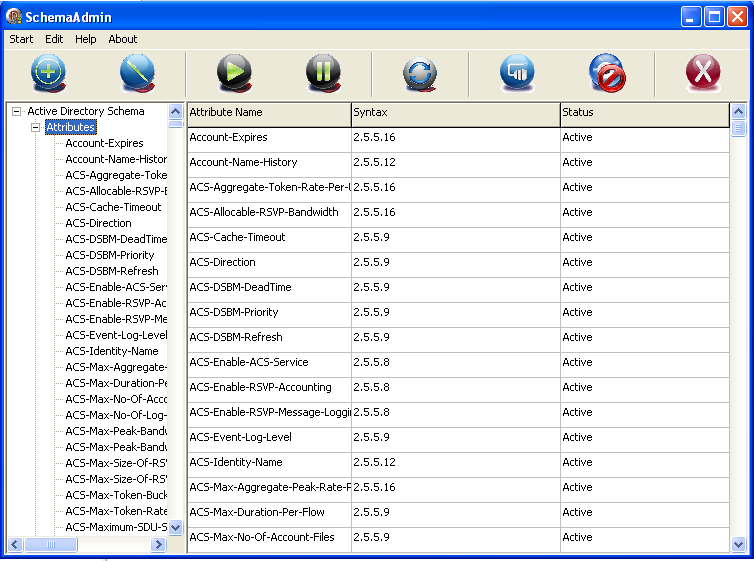


Рис. 4.2.. Главное окно с загруженными данными

1. В качестве теста будет модифицироваться атрибут adminDescription у класса myTest (рис. 4.2.3.).

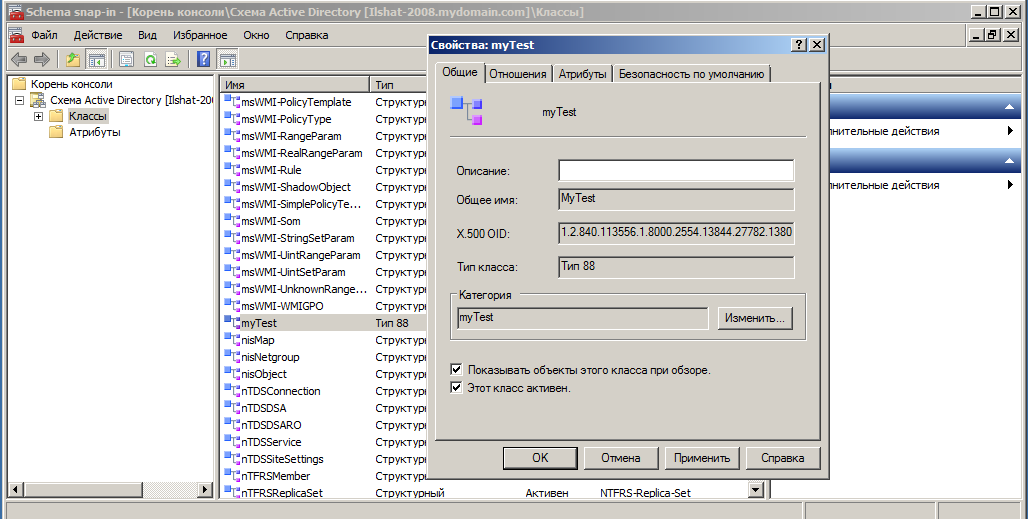


Рис. 4.2.3. Оснастка «Схема Active Directory» на Windows Server 2008 R2

1. В программе выбирается объект myTest и нажимается кнопка Modify, после чего вводятся необходимые данные (рис. 4.2.4.).

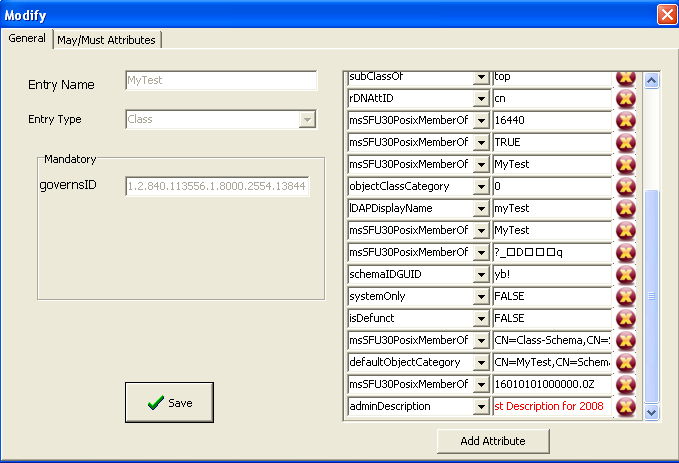


Рис. 4.2.. Окно изменения класса

1. Нажимается Save и проверяется, обновился ли класс через оснастку Active Directory Schema.

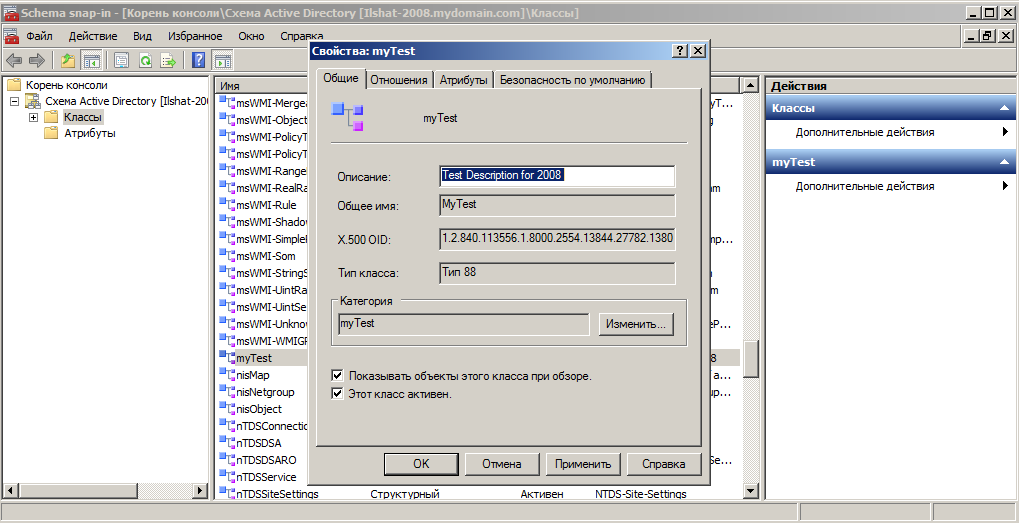


Рис. 4.2.. Свойства класса MyTest – оснастка «Схема Active Directory»

Тестирование прошло успешно.

## Тестирование на Windows Server 2012 R2

* 1. . Запускается программу на клиенте и вводится доменное имя рабочей станции с операционной системой Windows Server 2012 R2 (рис.4.3.1.).

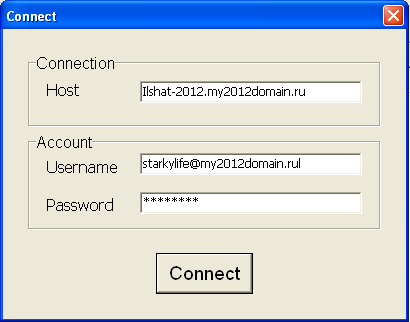


Рис. 4.3.. Окно соединения

1. Проверка установление соединения и вывода содержимое схемы (рис.4.3.2.).

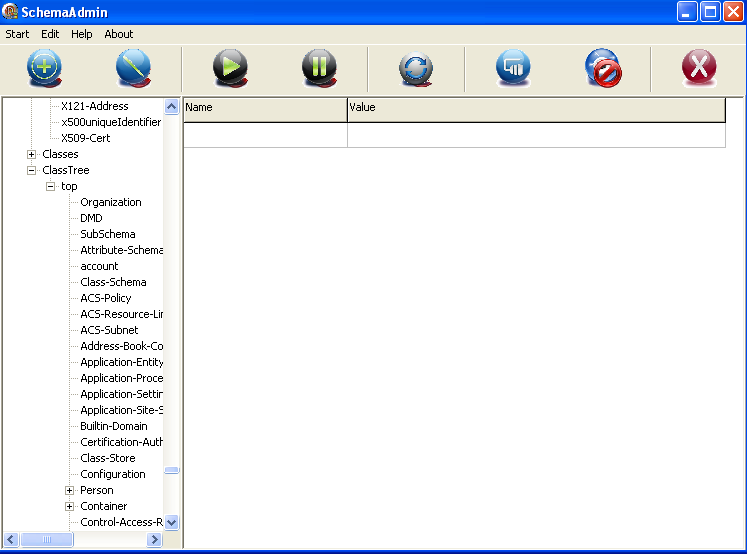


Рис. 4.3.. Главное окно с загруженными данными

Успешно.

1. Будет тестироваться активация и деактивация объекта схемы.

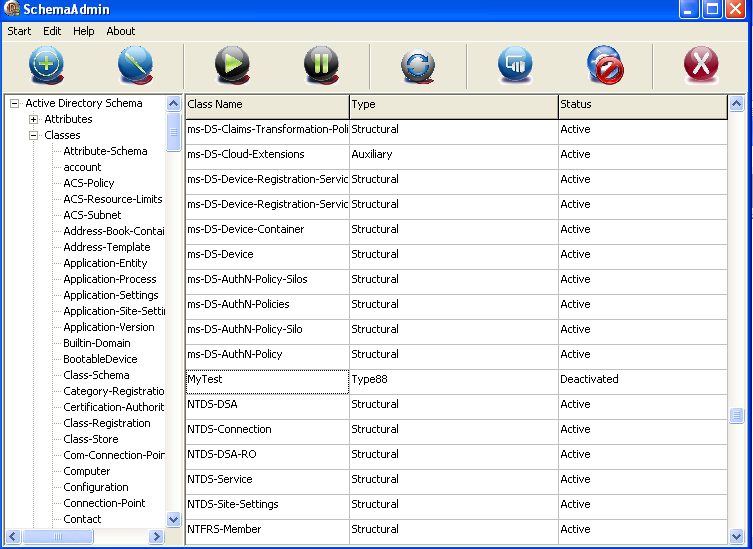


Рис. 4.3.. Главное окно с деактивированным классом MyTest

Нажимается кнопка Activate .

1. Результат работы программы.

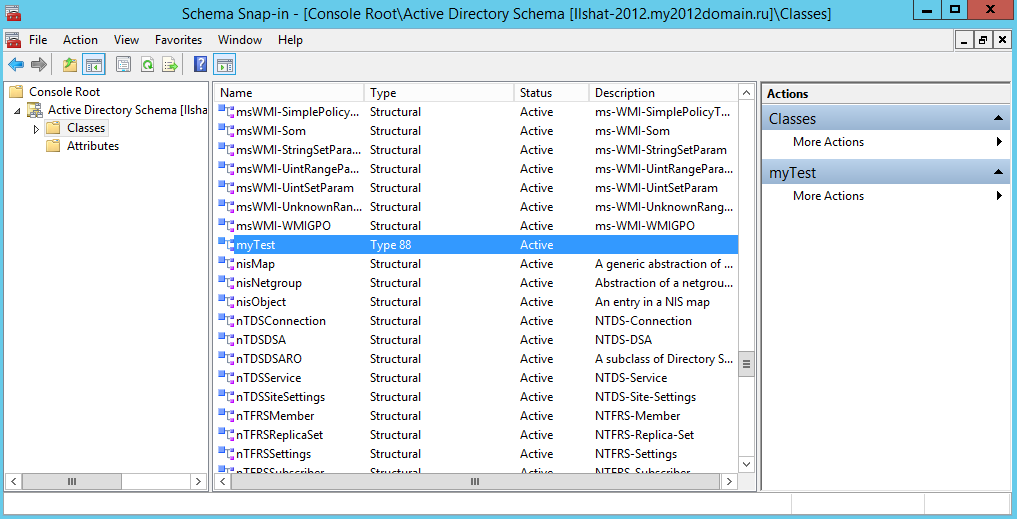


Рис. 4.3.4. Оснастка «Схема Active Directory» на Windows Server 2012 R2

Тестирование прошло успешно.

# Заключение

Служба каталогов – это сервис, без которого не может обойтись и нормально функционировать практически ни одна организация. Она является центральным звеном IT-инфраструктуры компании.

Технологии развиваются очень быстро в наше время, однако новые версии службы каталогов выпускаются очень редко, так как улучшение существующего занимает много времени. В связи с этим в компаниях часто ставятся вопросы расширения схемы Active Directory, чтобы удовлетворить текущие нужды предприятия.

Целью дипломной работы было разработать программный комплекс, который обеспечил бы организациям необходимый набор функционала для управления схемой службы каталогов Active Directory.

Для достижения этой цели были решены следующие задачи:

1. Выявлены необходимые условия расширения схемы.
2. Разработан программный комплекс со следующими возможностями:
   1. Подключение к службе каталогов.
   2. Вывод содержимое схемы. Сюда относится вывод списка всех атрибутов и классов в соответствующих разделах графического интерфейса.
   3. Вывод дерева классов. Дерево строится на основании атрибута subClassOf объектов классов.
   4. Добавление новых классов и атрибутов. В программе необходимо нажать кнопку «Add» в пункте меню или на верхней панели. Программа сама проверяет правильность вводимых данных и, в случае ошибки, выводит сообщение.
   5. Изменение существующих классов и атрибутов. Для этого необходимо выбрать объект и нажать соответствующую кнопку. Однако необходимо учитывать, что изменение некоторых атрибутов недопустимо, так как они являются системными, т.е. только Active Directory может изменять их значения.
3. Программа разработана в виде набора модулей. Модульная структура упрощает дальнейшую поддержку программного обеспечения.
4. В программе реализована совместимость с разными версиями серверных операционных систем. Было протестировано взаимодействие с серверами на основе Windows Server 2003 R2, Windows Server 2008 R2 и Windows Server 2012 R2.
5. Разработаны руководство пользователя, где описывается, как работать с программой, и руководство программиста, с описанием всех модулей и их взаимодействия.

В дальнейшем можно усовершенствовать программный комплекс, реализовав консольный режим и документирование изменений схемы.

Программа может быть использована администраторами службы каталогов Microsoft для внесения изменений в схему. Разработчики программного обеспечения, предусматривающее взаимодействие с Active Directory, могут внедрить в свой продукт сам модуль управления схемой.

# Conclusion

Absolutely no organization can normally function without a directory service. It is the central element of the company’s IT-infrastructure.

Nowadays technologies develop rapidly, but new versions of directory services come out rarely, because their development takes too much time. Concerning this, the companies very often decide to extend the Active Directory Schema to meet the current needs.

The goal of this work was to develop the software that gives companies functionality to manage the directory service’s schema.

To achieve this goal following tasks were solved:

1. Find out the necessary conditions for extending the schema.
2. Developed the software with following capabilities:
   1. Connecting to directory service.
   2. Displaying the schema content. It includes the withdrawal of the list of attributes and classes in the corresponding sections of the User Interface.
   3. Displaying the classes’ tree. The tree is constructed based on the *subClassOf* attribute.
   4. Adding new classes and attributes. For this must be pressed *Add* button on top panel or menu. Program checks the validity of entered data and displays an error message if something is wrong.
   5. Modifying existing classes and attributes. To do this, select the object and press corresponding button. Some attributes cannot be changed because they are system only. It means that only Active Directory itself can modify them.
3. The program is designed as set of modules. The modular structure makes easy the further support of the software.
4. The program features compatibility with different versions of server operating systems.
5. Have been written a user manual, which describes how to work with the program, and programmer’s guide with a description of all modules and their interaction.

In the future, software can be improved by implementing console mode and logging the schema changes.

Administrators of Active Directory for changing the schema can use this program. Moreover, software developers can implement the schema management module to their products.

# Литература

* + 1. Программирование в Delphi для Windows: учебное пособие / Архангельский А.Я. М.: ООО «Бином-Пресс», 2007 – 1248 стр.
    2. Совершенный код. Мастер-класс / Макконнелл С. М.: Издательство «Русская редакция»,2015 – 896 стр.
    3. Справочник системного администратора по программированию Windows / Коробко И.В. СПБ.: БХВ-Петербург, 2009 – 576 стр.
    4. Компоненты Delphi (Электронный ресурс): сайт «Delphi для начинающих», 2016 – Режим доступа: http://helpdelphi.ru/.
    5. Lightweight Directory Access Protocol (Электронный ресурс): сайт разработчиков Windows, 2016 – Режим доступа: <https://msdn.microsoft.com/en-us/library/windows/desktop/aa367008(v=vs.85).aspx>.
    6. LDAP Error Codes (Электронный ресурс): сайт ServiceNow, 2016 – Режим доступа: <http://wiki.servicenow.com/index.php?title=LDAP_Error_Codes#gsc.tab=0>.
    7. Active Directory Schema (Электронный ресурс): сайт разработчиков Windows, 2016 – Режим доступа: <https://msdn.microsoft.com/en-us/library/windows/desktop/ms675085(v=vs.85).aspx>.
    8. Понятия и обзор LDAP (Электронный ресурс): сайт Pro-LDAP, 2016 – Режим доступа: <http://pro-ldap.ru/tr/zytrax/ch2/>.

# Приложение 1. Листинг программы

**Модуль LDAPClientUnit.pas**

{ This unit contains LDAP wrapper class.

Author: Ismagilov Ilshat (student group 4410)

Email: ilshat.ismagilov2014@gmail.com

Organisation: KAI (kai.ru)

Version: 2.0

}

unit LDAPClientUnit;

interface

uses Windows, Classes, SysUtils, jwawinldap, HelpUnit\_LDAPClient;

const

MODIFY\_TYPE\_REPLACE = 0;

MODIFY\_TYPE\_DELETE = 1;

type

{ Type for LDAPClient status messages }

LDAPClientStatus = record

numb : integer;

msg : string;

end;

{ Entry type }

LDAPEntry = record

dn : string;

attributes : TList;

end;

{ Type for LDAPClient search data:

\* array of entries

\* attributes is a TList of LDAPAttribute }

LDAPEntryArray = array of LDAPEntry;

{ Class for attribute data }

LDAPAttribute = class

strict private

AttributeName : string;

AttributeValue : array of string;

AttributeValueCount : integer;

{ Getter function }

function GetAttributeValue(index : integer) : string;

{ Setter procedure }

procedure SetAttributeValue(index : integer; value : string);

public

property Name : string

read AttributeName

write AttributeName;

property Value[index : integer] : string

read GetAttributeValue

write SetAttributeValue;

property ValueCount : integer

read AttributeValueCount;

{ Constructor that initilize the LDAPAttribute object

\* sets the length of AttributeValue array }

constructor Create(valueCount : integer);

end;

{ Wrapper class to work with LDAP }

LDAPClient = class

strict private

pLDAPSession : PLDAP;

//TODO: Check functions

public

{ Initiates LDAP }

function Init(hostName : string;

portNumber : integer) : LDAPClientStatus;

{ Simple Bind Connect }

function ConnectSimple(userDN : string;

password : string) : LDAPClientStatus;

{ Disconnect from server }

procedure Disconnect();

{ Search : result as out parameter

\*send empty array to get all attributes }

function Search(catalogDN, filter : string;

attr : array of LDAPAttribute;

var searchResult : LDAPEntryArray) : LDAPClientStatus;

{ Add }

function AddEntry(entryDN : string;

attr : array of LDAPAttribute) : LDAPClientStatus;

{ Modify:

\* modifyType = 0 (change attribute/attributes)

\* modifyType = 1 (delete attribute/attributes) }

function ModifyEntry(entryDN : string;

attrToModify : array of LDAPAttribute;

modifyType : integer) : LDAPCLientStatus;

{ Delete }

function DeleteEntry(entryDN : string) : LDAPClientStatus;

{ Compare Values }

function CompareAttributeValue(entryDN,

attrName,

attrValueToCompare : string) : boolean;

end;

implementation

{ --------------------------------------------------------

------------------- IMPLEMENTATION ---------------------

------------------- LDAPAttribute ----------------------

-------------------------------------------------------- }

{ Constructor

\* sets the length of AttributeValue array }

constructor LDAPAttribute.Create(valueCount : integer);

begin

SetLength(AttributeValue, valueCount);

AttributeValueCount := valueCount;

end;

{ Getter for one element of AttributeValue }

function LDAPAttribute.GetAttributeValue(index : integer) : string;

begin

result := AttributeValue[index];

end;

{ Setter for one element of AttributeValue }

procedure LDAPAttribute.SetAttributeValue(index : integer; value : string);

begin

AttributeValue[index] := value;

end;

{ --------------------------------------------------------

------------------- IMPLEMENTATION ---------------------

--------------------- LDAPClient -----------------------

-------------------------------------------------------- }

{ ----------------- Init (public) ---------------------- }

function LDAPClient.Init(hostName : string;

portNumber : integer) : LDAPClientStatus;

var

returnCode, version : ulong;

resTemp : LDAPClientStatus;

begin

resTemp.numb := 0;

resTemp.msg := 'Success';

pLDAPSession := ldap\_init(PChar(hostName), portNumber);

if pLDAPSession = nil then

begin

resTemp.msg := 'LDAP Init fail!';

resTemp.numb := GetLastError();

result := resTemp;

Exit;

end;

// Setting option LDAP

// Version

version := LDAP\_VERSION3;

returnCode := ldap\_set\_option(pLDAPSession, LDAP\_OPT\_PROTOCOL\_VERSION, @version);

if returnCode <> LDAP\_SUCCESS then

begin

resTemp.msg := 'LDAP Set option(version) fail!';

resTemp.numb := returnCode;

result := resTemp;

Exit;

end;

// Auto reconnect

returnCode := ldap\_set\_option(pLDAPSession, LDAP\_OPT\_AUTO\_RECONNECT, LDAP\_OPT\_ON);

if returnCode <> LDAP\_SUCCESS then

begin

resTemp.msg := 'LDAP Set option(auto-reconnect) fail!';

resTemp.numb := returnCode;

result := resTemp;

Exit;

end;

result := resTemp;

end;

{ ---------------- ConnectSimple (public) ---------------- }

function LDAPClient.ConnectSimple(userDN : string;

password : string) : LDAPClientStatus;

var

returnCode : ulong;

resTemp : LDAPClientStatus;

cred, myDn : pchar;

begin

resTemp.numb := 0;

resTemp.msg := 'Success';

// Connecting...

returnCode := ldap\_connect(pLDAPSession, nil);

if returnCode <> LDAP\_SUCCESS then

begin

resTemp.msg := 'LDAP Connect fail!';

resTemp.numb := returnCode;

result := resTemp;

Exit;

end;

// Binding...

cred := PChar(password);

// Note: myDN := 'CN=Ilshat,CN=Users,DC=domain,DC=local';

// or myDN := 'starkylife@domain.local';

myDN := PChar(userDN);

returnCode := ldap\_simple\_bind\_s(pLDAPSession, myDN, cred);

if returnCode <> LDAP\_SUCCESS then

begin

resTemp.msg := 'LDAP Bind fail!';

resTemp.numb := returnCode;

result := resTemp;

Exit;

end;

result := resTemp;

end;

{ ---------------- Disconnect (public) ------------------- }

procedure LDAPClient.Disconnect();

begin

ldap\_unbind\_s(pLDAPSession);

end;

{ ---------------- Search (public) ----------------------- }

function LDAPClient.Search(catalogDN, filter : string;

attr : array of LDAPAttribute;

var searchResult : LDAPEntryArray) : LDAPClientStatus;

var

//Search

returnCode, attrCount, iAttribute : ulong;

resTemp : LDAPClientStatus;

pSearchResults : PLDAPMessage;

pTempAttributes : array of pchar;

pSearchAttributes : pchar;

//Count

entriesCount : ulong;

//Parse

pEntry : PLDAPMessage;

iCnt, iValue, i : ulong;

pBer : PBerElement;

pAttribute : pchar;

ppValue : ppchar;

attrName : string;

attrValue : array of string;

tempAttr : LDAPAttribute;

tempPPValue : ppchar;

begin

resTemp.numb := 0;

resTemp.msg := 'Success';

//------------ Check syntax of search filter ---------------

returnCode := ldap\_check\_filter(pLDAPSession, PChar(filter));

if returnCode <> LDAP\_SUCCESS then

begin

resTemp.numb := returnCode;

resTemp.msg := 'Wrong search filter!';

result := resTemp;

Exit;

end;

//---------------------------------------------

//----! Search Directory Get PLDAPMessage !----

if Length(attr) > 0 then

begin

//Forming array of attributes name

SetLength(pTempAttributes, Length(attr)+1);

attrCount := Length(pTempAttributes);

for iAttribute := 0 to attrCount - 2 do

begin

pTempAttributes[iAttribute] := PChar(attr[iAttribute].Name);

end;

pTempAttributes[attrCount-1] := nil;

pSearchAttributes := @(pTempAttributes[0]);

end

else pSearchAttributes := nil;

returnCode := ldap\_search\_s(pLDAPSession,

PChar(catalogDN),

LDAP\_SCOPE\_SUBTREE,

PChar(filter),

pSearchAttributes,

0,

pSearchResults);

if returnCode <> LDAP\_SUCCESS then

begin

resTemp.msg := 'LDAP Search fail!';

resTemp.numb := returnCode;

if pSearchResults <> nil then

ldap\_msgfree(pSearchResults);

result := resTemp;

Exit;

end;

//DONE: free the LDAPAttribute Objects

if Length(attr) > 0 then

for iAttribute := 0 to Length(attr) - 1 do

begin

attr[iAttribute].Free;

attr[iAttribute] := nil;

end;

//-------------------------------

//------! Count Entries !--------

entriesCount := ldap\_count\_entries(pLDAPSession, pSearchResults);

//-----------------------------------

//------! Parse PLDAPMessage !-------

SetLength(searchResult, entriesCount);

pEntry := nil;

pBer := nil;

pAttribute := nil;

ppValue := nil;

iValue := 0;

try

if entriesCount <> 0 then

begin

for iCnt := 0 to entriesCount - 1 do

begin

searchResult[iCnt].attributes := TList.Create;

//Get the first/next entry

if iCnt = 0 then pEntry := ldap\_first\_entry(pLDAPSession, pSearchResults)

else pEntry := ldap\_next\_entry(pLDAPSession, pEntry);

if pEntry = nil then

begin

resTemp.msg := 'Search Entry #' + IntToStr(iCnt+1) + ' dont exist!';

resTemp.numb := 1;

result := resTemp;

Exit;

end;

//use ldap\_get\_dn for entry name

searchResult[iCnt].dn := ldap\_get\_dn(pLDAPSession, pEntry);

//and ldap\_explode\_dn to make array of it's elements

//Get first attribute

pAttribute := ldap\_first\_attribute(pLDAPSession, pEntry, pBer);

//loop through all attributes

while pAttribute <> nil do

begin

//Get name of Attribute

attrName := pAttribute;

ppValue := ldap\_get\_values(pLDAPSession, pEntry, pAttribute);

if ppValue <> nil then

begin

iValue := ldap\_count\_values(ppValue);

SetLength(attrValue, iValue);

if iValue > 0 then

begin

//woriking with pointers to array

tempPPValue := ppValue;

for i := 0 to iValue - 1 do

begin

attrValue[i] := tempPPValue^;

Inc(tempPPValue);

end;

end;

end;

//Free Memory

if ppValue <> nil then

ldap\_value\_free(ppValue);

ppValue := nil;

tempPPValue := nil;

ldap\_memfree(pAttribute);

//Get Next Attribute

pAttribute := ldap\_next\_attribute(pLDAPSession, pEntry, pBer);

//Fill The Result List

tempAttr := LDAPAttribute.Create(Length(attrValue));

tempAttr.Name := attrName;

for i := 0 to Length(attrValue) - 1 do

tempAttr.Value[i] := attrValue[i];

searchResult[iCnt].attributes.Add(tempAttr);

end;

pBer := nil;

end;

end;

except

on E : Exception do

begin

ldap\_msgfree(pSearchResults);

ldap\_value\_free(ppValue);

resTemp.msg := E.Message;

resTemp.numb := 1;

result := resTemp;

Exit;

end;

end;

ldap\_msgfree(pSearchResults);

ldap\_value\_free(ppValue);

result := resTemp;

end;

{ ---------------- AddEntry (public) -------------------- }

function LDAPClient.AddEntry(entryDN : string;

attr : array of LDAPAttribute) : LDAPClientStatus;

var

returnCode, iAttribute, jVals, newEntryCount, valsCount : ulong;

dn : pchar;

attributesArray : array of HelpAttributeArray;

newEntry : array of PLDAPMod;

pMod : PLDAPMod;

resTemp : LDAPClientStatus;

begin

resTemp.numb := 0;

resTemp.msg := 'Success';

dn := PChar(entryDN);

//Forming new entry: array of attributes(LDAPMods)

SetLength(newEntry, Length(attr)+1);

SetLength(attributesArray, Length(attr));

newEntryCount := Length(newEntry);

for iAttribute := 0 to newEntryCount - 2 do

begin

attributesArray[iAttribute] := HelpAttributeArray.Create;

SetLength(attributesArray[iAttribute].Values, attr[iAttribute].ValueCount+1);

valsCount := Length(attributesArray[iAttribute].Values);

for jVals := 0 to valsCount-2 do

begin

attributesArray[iAttribute].Values[jVals] := PChar(attr[iAttribute].Value[jVals]);

end;

attributesArray[iAttribute].Values[valsCount-1] := nil;

New(pMod);

pMod^.mod\_op := LDAP\_MOD\_ADD;

pMod^.mod\_type := PChar(attr[iAttribute].Name);

pMod^.modv\_strvals := @(attributesArray[iAttribute].Values[0]);

newEntry[iAttribute] := pMod;

end;

newEntry[newEntryCount - 1] := nil;

//Call ldap\_add API function

returnCode := ldap\_add\_s(pLDAPSession, dn, newEntry[0]);

//Free memory

for iAttribute := 0 to Length(newEntry) - 2 do

begin

Dispose(newEntry[iAttribute]);

attributesArray[iAttribute].Free;

attributesArray[iAttribute] := nil;

end;

for iAttribute := 0 to Length(attr) - 1 do

begin

attr[iAttribute].Free;

attr[iAttribute] := nil;

end;

//Check if api call wass successfull

if returnCode <> LDAP\_SUCCESS then

begin

resTemp.msg := 'LDAP Add fail!';

resTemp.numb := returnCode;

result := resTemp;

Exit;

end;

result := resTemp;

end;

{ ---------------- ModifyEntry (public) ------------------ }

function LDAPClient.ModifyEntry(entryDN : string;

attrToModify : array of LDAPAttribute;

modifyType : integer) : LDAPCLientStatus;

var

returnCode, iAttribute, jVals, newEntryCount, valsCount, modType : ulong;

dn : pchar;

attributesArray : array of HelpAttributeArray;

newEntry : array of PLDAPMod;

pMod : PLDAPMod;

resTemp : LDAPClientStatus;

begin

resTemp.numb := 0;

resTemp.msg := 'Success';

dn := PChar(entryDN);

// Set modify type

if modifyType = MODIFY\_TYPE\_REPLACE then

modType := LDAP\_MOD\_REPLACE

else if modifyType = MODIFY\_TYPE\_DELETE then

modType := LDAP\_MOD\_DELETE

else

begin

resTemp.msg := 'LDAP Modify: wrong modify type!';

resTemp.numb := 1;

result := resTemp;

Exit;

end;

//Forming new entry: array of attributes(LDAPMods)

SetLength(newEntry, Length(attrToModify)+1);

SetLength(attributesArray, Length(attrToModify));

newEntryCount := Length(newEntry);

for iAttribute := 0 to newEntryCount - 2 do

begin

New(pMod);

pMod^.mod\_op := modType;

pMod^.mod\_type := PChar(attrToModify[iAttribute].Name);

if modifyType = MODIFY\_TYPE\_REPLACE then

begin

attributesArray[iAttribute] := HelpAttributeArray.Create;

SetLength(attributesArray[iAttribute].Values, attrToModify[iAttribute].ValueCount+1);

valsCount := Length(attributesArray[iAttribute].Values);

for jVals := 0 to valsCount-2 do

begin

attributesArray[iAttribute].Values[jVals] := PChar(attrToModify[iAttribute].Value[jVals]);

end;

attributesArray[iAttribute].Values[valsCount-1] := nil;

pMod^.modv\_strvals := @(attributesArray[iAttribute].Values[0]);

end

else

pMod^.modv\_strvals := nil;

newEntry[iAttribute] := pMod;

end;

newEntry[newEntryCount - 1] := nil;

//Call ldap\_add API function

returnCode := ldap\_modify\_s(pLDAPSession, dn, newEntry[0]);

//Free memory

for iAttribute := 0 to Length(newEntry) - 2 do

begin

Dispose(newEntry[iAttribute]);

attributesArray[iAttribute].Free;

attributesArray[iAttribute] := nil;

end;

for iAttribute := 0 to Length(attrToModify) - 1 do

begin

attrToModify[iAttribute].Free;

attrToModify[iAttribute] := nil;

end;

//Check if api call wass successfull

if returnCode <> LDAP\_SUCCESS then

begin

resTemp.msg := 'LDAP Modify fail!';

resTemp.numb := returnCode;

result := resTemp;

Exit;

end;

result := resTemp;

end;

{ ---------------- DeleteEntry (public) ------------------ }

function LDAPClient.DeleteEntry(entryDN : string) : LDAPClientStatus;

var

returnCode : ulong;

resTemp : LDAPClientStatus;

begin

resTemp.numb := 0;

resTemp.msg := 'Success';

returnCode := ldap\_delete\_s(pLDAPSession, PChar(entryDN));

if returnCode <> LDAP\_SUCCESS then

begin

resTemp.msg := 'LDAP Delete fail!';

resTemp.numb := returnCode;

result := resTemp;

end;

result := resTemp;

end;

{ ---------------- CompareAttributeValue (public) ------- }

function LDAPClient.CompareAttributeValue(entryDN,

attrName,

attrValueToCompare : string) : boolean;

var

returnCode : ulong;

begin

returnCode := ldap\_compare\_s(pLDAPSession,

PChar(entryDN),

PChar(attrName),

PChar(attrValueToCompare));

if returnCode = LDAP\_COMPARE\_TRUE then

begin

result := true;

end

else

begin

result := false;

end;

end;

end.

**Модуль ADSchemaUnit.pas**

{ This unit contains class for Active Directory Schema Manipulation.

Author: Ismagilov Ilshat (student group 4410)

Email: ilshat.ismagilov2014@gmail.com

Organisation: KAI (kai.ru)

Version: 2.0

}

unit ADSchemaUnit;

interface

uses Windows, Classes, SysUtils, ADSchemaTypes, ADSchemaHelpUnit, LDAPClientUnit;

type

EntryType = (

ClassEntry,

AttributeEntry);

{ Public class for administrating the Active Directory(AD) Schema }

ADSchema = class

strict private

//working with LDAPClient

client : LDAPClient;

//Stores the state of connection

isSchemaActive : boolean;

//Client Info

\_HostName, \_UserName, \_Password : string;

\_PortNumber : integer;

//Schema Catalog DN

SchemaDN : string;

currentUserDN : string;

{ Private function for establishing connection }

function pvEstablishConnection() : ADSchemaStatus;

{ Private Function that tries to reconnect to server with current Client Info

\*returns the statues of connection }

function pvTryConnect() : ADSchemaStatus;

{ Private Function that checks connection by sending simple query

\*if fails return false

\*if succeeds return true}

function pvCheckConnection() : boolean;

{ The only function that can change SchemaDN value!

\*Returns true if success }

function pvSetSchemaDN() : boolean;

//working with LDAPClient - Clearing return values of Search

procedure pvClearLDAPSearchResults(var searchResults : LDAPEntryArray);

//search and get

function pvGetEntries(dn : string;

filter : string;

withAttr : array of string;

var status : ADSchemaStatus) : ADEntryList;

//Checks if entry has all needed attributes

function pvCheckEntry(chEntry : ADEntry;

var neededAttributes : TStringList) : boolean;

//Extended Search if too many values is expected

function pvSearchExtended(dn : string;

entrType : EntryType;

withAttr : array of string;

var status : ADSchemaStatus) : ADEntryList;

public

property isActive : boolean

read isSchemaActive;

{ -----------------------------------------------------------------

--------------------- Connection functions ---------------------- }

//Constructor creates an object and connects to server: isActive = true

constructor Create(hostName, userName, password: string; portNumber : integer;

var status : ADSchemaStatus); overload;

//Just creating object: isActive = false. You need to use Connect procedure

constructor Create();overload;

//Destructor

destructor Destroy(); override;

//Connects to server: if success changes isActive to True

procedure Connect(hostName, userName, password: string; portNumber : integer;

var status : ADSchemaStatus);

{ Disconnects from server

\*it also called from destructor if isActive=true }

procedure Disconnect();

{ -----------------------------------------------------------------

---------------------- Data Handling Functions ------------------ }

{ Function returns ADEntry object with all attributes }

function GetEntry(CNname : string;

entrType : EntryType;

var status : ADSchemaStatus) : ADEntry; overload;

{ Function returns ADEntry object with given attributes }

function GetEntry(CNname : string;

entrType : EntryType;

withAttr : array of string;

var status : ADSchemaStatus) : ADEntry; overload;

{ Function returns List of ADEntry with ALL attributes }

function GetAll(entrType : EntryType;

var status : ADSchemaStatus) : ADEntryList; overload;

{ Function returns List of ADEntry with GIVEN attributes }

function GetAll(entrType : EntryType;

withAttr : array of string;

var status : ADSchemaStatus) : ADEntryList; overload;

{ Search and get function with LDAP filter }

function GetEntries(filter : string;

withAttr : array of string;

var status : ADSchemaStatus) : ADEntryList;

{ Function adds new entry to schema

\* given ADEntry object is not freed in function }

function AddEntry(newEntry : ADEntry) : ADSchemaStatus;

{ Function modifies Entry (only attributes that given)

\* given ADEntry object is not freed in function }

function ModifyEntryAttributes(modifiedEntry : ADEntry) : ADSchemaStatus;

{ Function deletes all attributes in given ADEntryObject

\* given ADEntry object is not freed in function }

function DeleteEntryAttributes(entryWithDeleteAttributes : ADEntry)

: ADSchemaStatus; overload;

{ Function deletes given attributes of entry with given name }

function DeleteEntryAttributes(name : string;

attrToDelete : array of string)

: ADSchemaStatus; overload;

{ Function for activating defunct class or attribute }

function ActivateEntry(defunctEntryName : string) : ADSchemaStatus;

{ Function for deactivating active class or attribute }

function DeactivateEntry(activeEntryName : string) : ADSchemaStatus;

end;

implementation

{ --------------------------------------------------------

------------------- IMPLEMENTATION ---------------------

------------------- ADSchema ---------------------------

-------------------------------------------------------- }

{ Public }

constructor ADSchema.Create(hostName, userName, password: string; portNumber : integer;

var status : ADSchemaStatus);

begin

client := LDAPClient.Create;

Connect(hostName, userName, password, portNumber, status);

end;

{ Public }

constructor ADSchema.Create();

begin

client := LDAPClient.Create;

isSchemaActive := false;

end;

{ Public }

destructor ADSchema.Destroy();

begin

if isSchemaActive then

begin

Disconnect;

end;

if client <> nil then

begin

client.Free;

client := nil;

end;

inherited;

end;

{ Public }

procedure ADSchema.Connect(hostName, userName, password: string; portNumber : integer;

var status : ADSchemaStatus);

begin

{if status <> nil then

status.free; }

//check in values

//set in values

\_HostName := hostName;

\_PortNumber := portNumber;

\_UserName := userName;

\_Password := password;

//establishConnection function

status := pvEstablishConnection();

if status.StatusType <> SuccessStatus then

begin

if isSchemaActive then

isSchemaActive := false;

Exit;

end;

isSchemaActive := true;

//Forming SchemaDN

if not pvSetSchemaDN then

begin

status.Free;

status := ADSchemaStatus.Create(1, ADSchemaError, 'SetSchemaDN error!');

isSchemaActive := false;

Disconnect;

Exit;

end;

end;

{ Public }

procedure ADSchema.Disconnect();

begin

//LDAPClient disconnect

client.Disconnect;

\_HostName := '';

\_UserName := '';

\_Password := '';

isSchemaActive := false;

end;

{ ----------------------------------------------

------------- Private functions -------------- }

{ Private }

function ADSchema.pvEstablishConnection() : ADSchemaStatus;

var

clientStatus : LDAPClientStatus;

begin

//LDAPCLient connect

clientStatus := client.Init(\_HostName, \_PortNumber);

if clientStatus.numb <> 0 then

begin

result := ADSchemaStatus.Create(clientStatus.numb, LDAPError, clientStatus.msg);

Exit;

end;

clientStatus := client.ConnectSimple(\_UserName, \_Password);

if clientStatus.numb <> 0 then

begin

result := ADSchemaStatus.Create(clientStatus.numb, LDAPError, clientStatus.msg);

Exit;

end;

result := ADSchemaStatus.Create();

end;

{ Private }

function ADSchema.pvTryConnect() : ADSchemaStatus;

begin

//try to establishConnection

result := pvEstablishConnection;

//if connection is established

if result.StatusType = SuccessStatus then

begin

//set isSchemaActive = true

isSchemaActive := true;

Exit;

end;

isSchemaActive := false;

end;

{ Private }

function ADSchema.pvCheckConnection() : boolean;

var

clientStatus : LDAPClientStatus;

searchResult : LDAPEntryArray;

begin

//Check is Schema Active

if isSchemaActive = false then

begin

result := isSchemaActive;

Exit;

end;

//Simple query to check connection

clientStatus := client.Search('CN=top,' + SchemaDN, '(objectClass=classSchema)', [], searchResult);

pvClearLDAPSearchResults(searchResult);

//if false then call Disconnect to clear memory

if clientStatus.numb <> 0 then

begin

Disconnect;

end;

//return isSchemaActive

result := isSchemaActive;

end;

{ Private }

function ADSchema.pvSetSchemaDN() : boolean;

var

dn : string;

parsedDn : string;

begin

dn := GetUserNameExString(NameFullyQualifiedDN);

currentUserDN := dn;

//parse the dn

if pos('DC=', dn) = 0 then

begin

result := false;

Exit;

end;

parsedDn := copy(dn, pos('DC=', dn));

Insert('CN=Schema,CN=Configuration,', parsedDn, 0);

SchemaDN := parsedDn;

result := true;

end;

{ -----------------------------------------------------------------

---------------------- Data Handling Functions ------------------ }

{ Public }

{ Function returns ADEntry object with all attributes }

function ADSchema.GetEntry(CNname : string;

entrType : EntryType;

var status : ADSchemaStatus) : ADEntry;

begin

result := GetEntry(CNname, entrType, [], status);

end;

{ Public }

{ Function returns ADEntry object with given attributes }

function ADSchema.GetEntry(CNname : string;

entrType : EntryType;

withAttr : array of string;

var status : ADSchemaStatus) : ADEntry;

var

entryDN, filter : string;

resTemp : ADEntryList;

begin

//forming entry dn

entryDN := 'CN=' + CNname + ',' + SchemaDN;

//forming filter

case entrType of

ClassEntry: filter := '(objectClass=classSchema)';

AttributeEntry: filter := '(objectClass=attributeSchema)';

end;

resTemp := pvGetEntries(entryDN, filter, withAttr, status);

//Check item count

if resTemp = nil then

begin

result := nil;

Exit;

end;

if resTemp.EntriesCount <> 1 then

begin

status.free;

status := ADSchemaStatus.Create(2, ADSchemaError, 'Wrong number of entries');

result := nil;

resTemp.Destroy;

Exit;

end;

result := resTemp.Items[0];

end;

{ Public }

{ Function returns List of ADEntry with ALL attributes }

function ADSchema.GetAll(entrType : EntryType;

var status : ADSchemaStatus) : ADEntryList;

begin

result := GetAll(entrType, [], status);

end;

{ Public }

{ Function returns List of ADEntry with GIVEN attributes }

function ADSchema.GetAll(entrType : EntryType;

withAttr : array of string;

var status : ADSchemaStatus) : ADEntryList;

var

entryDN, filter : string;

begin

//forming entry dn

entryDN := SchemaDN;

//forming filter

case entrType of

ClassEntry: filter := '(objectClass=classSchema)';

AttributeEntry: filter := '(objectClass=attributeSchema)';

end;

{result := pvGetEntries(entryDN, filter, withAttr, status);

if status.StatusNumb = 4 then

begin

if result <> nil then

begin

result.Destroy;

result := nil;

end;

status.Free;

status := nil;}

result := pvSearchExtended(entryDN, entrType, withAttr, status);

//end;

end;

{ Public }

{ Search and get function with LDAP filter }

function ADSchema.GetEntries(filter : string;

withAttr : array of string;

var status : ADSchemaStatus) : ADEntryList;

var

entryDN : string;

begin

entryDN := SchemaDN;

result := pvGetEntries(entryDN, filter, withAttr, status);

end;

{ Public }

{ Function adds new entry to schema }

function ADSchema.AddEntry(newEntry : ADEntry) : ADSchemaStatus;

var

neededAttributes : TStringList;

clientStatus : LDAPClientStatus;

entryDN : string;

attributes : array of LDAPAttribute;

attr : LDAPAttribute;

iAttribute, iValue : integer;

begin

//Check connection

//if connection lost exit

if pvCheckConnection = false then

begin

result := pvTryConnect;

if result.StatusType <> SuccessStatus then

begin

Exit;

end;

result.Free;

result := nil;

end;

//check given entry attributes

if pvCheckEntry(newEntry, neededAttributes) = false then

begin

//if there is no all "must" attributes,

//exit with message of needed attributes

Result := ADSchemaStatus.Create(3, ADSchemaError, neededAttributes.CommaText);

neededAttributes.Free;

neededAttributes := nil;

Exit;

end;

//form data for calling the LDAPClient function

entryDN := 'cn=' + newEntry.Name + ',' + SchemaDN;

SetLength(attributes, newEntry.AttributesCount);

for iAttribute := 0 to Length(attributes) - 1 do

begin

attr := LDAPAttribute.Create(newEntry.Attributes[iAttribute].ValuesCount);

attr.Name := newEntry.Attributes[iAttribute].Name;

for iValue := 0 to attr.ValueCount - 1 do

begin

attr.Value[iValue] := newEntry.Attributes[iAttribute].Values[iValue];

end;

attributes[iAttribute] := attr;

end;

//call LDAPClient AddEntry function

clientStatus := client.AddEntry(entryDN, attributes);

//check status

if clientStatus.numb <> 0 then

begin

//if fail, form ADSchemaStatus with LDAPError

Result :=ADSchemaStatus.Create(clientStatus.numb, LDAPError, clientStatus.msg);

Exit;

end;

//if success, form ADSchemaStatus with SuccessStatus

result := ADSchemaStatus.Create;

end;

{ Public }

{ Function modifies Entry (only attributes that given) }

function ADSchema.ModifyEntryAttributes(modifiedEntry : ADEntry) : ADSchemaStatus;

var

clientStatus : LDAPClientStatus;

entryDN : string;

attributes : array of LDAPAttribute;

attr : LDAPAttribute;

iAttribute, iValue : integer;

begin

//Check connection

//if connection lost exit

if pvCheckConnection = false then

begin

result := pvTryConnect;

if result.StatusType <> SuccessStatus then

begin

Exit;

end;

result.Free;

result := nil;

end;

//form data for calling the LDAPClient function

entryDN := 'CN=' + modifiedEntry.Name + ',' + SchemaDN;

SetLength(attributes, modifiedEntry.AttributesCount);

for iAttribute := 0 to Length(attributes) - 1 do

begin

attr := LDAPAttribute.Create(modifiedEntry.Attributes[iAttribute].ValuesCount);

attr.Name := modifiedEntry.Attributes[iAttribute].Name;

for iValue := 0 to attr.ValueCount - 1 do

begin

attr.Value[iValue] := modifiedEntry.Attributes[iAttribute].Values[iValue];

end;

attributes[iAttribute] := attr;

end;

//call LDAPClient ModfiyEntry function

clientStatus := client.ModifyEntry(entryDN, attributes, MODIFY\_TYPE\_REPLACE);

//check status

if clientStatus.numb <> 0 then

begin

//if fail, form ADSchemaStatus with LDAPError

Result := ADSchemaStatus.Create(clientStatus.numb, LDAPError, clientStatus.msg);

Exit;

end;

//if success, form ADSchemaStatus with SuccessStatus

result := ADSchemaStatus.Create;

end;

{ Public }

{ Function deletes all attributes in given ADEntryObject }

function ADSchema.DeleteEntryAttributes(entryWithDeleteAttributes : ADEntry)

: ADSchemaStatus;

var

clientStatus : LDAPClientStatus;

entryDN : string;

attributes : array of LDAPAttribute;

attr : LDAPAttribute;

iAttribute, iValue : integer;

begin

//Check connection

//if connection lost exit

if pvCheckConnection = false then

begin

result := pvTryConnect;

if result.StatusType <> SuccessStatus then

begin

Exit;

end;

result.Free;

result := nil;

end;

//form data for calling the LDAPClient function

entryDN := 'CN=' + entryWithDeleteAttributes.Name + ',' + SchemaDN;

SetLength(attributes, entryWithDeleteAttributes.AttributesCount);

for iAttribute := 0 to Length(attributes) - 1 do

begin

attr := LDAPAttribute.Create(entryWithDeleteAttributes.Attributes[iAttribute].ValuesCount);

attr.Name := entryWithDeleteAttributes.Attributes[iAttribute].Name;

for iValue := 0 to attr.ValueCount - 1 do

begin

attr.Value[iValue] := entryWithDeleteAttributes.Attributes[iAttribute].Values[iValue];

end;

attributes[iAttribute] := attr;

end;

//call LDAPClient Modify function

clientStatus := client.ModifyEntry(entryDN, attributes, MODIFY\_TYPE\_DELETE);

//check status

if clientStatus.numb <> 0 then

begin

//if fail, form ADSchemaStatus with LDAPError

Result := ADSchemaStatus.Create(clientStatus.numb, LDAPError, clientStatus.msg);

Exit;

end;

//if success, form ADSchemaStatus with SuccessStatus

result := ADSchemaStatus.Create;

end;

{ Public }

{ Function deletes given attributes of entry with given name }

function ADSchema.DeleteEntryAttributes(name : string;

attrToDelete : array of string)

: ADSchemaStatus;

var

clientStatus : LDAPClientStatus;

entryDN : string;

attributes : array of LDAPAttribute;

attr : LDAPAttribute;

iAttribute : integer;

begin

//Check connection

//if connection lost exit

if pvCheckConnection = false then

begin

result := pvTryConnect;

if result.StatusType <> SuccessStatus then

begin

Exit;

end;

result.Free;

result := nil;

end;

//form data for calling the LDAPClient function

entryDN := 'CN=' + name + ',' + SchemaDN;

SetLength(attributes, Length(attrToDelete));

for iAttribute := 0 to Length(attributes) - 1 do

begin

attr := LDAPAttribute.Create(0);

attr.Name := attrToDelete[iAttribute];

attributes[iAttribute] := attr;

end;

//call LDAPClient MofifyEntry function

clientStatus := client.ModifyEntry(entryDN, attributes, MODIFY\_TYPE\_DELETE);

//check status

if clientStatus.numb <> 0 then

begin

//if fail, form ADSchemaStatus with LDAPError

Result := ADSchemaStatus.Create(clientStatus.numb, LDAPError, clientStatus.msg);

Exit;

end;

//if success, form ADSchemaStatus with SuccessStatus

result := ADSchemaStatus.Create;

end;

{ Public }

{ Function for activating defunct class or attribute }

function ADSchema.ActivateEntry(defunctEntryName : string) : ADSchemaStatus;

var

defunctEntry : ADEntry;

begin

defunctEntry := ADEntry.Create(defunctEntryName);

defunctEntry.AddAttribute('isDefunct', ['FALSE']);

Result := ModifyEntryAttributes(defunctEntry);

end;

{ Public }

{ Function for deactivating active class or attribute }

function ADSchema.DeactivateEntry(activeEntryName : string) : ADSchemaStatus;

var

activeEntry : ADEntry;

begin

activeEntry := ADEntry.Create(activeEntryName);

activeEntry.AddAttribute('isDefunct', ['TRUE']);

Result := ModifyEntryAttributes(activeEntry);

end;

{ Private }

procedure ADSchema.pvClearLDAPSearchResults(var searchResults : LDAPEntryArray);

var

iEntry : integer;

iAttribute : integer;

begin

for iEntry := 0 to Length(searchResults) - 1 do

begin

for iAttribute := 0 to searchResults[iEntry].attributes.Count - 1 do

begin

LDAPAttribute(searchResults[iEntry].attributes).Free;

LDAPAttribute(searchResults[iEntry].attributes) := nil;

end;

searchResults[iEntry].attributes.Free;

searchResults[iEntry].attributes := nil;

end;

SetLength(searchResults, 0);

end;

{ Private }

//search and get

function ADSchema.pvGetEntries(dn : string;

filter : string;

withAttr : array of string;

var status : ADSchemaStatus) : ADEntryList;

var

entryDN, searchFilter : string;

attrs : array of LDAPAttribute;

attr : LDAPAttribute;

iAttribute, iValue, iEntry : integer;

searchResults : LDAPEntryArray;

ldapStatus : LDAPClientStatus;

attributeIndex, entryIndex : integer;

parsedName : string;

begin

//Check Connection

if pvCheckConnection = false then

begin

status := pvTryConnect;

if status.StatusType <> SuccessStatus then

begin

result := nil;

Exit;

end;

if status <> nil then

begin

status.Free;

status := nil;

end;

if pvCheckConnection = false then

begin

result := nil;

status := ADSchemaStatus.Create(4, ADSchemaError, 'Cant get schema entries!');

Exit;

end;

end;

//forming entry dn

entryDN := dn;

//forming filter

searchFilter := filter;

//forming attributes array

SetLength(attrs, Length(withAttr));

for iAttribute := 0 to Length(withAttr) - 1 do

begin

attr := LDAPAttribute.Create(0);

attr.Name := withAttr[iAttribute];

attrs[iAttribute] := attr;

end;

//Call LDAPClient function

ldapStatus := client.Search(entryDN, searchFilter, attrs, searchResults);

if ldapStatus.numb <> 0 then

begin

status := ADSchemaStatus.Create(ldapStatus.numb, LDAPError, ldapStatus.msg);

result := nil;

Exit;

end;

//parse Search Results and form return object

result := ADEntryList.Create;

for iEntry := 0 to Length(searchResults) - 1 do

begin

parsedName := searchResults[iEntry].dn;

//delete 'CN='

Delete(parsedName,1,3);

//copy

parsedName := copy(parsedName, 1, pos(',CN=', parsedName)-1);

//create new entry

entryIndex := Result.AddEntry(parsedName);

for iAttribute := 0 to searchResults[iEntry].attributes.Count - 1 do

begin

attr := LDAPAttribute(searchResults[iEntry].attributes[iAttribute]);

attributeIndex := result.Items[entryIndex].AddAttribute(attr.Name);

for iValue := 0 to attr.ValueCount - 1 do

result.Items[entryIndex].Attributes[attributeIndex].AddValue(attr.Value[iValue]);

end;

end;

//Clearing return values of client.Search

pvClearLDAPSearchResults(searchResults);

status := ADSchemaStatus.Create;

end;

{ PRIVATE }

//Extended Search if too many values is expected

function ADSchema.pvSearchExtended(dn : string;

entrType : EntryType;

withAttr : array of string;

var status : ADSchemaStatus) : ADEntryList;

var

entryDN, searchFilter : string;

attrs : array of LDAPAttribute;

attr : LDAPAttribute;

iAttribute, iValue, iEntry, i: integer;

searchResults : LDAPEntryArray;

ldapStatus : LDAPClientStatus;

attributeIndex, entryIndex : integer;

parsedName : string;

helpString : string;

begin

helpString := 'abcdefghijklmnopqrstuvwxyz'; //26

//forming entry dn

entryDN := dn;

result := ADEntryList.Create;

for i := 1 to Length(helpString) do

begin

//forming filter

case entrType of

ClassEntry: searchFilter := '(&(objectClass=classSchema)(cn='+ helpString[i] + '\*))';

AttributeEntry: searchFilter := '(&(objectClass=attributeSchema)(cn='+ helpString[i] + '\*))';

end;

//forming attributes array

SetLength(attrs, Length(withAttr));

for iAttribute := 0 to Length(withAttr) - 1 do

begin

attr := LDAPAttribute.Create(0);

attr.Name := withAttr[iAttribute];

attrs[iAttribute] := attr;

end;

//Call LDAPClient function

ldapStatus := client.Search(entryDN, searchFilter, attrs, searchResults);

if ldapStatus.numb <> 0 then

begin

status := ADSchemaStatus.Create(ldapStatus.numb, LDAPError, ldapStatus.msg);

result.Destroy;

result := nil;

Exit;

end;

//parse Search Results and form return object

for iEntry := 0 to Length(searchResults) - 1 do

begin

parsedName := searchResults[iEntry].dn;

//delete 'CN='

Delete(parsedName,1,3);

//copy

parsedName := copy(parsedName, 1, pos(',CN=', parsedName)-1);

//create new entry

entryIndex := Result.AddEntry(parsedName);

for iAttribute := 0 to searchResults[iEntry].attributes.Count - 1 do

begin

attr := LDAPAttribute(searchResults[iEntry].attributes[iAttribute]);

attributeIndex := result.Items[entryIndex].AddAttribute(attr.Name);

for iValue := 0 to attr.ValueCount - 1 do

result.Items[entryIndex].Attributes[attributeIndex].AddValue(attr.Value[iValue]);

end;

end;

//Clearing return values of client.Search

pvClearLDAPSearchResults(searchResults);

end;

status := ADSchemaStatus.Create;

end;

{ Private }

//Checks if entry has all needed attributes

function ADSchema.pvCheckEntry(chEntry : ADEntry;

var neededAttributes : TStringList) : boolean;

type

objectType = (notDefined, attributeType, classType);

var

iAttribute, iValue, i : integer;

objectClasses : array of string;

oType : objectType;

isCnExist,

isObjectClassesExist,

isGovernsIDExist,

isAttributeIDExist,

isAttributeSyntaxExist,

isOMSyntaxExist : boolean;

begin

//Check if 'cn' attribute is set

//Check if 'objectClass' attriubute is set

isCnExist := false;

isObjectClassesExist := false;

isGovernsIDExist := false;

isAttributeIDExist := false;

isAttributeSyntaxExist := false;

isOMSyntaxExist := false;

for iAttribute := 0 to chEntry.AttributesCount - 1 do

begin

if not isObjectClassesExist then

begin

if chEntry.Attributes[iAttribute].Name = 'objectClass' then

begin

SetLength(objectClasses, chEntry.Attributes[iAttribute].ValuesCount);

for iValue := 0 to Length(objectClasses) - 1 do

begin

objectClasses[iValue] := chEntry.Attributes[iAttribute].Values[iValue];

end;

isObjectClassesExist := true;

end;

end;

if not isCnExist then

begin

if chEntry.Attributes[iAttribute].Name = 'cn' then

isCnExist := true;

end;

if not isGovernsIDExist then

begin

if chEntry.Attributes[iAttribute].Name = 'governsID' then

isGovernsIDExist := true;

end;

if not isAttributeIDExist then

begin

if chEntry.Attributes[iAttribute].Name = 'attributeID' then

isAttributeIDExist := true;

end;

if not isAttributeSyntaxExist then

begin

if chEntry.Attributes[iAttribute].Name = 'attributeSyntax' then

isAttributeSyntaxExist := true;

end;

if not isOMSyntaxExist then

begin

if chEntry.Attributes[iAttribute].Name = 'oMSyntax' then

isOMSyntaxExist := true;

end;

end;

if not isCnExist then

begin

neededAttributes := TStringList.Create;

neededAttributes.Add('cn');

result := false;

Exit;

end;

if not isObjectClassesExist then

begin

neededAttributes := TStringList.Create;

neededAttributes.Add('objectClass');

result := false;

Exit;

end;

{ write check of must attributes }

//Check existence of 'must' attributes

for i := 0 to Length(objectClasses) - 1 do

begin

if objectClasses[i] = 'classSchema' then

begin

oType := classType;

Break;

end;

if objectClasses[i] = 'attributeSchema' then

begin

oType := attributeType;

Break;

end;

end;

if oType = notDefined then

begin

neededAttributes := TStringList.Create;

neededAttributes.Add('classSchema');

neededAttributes.Add('attributeSchema');

result := false;

Exit;

end;

if oType = attributeType then

begin

{ Check existence:

\*attributeID

\*attributeSyntax

\*oMSyntax }

if not isAttributeIDExist or not isAttributeSyntaxExist or not isOMSyntaxExist then

begin

neededAttributes := TStringList.Create;

neededAttributes.Add('attributeID');

neededAttributes.Add('attributeSyntax');

neededAttributes.Add('oMSyntax');

result := false;

Exit;

end;

end;

if oType = classType then

begin

{ Check existence:

\*governsID }

if not isGovernsIDExist then

begin

neededAttributes := TStringList.Create;

neededAttributes.Add('governsID');

result := false;

Exit;

end;

end;

result := true;

end;

end.

**Модуль ADSchemaTypes.pas**

unit ADSchemaTypes;

interface

uses Windows, Classes;

type

{ Enum for error types }

ErrorTypeEnum = (

SuccessStatus,

LDAPError,

ADSchemaError);

{ Status model

\*gets error number and forms output message }

ADSchemaStatus = class

strict private

pvErrorNumb : integer;

pvErrorType : ErrorTypeEnum;

pvErrorMsg : string;

function pvGetLDAPMsg(errorNumb : integer) : string;

function pvGetADSchemaMsg(errorNumb : integer) : string;

public

property StatusNumb : integer

read pvErrorNumb;

property StatusType : ErrorTypeEnum

read pvErrorType;

property StatusMsg : string

read pvErrorMsg;

{ Creates object and sets pvErrorMsg by erNumb and erType,

or leave it as it is }

constructor Create(erNumb : integer;

erType : ErrorTypeEnum;

erMsg : string); overload;

{ Creates and sets pvErrorMsg=0 and pvErrorMsg=Success }

constructor Create(); overload;

end;

{ Attribute model }

ADAttribute = class

strict private

AttributeName : string;

AttributeValues : TStringList;

function GetValuesCount() : integer;

function GetValue(index : integer) : string;

procedure SetValue(index : integer; value : string);

public

property Name : string

read AttributeName

write AttributeName;

property Values[index : integer] : string

read GetValue

write SetValue;

property ValuesCount : integer

read GetValuesCount;

function GetList() : TStringList;

function SearchValue(value : string) : integer;

procedure AddValue(value : string);

procedure DeleteValue(valueIndex : integer);

constructor Create(attrName : string; attrValues : array of string); overload;

constructor Create(attrName : string); overload;

destructor Destroy(); override;

end;

{ Entry model }

ADEntry = class

strict private

EntryName : string;

EntryAttributes : TList;

function GetAttributesCount() : integer;

function GetAttribute(index : integer) : ADAttribute;

//Because of memory risks

//procedure SetAttribute(index : integer; value : ADAttribute);

public

property Name : string

read EntryName

write EntryName;

property Attributes[index : integer] : ADAttribute

read GetAttribute;

//write SetAttribute;

property AttributesCount : integer

read GetAttributesCount;

function GetList() : TList;

function GetAttributeByName(attrName : string) : ADAttribute;

function AddAttribute(attribute : ADAttribute) : integer; overload;

function AddAttribute(attrName : string; attrValues : array of string) : integer; overload;

function AddAttribute(attrName : string) : integer; overload;

procedure DeleteAttribute(index : integer); overload;

procedure DeleteAttribute(name : string); overload;

constructor Create(entrName : string); overload;

constructor Create(entrName : string; attrs : array of ADAttribute); overload;

destructor Destroy(); override;

end;

{ Entries List model }

ADEntryList = class

strict private

Entries : TList;

function GetEntriesCount() : integer;

function GetEntry(index : integer) : ADEntry;

public

property Items[index : integer] : ADEntry

read GetEntry;

property EntriesCount : integer

read GetEntriesCount;

function GetList() : TList;

function AddEntry(entry : ADEntry) : integer; overload;

function AddEntry(entryName : string) : integer; overload;

function AddEntry(entryName : string;

attrs : array of ADAttribute) : integer; overload;

procedure DeleteEntry(index : integer); overload;

procedure DeleteEntry(name : string); overload;

constructor Create();

destructor Destroy(); override;

end;

implementation

{ --------------------------------------------------------

------------------- IMPLEMENTATION ---------------------

------------------- ADSchemaStatus ---------------------

-------------------------------------------------------- }

{ Public }

constructor ADSchemaStatus.Create(erNumb : integer;

erType : ErrorTypeEnum;

erMsg : string);

var

res : string;

begin

pvErrorNumb := erNumb;

pvErrorType := erType;

pvErrorMsg := erMsg;

case erType of

LDAPError: res := pvGetLDAPMsg(erNumb);

ADSchemaError: res := pvGetADSchemaMsg(erNumb);

end;

if res <> '' then

pvErrorMsg := erMsg + ' --- ' + res;

end;

{ Public }

constructor ADSchemaStatus.Create();

begin

pvErrorNumb := 0;

pvErrorType := SuccessStatus;

pvErrorMsg := 'Success';

end;

{ Private }

function ADSchemaStatus.pvGetLDAPMsg(errorNumb : integer) : string;

var

res : string;

begin

//Make case of all LDAP errors { TODO: make message for every error }

case errorNumb of

1 : res := 'LDAP\_OPERATIONS\_ERROR: Indicates an internal error. The server is unable to respond with a more specific error and is also unable to properly respond to a request.';

2 : res := 'LDAP\_PROTOCOL\_ERROR: Indicates that the server has received an invalid or malformed request from the client.';

end;

Insert('LDAP ERROR: ', res, 0);

result := res;

end;

{ Private }

function ADSchemaStatus.pvGetADSchemaMsg(errorNumb : integer) : string;

var

res : string;

begin

//Make case of all ADSchema errors

case errorNumb of

1 : res := 'Cant set Schema DN!';

2 : res := 'Cant parse search results!';

3 : res := 'There is no all needed attributes in Entry!';

4 : res := 'Invalid password or user is not in schema admin group!';

end;

Insert('ADSchema ERROR: ', res, 0);

result := res;

end;

{ --------------------------------------------------------

------------------- IMPLEMENTATION ---------------------

------------------- ADAttribute ------------------------

-------------------------------------------------------- }

{ Public }

constructor ADAttribute.Create(attrName : string; attrValues : array of string);

var

i : integer;

begin

AttributeName := attrName;

AttributeValues := TStringList.Create;

for i := 0 to Length(attrValues) - 1 do

AttributeValues.Add(attrValues[i]);

end;

{ Public }

constructor ADAttribute.Create(attrName : string);

begin

AttributeName := attrName;

AttributeValues := TStringList.Create;

end;

{ Public }

destructor ADAttribute.Destroy();

begin

if AttributeValues <> nil then

begin

AttributeValues.Free;

AttributeValues := nil;

end;

inherited;

end;

{ Public }

function ADAttribute.GetList() : TStringList;

begin

result := AttributeValues;

end;

{ Public }

function ADAttribute.SearchValue(value : string) : integer;

var

i : integer;

begin

result := -1;

if AttributeValues.Count > 0 then

begin

for i := 0 to AttributeValues.Count - 1 do

begin

if AttributeValues[i] = value then

begin

result := i;

Break;

end;

end;

end;

end;

{ Public }

procedure ADAttribute.AddValue(value : string);

begin

AttributeValues.Add(value);

end;

{ Public }

procedure ADAttribute.DeleteValue(valueIndex : integer);

begin

AttributeValues.Delete(valueIndex);

end;

{ Private }

function ADAttribute.GetValuesCount() : integer;

begin

result := AttributeValues.Count;

end;

{ Private }

function ADAttribute.GetValue(index : integer) : string;

begin

result := AttributeValues[index];

end;

{ Private }

procedure ADAttribute.SetValue(index : integer; value : string);

begin

AttributeValues[index] := value;

end;

{ --------------------------------------------------------

------------------- IMPLEMENTATION ---------------------

------------------- ADEntry ----------------------------

-------------------------------------------------------- }

{ Public }

constructor ADEntry.Create(entrName : string);

begin

EntryName := entrName;

EntryAttributes := TList.Create;

end;

{ Public }

constructor ADEntry.Create(entrName : string; attrs : array of ADAttribute);

var

i : integer;

begin

EntryName := entrName;

EntryAttributes := TList.Create;

for i := 0 to Length(attrs) - 1 do

EntryAttributes.Add(attrs[i]);

end;

{ Public }

destructor ADEntry.Destroy();

var

i : integer;

begin

if (EntryAttributes <> nil) then

begin

if EntryAttributes.Count > 0 then

begin

for i := 0 to EntryAttributes.Count - 1 do

ADAttribute(EntryAttributes[i]).Destroy;

EntryAttributes.Clear;

end;

EntryAttributes.Free;

EntryAttributes := nil;

end;

inherited;

end;

{ Public }

function ADEntry.GetList() : TList;

begin

result := EntryAttributes;

end;

{ Public }

function ADEntry.GetAttributeByName(attrName : string) : ADAttribute;

var

i : integer;

begin

if AttributesCount > 0 then

for i := 0 to AttributesCount - 1 do

if Attributes[i].Name = attrName then

begin

result := Attributes[i];

Exit;

end;

result := nil;

end;

{ Public }

function ADEntry.AddAttribute(attribute : ADAttribute) : integer;

begin

EntryAttributes.Add(attribute);

result := EntryAttributes.Count - 1;

end;

{ Public }

function ADEntry.AddAttribute(attrName : string; attrValues : array of string) : integer;

begin

EntryAttributes.Add(ADAttribute.Create(attrName, attrValues));

result := EntryAttributes.Count - 1;

end;

{ Public }

function ADEntry.AddAttribute(attrName : string) : integer;

begin

EntryAttributes.Add(ADAttribute.Create(attrName));

result := EntryAttributes.Count - 1;

end;

{ Public }

procedure ADEntry.DeleteAttribute(index : integer);

begin

ADAttribute(EntryAttributes[index]).Destroy;

EntryAttributes.Delete(index);

end;

{ Public }

procedure ADEntry.DeleteAttribute(name : string);

var

i : integer;

begin

for i := 0 to EntryAttributes.Count - 1 do

begin

if ADAttribute(EntryAttributes[i]).Name = name then

begin

ADAttribute(EntryAttributes[i]).Destroy;

EntryAttributes.Delete(i);

break;

end;

end;

end;

{ Private }

function ADEntry.GetAttributesCount() : integer;

begin

result := EntryAttributes.Count;

end;

{ Private }

function ADEntry.GetAttribute(index : integer) : ADAttribute;

begin

result := ADAttribute(EntryAttributes[index]);

end;

{ --------------------------------------------------------

------------------- IMPLEMENTATION ---------------------

------------------- ADEntryList ------------------------

-------------------------------------------------------- }

{ Public }

constructor ADEntryList.Create();

begin

Entries := TList.Create;

end;

{ Public }

destructor ADEntryList.Destroy();

var

i : integer;

begin

if Entries <> nil then

begin

if Entries.count > 0 then

begin

for i := 0 to Entries.Count - 1 do

begin

ADEntry(Entries[i]).Destroy;

end;

Entries.Clear;

end;

Entries.Free;

Entries := nil;

end;

inherited;

end;

{ Public }

function ADEntryList.GetList() : TList;

begin

result := Entries;

end;

{ Public }

function ADEntryList.AddEntry(entry : ADEntry) : integer;

begin

Entries.Add(entry);

result := Entries.Count - 1;

end;

{ Public }

function ADEntryList.AddEntry(entryName : string) : integer;

begin

Entries.Add(ADEntry.Create(entryName));

result := Entries.Count - 1;

end;

{ Public }

function ADEntryList.AddEntry(entryName : string; attrs : array of ADAttribute) : integer;

begin

Entries.Add(ADEntry.Create(entryName, attrs));

result := Entries.Count - 1;

end;

{ Public }

procedure ADEntryList.DeleteEntry(index : integer);

begin

ADEntry(Entries[index]).Destroy;

Entries.Delete(index);

end;

{ Public }

procedure ADEntryList.DeleteEntry(name : string);

var

i : integer;

begin

for i := 0 to Entries.Count - 1 do

begin

if ADEntry(Entries[i]).Name = name then

begin

ADEntry(Entries[i]).Destroy;

Entries.Delete(i);

break;

end;

end;

end;

{ Private }

function ADEntryList.GetEntriesCount() : integer;

begin

result := Entries.Count;

end;

{ Private }

function ADEntryList.GetEntry(index : integer) : ADEntry;

begin

result := ADEntry(Entries[index]);

end;

end.

**Модуль ADSchemaHelpUnit.pas**

unit ADSchemaHelpUnit;

interface

uses Windows;

const

NameUnknown = 0;

NameFullyQualifiedDN = 1;

NameSamCompatible = 2;

NameDisplay = 3;

NameUniqueId = 6;

NameCanonical = 7;

NameUserPrincipal = 8;

NameCanonicalEx = 9;

NameServicePrincipal = 10;

NameDnsDomain = 12;

//function to get current user DN

function GetUserNameExString(ANameFormat: DWORD): string;

implementation

function GetUserNameExString(ANameFormat: DWORD): string;

var

Buf: array[0..256] of Char;

BufSize: DWORD;

GetUserNameEx: function (NameFormat: DWORD; lpNameBuffer: LPSTR;

var nSize: ULONG): BOOL; stdcall;

begin

Result := '';

BufSize := SizeOf(Buf) div SizeOf(Buf[0]);

GetUserNameEx := GetProcAddress(GetModuleHandle('secur32.dll'), 'GetUserNameExA');

if Assigned(GetUserNameEx) then

if GetUserNameEx(ANameFormat, Buf, BufSize) then

Result := Buf;

end;

end.

**Модуль MainPage.pas**

unit MainPage;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, ComCtrls, ExtCtrls, Grids, ToolWin, ADSchemaUnit, ADSchemaTypes,

Menus, ImgList, ConnectPage, Errorpage, AddPage;

type

TStringGrid = class(Grids.TStringGrid)

public

function SelectCell(ACol: Integer; ARow: Integer): Boolean; override;

procedure KeyPress(var Key: Char); override;

end;

TMainForm = class(TForm)

GridPanel1: TGridPanel;

TreeViewMain: TTreeView;

StringGridMain: TStringGrid;

ToolBar1: TToolBar;

AddButton: TToolButton;

ModifyButton: TToolButton;

ToolButton3: TToolButton;

ActivateButton: TToolButton;

DeactivateButton: TToolButton;

ToolButton6: TToolButton;

RefreshButton: TToolButton;

ToolButton8: TToolButton;

ConnectButton: TToolButton;

DisconnectButton: TToolButton;

ToolButton11: TToolButton;

ExitButton: TToolButton;

MainMenu1: TMainMenu;

sss: TMenuItem;

Start1: TMenuItem;

Connect1: TMenuItem;

Disconnect1: TMenuItem;

Exit1: TMenuItem;

Help1: TMenuItem;

About1: TMenuItem;

Add1: TMenuItem;

Modify1: TMenuItem;

Activate1: TMenuItem;

Deactivate1: TMenuItem;

Refresh1: TMenuItem;

Howto1: TMenuItem;

AttributeSyntaxes1: TMenuItem;

ImageListTool: TImageList;

procedure FormCreate(Sender: TObject);

procedure FormResize(Sender: TObject);

procedure TreeViewMainChange(Sender: TObject; Node: TTreeNode);

procedure FormClose(Sender: TObject; var Action: TCloseAction);

procedure StringGridMainDrawCell(Sender: TObject; ACol, ARow: Integer;

Rect: TRect; State: TGridDrawState);

procedure Exit1Click(Sender: TObject);

procedure Disconnect1Click(Sender: TObject);

procedure Connect1Click(Sender: TObject);

procedure StringGridMainDblClick(Sender: TObject);

procedure StringGridMainSelectCellWithEdit(Sender: TObject; ACol, ARow: Integer;

var CanSelect: Boolean);

procedure Activate1Click(Sender: TObject);

procedure Deactivate1Click(Sender: TObject);

procedure Refresh1Click(Sender: TObject);

procedure Add1Click(Sender: TObject);

procedure Modify1Click(Sender: TObject);

private

{ Private declarations }

ATTRIBUTE\_NODE : TTreeNode;

CLASS\_NODE : TTreeNode;

TREE\_NODE : TTreeNode;

TreeViewCurrentNode : TTreeNode;

schema : ADSchema;

procedure pvResizeGridColumns();

procedure pvConnect();

procedure pvLoadItemsToTree();

procedure pvSetClassesToTable();

procedure pvSetAttributesToTable();

procedure pvSetEntryInfoToTable(entryName: string);

procedure pvSetChildClassesToTable(parentClassName : string);

procedure pvChangeDisconnectButtonsState(state : boolean);

procedure pvClearStringGridTable();

//Error procedure

procedure pvShowErrorDialog(errorNumb: integer; errorMsg: string);

public

{ Public declarations }

procedure LoadInOtherThread;

procedure SaveTable();

end;

TLoadThread = class(TThread)

protected

procedure Execute; override;

end;

var

MainForm: TMainForm;

implementation

{$R \*.dfm}

{ Working with thread

LOADING elements of TreeView component }

procedure TLoadThread.Execute;

begin

MainForm.LoadInOtherThread;

end;

procedure TMainForm.LoadInOtherThread;

begin

pvLoadItemsToTree;

end;

procedure TMainForm.Modify1Click(Sender: TObject);

var

addForm : TAddForm;

begin

addForm := TAddForm.Create(Application);

addForm.schema := schema;

addForm.isModify := true;

//TODO: set entry Name

with StringGridMain do

begin

if (rowCount > 1) and (row > 0) then

begin

addForm.entryName := Cells[0, Row];

end else

if (TreeViewCurrentNode <> CLASS\_NODE) and

(TreeViewCurrentNode <> ATTRIBUTE\_NODE) and

(TreeViewCurrentNode <> TREE\_NODE) then

if TreeViewCurrentNode.Selected then

begin

addForm.entryName := TreeViewCurrentNode.Text;

end;

end;

try

if addForm.ShowModal = mrOk then

Refresh1.Click;

finally

addForm.Free;

end;

end;

{ TStringGrid }

function TStringGrid.SelectCell(ACol, ARow: Integer): Boolean;

begin

if (EditorMode=True) then MainForm.SaveTable;

result:=inherited SelectCell(ACol, ARow);

end;

procedure TStringGrid.KeyPress(var Key: Char);

begin

if key=#13 then MainForm.SaveTable;

inherited;

end;

{ TMainForm }

procedure TMainForm.pvChangeDisconnectButtonsState(state : boolean);

begin

Disconnect1.Enabled := state;

DisconnectButton.Enabled := state;

end;

//Error handling

procedure TMainForm.pvShowErrorDialog(errorNumb: integer; errorMsg: string);

var

errorForm : TErrorForm;

begin

errorForm := TErrorForm.Create(Application);

errorForm.SetErrorInfo(errorNumb, errorMsg);

try

errorForm.ShowModal;

finally

errorForm.Free;

end;

end;

procedure TMainForm.Refresh1Click(Sender: TObject);

var

newThread : TLoadThread;

begin

ATTRIBUTE\_NODE.DeleteChildren;

CLASS\_NODE.DeleteChildren;

TREE\_NODE.DeleteChildren;

pvClearStringGridTable();

newThread := TLoadThread.Create(false);

end;

//Getting info from form

procedure TMainForm.pvConnect;

var

conForm : TConnectForm;

modalRes : integer;

newThread : TLoadThread;

begin

conForm := TConnectForm.Create(Application);

try

modalRes := conform.ShowModal;

if modalRes = mrOk then

begin

schema := conform.GetSchema;

conForm.Close;

//load Tree View

//pvLoadItemsToTree;

newThread := TLoadThread.Create(false);

pvChangeDisconnectButtonsState(true);

end else

if modalRes = mrCancel then

begin

conForm.Close;

end;

finally

conform.Free;

end;

end;

procedure TMainForm.Activate1Click(Sender: TObject);

var status : ADSchemaStatus;

modEntry : ADEntry;

begin

if schema = nil then

Exit;

if (TreeViewCurrentNode <> CLASS\_NODE) and

(TreeViewCurrentNode <> ATTRIBUTE\_NODE) and

(TreeViewCurrentNode <> TREE\_NODE) then

begin

with StringGridMain do

begin

if (rowCount > 1) and (row > 0) then

begin

modEntry := ADEntry.Create(Cells[0, Row]);

modEntry.AddAttribute('isDefunct', ['FALSE']);

status := schema.ModifyEntryAttributes(modEntry);

modEntry.Destroy;

modEntry := nil;

if status.StatusType <> SuccessStatus then

pvShowErrorDialog(status.StatusNumb, status.StatusMsg);

status.Free;

status := nil;

end else

if TreeViewCurrentNode.Selected then

begin

modEntry := ADEntry.Create(TreeViewCurrentNode.Text);

modEntry.AddAttribute('isDefunct', ['FALSE']);

status := schema.ModifyEntryAttributes(modEntry);

modEntry.Destroy;

modEntry := nil;

if status.StatusType <> SuccessStatus then

pvShowErrorDialog(status.StatusNumb, status.StatusMsg);

status.Free;

status := nil;

end;

end;

end;

end;

procedure TMainForm.Add1Click(Sender: TObject);

var

addForm : TAddForm;

begin

addForm := TAddForm.Create(Application);

addForm.schema := schema;

addForm.isModify := false;

try

if addForm.ShowModal = mrOk then

Refresh1.Click;

finally

addForm.Free;

end;

end;

procedure TMainForm.Connect1Click(Sender: TObject);

begin

pvConnect;

end;

procedure TMainForm.Deactivate1Click(Sender: TObject);

var status : ADSchemaStatus;

modEntry : ADEntry;

begin

if schema = nil then

Exit;

if (TreeViewCurrentNode <> CLASS\_NODE) and

(TreeViewCurrentNode <> ATTRIBUTE\_NODE) and

(TreeViewCurrentNode <> TREE\_NODE) then

begin

with StringGridMain do

begin

if (rowCount > 1) and (row > 0) then

begin

modEntry := ADEntry.Create(Cells[0, Row]);

modEntry.AddAttribute('isDefunct', ['TRUE']);

status := schema.ModifyEntryAttributes(modEntry);

modEntry.Destroy;

modEntry := nil;

if status.StatusType <> SuccessStatus then

pvShowErrorDialog(status.StatusNumb, status.StatusMsg);

status.Free;

status := nil;

end else

if TreeViewCurrentNode.Selected then

begin

modEntry := ADEntry.Create(TreeViewCurrentNode.Text);

modEntry.AddAttribute('isDefunct', ['TRUE']);

status := schema.ModifyEntryAttributes(modEntry);

modEntry.Destroy;

modEntry := nil;

if status.StatusType <> SuccessStatus then

pvShowErrorDialog(status.StatusNumb, status.StatusMsg);

status.Free;

status := nil;

end;

end;

end;

end;

procedure TMainForm.Disconnect1Click(Sender: TObject);

begin

if schema <> nil then

schema.Disconnect;

ATTRIBUTE\_NODE.DeleteChildren;

CLASS\_NODE.DeleteChildren;

TREE\_NODE.DeleteChildren;

pvChangeDisconnectButtonsState(false);

pvClearStringGridTable();

end;

procedure TMainForm.Exit1Click(Sender: TObject);

begin

MainForm.Close;

end;

procedure TMainForm.FormClose(Sender: TObject; var Action: TCloseAction);

begin

if schema <> nil then

schema.Destroy;

end;

procedure TMainForm.FormCreate(Sender: TObject);

var

//status : ADSchemaStatus;

firstNode : TTreeNode;

begin

Application.HintColor := clGrayText;

Application.HintPause := 250;

pvResizeGridColumns;

firstNode := TreeViewMain.Items.GetFirstNode;

ATTRIBUTE\_NODE := TreeViewMain.Items.AddChild(firstNode, 'Attributes');

CLASS\_NODE := TreeViewMain.Items.AddChild(firstNode, 'Classes');

TREE\_NODE := TreeViewMain.Items.AddChild(firstNode, 'ClassTree');

StringGridMain.Cells[0, 0] := 'Name';

StringGridMain.Cells[1, 0] := 'Value';

end;

procedure TMainForm.FormResize(Sender: TObject);

begin

pvResizeGridColumns;

end;

{ PRIVATE Resize function for stringGrid columns }

procedure TMainForm.pvResizeGridColumns();

begin

StringGridMain.ColWidths[0] := trunc(StringGridMain.Width\*0.29);

if StringGridMain.ColCount = 2 then

begin

StringGridMain.ColWidths[1] := trunc(StringGridMain.Width\*0.67);

end;

if StringGridMain.ColCount = 3 then

begin

StringGridMain.ColWidths[1] := trunc(StringGridMain.Width\*0.37);

StringGridMain.ColWidths[2] := trunc(StringGridMain.Width\*0.3);

end;

end;

procedure TMainForm.StringGridMainDblClick(Sender: TObject);

begin

with StringGridMain do

begin

//Cells[0,0] := IntToStr(Col) + ' ' + IntToStr(Row);

if Col = 0 then

if Row > 0 then

begin

if ColCount <> 2 then

pvSetEntryInfoToTable(StringGridMain.Cells[col, row]);

end;

end;

end;

procedure TMainForm.StringGridMainDrawCell(Sender: TObject; ACol, ARow: Integer;

Rect: TRect; State: TGridDrawState);

var

AGrid : TStringGrid;

begin

AGrid := TStringGrid(Sender);

if ARow = 0 then

begin

AGrid.Canvas.Brush.Color := clBtnFace;

AGrid.Canvas.FillRect(Rect);

AGrid.Canvas.TextOut(Rect.Left + 2, Rect.Top + 2, AGrid.Cells[ACol, ARow]);

end;

end;

procedure TMainForm.StringGridMainSelectCellWithEdit(Sender: TObject; ACol,

ARow: Integer; var CanSelect: Boolean);

begin

if ACol = 0 then

StringGridMain.Options := StringGridMain.Options-[goEditing]

else

StringGridMain.Options := StringGridMain.Options+[goEditing];

end;

procedure TMainForm.TreeViewMainChange(Sender: TObject; Node: TTreeNode);

begin

{ Controling wich node is selected in tree }

TreeViewCurrentNode := Node;

StringGridMain.Selection := TGridRect(Rect(-1,-1,-1,-1));

if schema <> nil then

if schema.isActive then

begin

//Change the table content

if TreeViewCurrentNode = CLASS\_NODE then

pvSetClassesToTable

else

if TreeViewCurrentNode = ATTRIBUTE\_NODE then

pvSetAttributesToTable

else

if (TreeViewCurrentNode.Parent = CLASS\_NODE)

or (TreeViewCurrentNode.Parent = ATTRIBUTE\_NODE) then

pvSetEntryInfoToTable(TreeViewCurrentNode.Text)

else

if TreeViewCurrentNode <> TreeViewMain.Items.GetFirstNode then

pvSetChildClassesToTable(TreeViewCurrentNode.Text);

end;

end;

{ PRIVATE Load TreeView Items function }

procedure TMainForm.pvLoadItemsToTree();

var

entries : ADEntryList;

status : ADSchemaStatus;

iEntry : integer;

topEntry : ADEntry;

curNode, parentNode : TTreeNode;

test : string;

begin

//Get Attributes

if schema = nil then

Exit;

entries := schema.GetAll(AttributeEntry, ['cn'], status);

if status.StatusType <> SuccessStatus then

begin

pvShowErrorDialog(status.StatusNumb, status.StatusMsg);

status.Free;

status := nil;

if entries <> nil then

begin

entries.Destroy;

entries := nil;

end;

Exit;

end;

status.Free;

status := nil;

if entries <> nil then

if entries.EntriesCount > 0 then

begin

for iEntry := 0 to entries.EntriesCount - 1 do

begin

TreeViewMain.Items.AddChild(ATTRIBUTE\_NODE, entries.Items[iEntry].Name);

end;

entries.Destroy;

entries := nil;

end;

//Get Classes

entries := schema.GetAll(ClassEntry, ['cn'], status);

if status.StatusType <> SuccessStatus then

begin

pvShowErrorDialog(status.StatusNumb, status.StatusMsg);

status.Free;

status := nil;

if entries <> nil then

begin

entries.Destroy;

entries := nil;

end;

Exit;

end;

status.Free;

status := nil;

if entries <> nil then

if entries.EntriesCount > 0 then

begin

for iEntry := 0 to entries.EntriesCount - 1 do

begin

TreeViewMain.Items.AddChild(CLASS\_NODE, entries.Items[iEntry].Name);

end;

entries.Destroy;

entries := nil;

end;

//----------------------------------------------------------

//---------------- Form Class Tree -----------------------

// -----------------------------------------------------

curNode := TreeViewMain.Items.AddChild(TREE\_NODE, 'top');

//top class childs

entries := schema.GetEntries('(&(objectClass=classSchema)(subClassOf=top))', ['cn'], status);

if status.StatusType <> SuccessStatus then

begin

pvShowErrorDialog(status.StatusNumb, status.StatusMsg);

status.Free;

status := nil;

if entries <> nil then

begin

entries.Destroy;

entries := nil;

end;

Exit;

end;

status.Free;

status := nil;

if entries <> nil then

if entries.EntriesCount > 0 then

begin

for iEntry := 0 to entries.EntriesCount - 1 do

begin

if entries.Items[iEntry].Name <> 'Top' then

TreeViewMain.Items.AddChild(curNode, entries.Items[iEntry].Name);

end;

entries.Destroy;

entries := nil;

// while loop to form other subclasses

parentNode := curNode;

curNode := curNode.getFirstChild;

while curNode.Text <> 'top' do

begin

if curNode.Text <> parentNode.Text then

begin

entries := schema.GetEntries('(&(objectClass=classSchema)(subClassOf=' + curNode.Text + '))', ['cn'], status);

status.free;

status := nil;

if (entries <> nil) then

if entries.EntriesCount > 0 then

begin

for iEntry := 0 to entries.EntriesCount - 1 do

begin

TreeViewMain.Items.AddChild(curNode, entries.Items[iEntry].Name);

end;

entries.Destroy;

entries := nil;

parentNode := curNode;

curNode := curNode.getFirstChild;

Continue;

end;

end

else parentNode := curNode.Parent;

if curNode.Text = parentNode.GetLastChild.Text then

begin

curNode := parentNode;

continue;

end

else curNode := curNode.getNextSibling;

end; { while end }

end;

end;

procedure TMainForm.pvSetClassesToTable();

var

entries : ADEntryList;

status : ADSchemaStatus;

iEntry, iGridRowCount : integer;

curAttrValue : string;

curAttr : ADAttribute;

begin

//GET Classes

entries := schema.GetAll(ClassEntry, ['isDefunct', 'objectClassCategory'], status);

if status.StatusType <> SuccessStatus then

begin

pvShowErrorDialog(status.StatusNumb, status.StatusMsg);

status.Free;

status := nil;

if entries <> nil then

begin

entries.Destroy;

entries := nil;

end;

Exit;

end;

status.Free;

status := nil;

if entries = nil then

Exit;

//Fill The Table

pvClearStringGridTable;

StringGridMain.ColCount := 3;

StringGridMain.Cells[0,0] := 'Class Name';

StringGridMain.Cells[1,0] := 'Type';

StringGridMain.Cells[2,0] := 'Status';

pvResizeGridColumns;

StringGridMain.Options := StringGridMain.Options-[goEditing];

//StringGridMain.OnSelectCell := StringGridMainSelectCell;

StringGridMain.RowCount := entries.EntriesCount + 1;

iGridRowCount := 0;

if entries.EntriesCount <> 0 then

begin

for iEntry := 0 to entries.EntriesCount - 1 do

begin

iGridRowCount := iGridRowCount + 1;

StringGridMain.Cells[0, iGridRowCount] := entries.Items[iEntry].Name;

if (entries.Items[iEntry].AttributesCount > 0) then

begin

curAttr := entries.Items[iEntry].GetAttributeByName('objectClassCategory');

if curAttr <> nil then

begin

if curAttr.ValuesCount > 0 then

begin

curAttrValue := curAttr.Values[0];

if curAttrValue = '0' then

StringGridMain.Cells[1, iGridRowCount] := 'Type88'

else if curAttrValue = '1' then

StringGridMain.Cells[1, iGridRowCount] := 'Structural'

else if curAttrValue = '2' then

StringGridMain.Cells[1, iGridRowCount] := 'Abstract'

else if curAttrValue = '3' then

StringGridMain.Cells[1, iGridRowCount] := 'Auxiliary';

end; {curAttr.ValuesCount > 0}

end; {curAttr <> nil}

curAttr := entries.Items[iEntry].GetAttributeByName('isDefunct');

if curAttr <> nil then

begin

if curAttr.ValuesCount > 0 then

begin

if curAttr.Values[0] = 'FALSE' then

StringGridMain.Cells[2, iGridRowCount] := 'Active'

else

StringGridMain.Cells[2, iGridRowCount] := 'Deactivated';

end;

end {curAttr <> nil}

else StringGridMain.Cells[2, iGridRowCount] := 'Active';

end; {(entries.Items[iEntry].AttributesCount > 0)}

end; {for iEntry := 0 to entries.EntriesCount - 1 do}

end; {entries.EntriesCount <> 0}

if entries <> nil then

begin

entries.Destroy;

entries := nil;

end;

end;

procedure TMainForm.pvSetAttributesToTable();

var

entries : ADEntryList;

status : ADSchemaStatus;

iEntry, iGridRowCount, i : integer;

//curAttrValue : string;

curAttr : ADAttribute;

begin

//GET Attributes

entries := schema.GetAll(AttributeEntry, ['isDefunct', 'attributeSyntax'], status);

if status.StatusType <> SuccessStatus then

begin

pvShowErrorDialog(status.StatusNumb, status.StatusMsg);

status.Free;

status := nil;

if entries <> nil then

begin

entries.Destroy;

entries := nil;

end;

Exit;

end;

status.Free;

status := nil;

if entries = nil then

Exit;

//Fill The Table

pvClearStringGridTable;

StringGridMain.ColCount := 3;

StringGridMain.Cells[0,0] := 'Attribute Name';

StringGridMain.Cells[1,0] := 'Syntax';

StringGridMain.Cells[2,0] := 'Status';

pvResizeGridColumns;

StringGridMain.Options := StringGridMain.Options-[goEditing];

//StringGridMain.OnSelectCell := StringGridMainSelectCell;

StringGridMain.RowCount := entries.EntriesCount + 1;

iGridRowCount := 0;

if entries.EntriesCount <> 0 then

begin

for iEntry := 0 to entries.EntriesCount - 1 do

begin

iGridRowCount := iGridRowCount + 1;

StringGridMain.Cells[0, iGridRowCount] := entries.Items[iEntry].Name;

if (entries.Items[iEntry].AttributesCount > 0) then

begin

curAttr := entries.Items[iEntry].GetAttributeByName('attributeSyntax');

if curAttr <> nil then

begin

if curAttr.ValuesCount > 0 then

begin

StringGridMain.Cells[1, iGridRowCount] := curAttr.Values[0];

end; {curAttr.ValuesCount > 0}

end; {curAttr <> nil}

curAttr := entries.Items[iEntry].GetAttributeByName('isDefunct');

if curAttr <> nil then

begin

if curAttr.ValuesCount > 0 then

begin

if curAttr.Values[0] = 'FALSE' then

StringGridMain.Cells[2, iGridRowCount] := 'Active'

else

StringGridMain.Cells[2, iGridRowCount] := 'Deactivated';

end;

end {curAttr <> nil}

else StringGridMain.Cells[2, iGridRowCount] := 'Active';

end; {(entries.Items[iEntry].AttributesCount > 0)}

end; {for iEntry := 0 to entries.EntriesCount - 1 do}

end; {entries.EntriesCount <> 0}

if entries <> nil then

begin

entries.Destroy;

entries := nil;

end;

end;

procedure TMainForm.pvSetEntryInfoToTable(entryName : string);

var

entrType : EntryType;

entries : ADEntryList;

entry : ADEntry;

status : ADSchemaStatus;

iGridRowCount, iAttribute, iValue : integer;

attribute : ADAttribute;

begin

// set entry info

entries := schema.GetEntries('(cn=' + entryName + ')', [], status);

//entry := schema.GetEntry(entryName, entrType, status);

if status.StatusType <> SuccessStatus then

begin

pvShowErrorDialog(status.StatusNumb, status.StatusMsg);

status.Free;

status := nil;

if entry <> nil then

begin

entries.Destroy;

entries := nil;

end;

Exit;

end;

status.Free;

status := nil;

if entries = nil then

Exit;

if entries.EntriesCount <> 1 then

Exit;

entry := entries.Items[0];

//Fill The Table

pvClearStringGridTable;

StringGridMain.ColCount := 2;

StringGridMain.Cells[0,0] := 'Attribute Name';

StringGridMain.Cells[1,0] := 'Attribute Value';

pvResizeGridColumns;

StringGridMain.Options := StringGridMain.Options+[goEditing];

StringGridMain.OnSelectCell := StringGridMainSelectCellWithEdit;

StringGridMain.RowCount := entry.AttributesCount + 1;

iGridRowCount := 0;

if entry.AttributesCount <> 0 then

begin

for iAttribute := 0 to entry.AttributesCount - 1 do

begin

attribute := entry.Attributes[iAttribute];

if attribute.ValuesCount > 0 then

begin

if attribute.ValuesCount > 1 then

StringGridMain.RowCount := StringGridMain.RowCount + (attribute.ValuesCount-1);

for iValue := 0 to attribute.ValuesCount - 1 do

begin

iGridRowCount := iGridRowCount + 1;

StringGridMain.Cells[0, iGridRowCount] := attribute.Name;

StringGridMain.Cells[1, iGridRowCount] := attribute.Values[iValue];

end;

end;

end; {for iEntry := 0 to entries.EntriesCount - 1 do}

end; {entries.EntriesCount <> 0}

if entry <> nil then

begin

entry.Destroy;

entry := nil;

end;

end;

procedure TMainForm.pvSetChildClassesToTable(parentClassName : string);

var

entries : ADEntryList;

status : ADSchemaStatus;

iEntry, iGridRowCount : integer;

curAttrValue : string;

curAttr : ADAttribute;

filter : string;

begin

//set clild classes

if parentClassName = TREE\_NODE.Text then

filter := '(&(objectClass=classSchema)(cn=top))'

else

filter := '(&(objectClass=classSchema)(subClassOf=' + parentClassName + '))';

//GET Child Classes

entries := schema.GetEntries(filter, ['isDefunct', 'objectClassCategory'], status);

if status.StatusType <> SuccessStatus then

begin

pvShowErrorDialog(status.StatusNumb, status.StatusMsg);

status.Free;

status := nil;

if entries <> nil then

begin

entries.Destroy;

entries := nil;

end;

Exit;

end;

status.Free;

status := nil;

if entries = nil then

Exit;

//Fill The Table

pvClearStringGridTable;

StringGridMain.ColCount := 3;

StringGridMain.Cells[0,0] := 'Class Name';

StringGridMain.Cells[1,0] := 'Type';

StringGridMain.Cells[2,0] := 'Status';

pvResizeGridColumns;

StringGridMain.Options := StringGridMain.Options-[goEditing];

//StringGridMain.OnSelectCell := StringGridMainSelectCell;

StringGridMain.RowCount := entries.EntriesCount + 1;

iGridRowCount := 0;

if entries.EntriesCount <> 0 then

begin

for iEntry := 0 to entries.EntriesCount - 1 do

begin

iGridRowCount := iGridRowCount + 1;

StringGridMain.Cells[0, iGridRowCount] := entries.Items[iEntry].Name;

if (entries.Items[iEntry].AttributesCount > 0) then

begin

curAttr := entries.Items[iEntry].GetAttributeByName('objectClassCategory');

if curAttr <> nil then

begin

if curAttr.ValuesCount > 0 then

begin

curAttrValue := curAttr.Values[0];

if curAttrValue = '0' then

StringGridMain.Cells[1, iGridRowCount] := 'Type88'

else if curAttrValue = '1' then

StringGridMain.Cells[1, iGridRowCount] := 'Structural'

else if curAttrValue = '2' then

StringGridMain.Cells[1, iGridRowCount] := 'Abstract'

else if curAttrValue = '3' then

StringGridMain.Cells[1, iGridRowCount] := 'Auxiliary';

end; {curAttr.ValuesCount > 0}

end; {curAttr <> nil}

curAttr := entries.Items[iEntry].GetAttributeByName('isDefunct');

if curAttr <> nil then

begin

if curAttr.ValuesCount > 0 then

begin

if curAttr.Values[0] = 'FALSE' then

StringGridMain.Cells[2, iGridRowCount] := 'Active'

else

StringGridMain.Cells[2, iGridRowCount] := 'Deactivated';

end;

end {curAttr <> nil}

else StringGridMain.Cells[2, iGridRowCount] := 'Active';

end; {(entries.Items[iEntry].AttributesCount > 0)}

end; {for iEntry := 0 to entries.EntriesCount - 1 do}

end; {entries.EntriesCount <> 0}

if entries <> nil then

begin

entries.Destroy;

entries := nil;

end;

end;

procedure TMainForm.pvClearStringGridTable();

var

i : integer;

begin

with StringGridMain do

begin

for i:=0 to RowCount-1 do

Rows[i].Clear;

StringGridMain.RowCount := 2;

end;

end;

procedure TMainForm.SaveTable();

var

attrName, attrValue, entryName : string;

status : ADSchemaStatus;

modEntry : ADEntry;

whileRow : integer;

attrIndex : integer;

begin

with StringGridMain do

begin

if Cells[0,3] = 'cn' then

begin

attrValue := Cells[Col, Row];

attrName := Cells[0, Row];

entryName := Cells[1,3];

modEntry := ADEntry.Create(entryName);

attrIndex := modEntry.AddAttribute(attrName, [attrValue]);

if Cells[0, Row-1] = attrName then

begin

whileRow := Row-1;

while Cells[0, whileRow] = attrName do

begin

modEntry.Attributes[attrIndex].AddValue(Cells[1, whileRow]);

whileRow := whileRow - 1;

if whileRow <= 0 then

break;

end;

end;

if (Row+1) < (RowCount-1) then

if Cells[0, Row + 1] = attrName then

begin

whileRow := Row+1;

while Cells[0, whileRow] = attrName do

begin

modEntry.Attributes[attrIndex].AddValue(Cells[1, whileRow]);

whileRow := whileRow + 1;

if whileRow > (RowCount-1) then

break;

end;

end;

status := schema.ModifyEntryAttributes(modEntry);

modEntry.Destroy;

modEntry := nil;

if status.StatusType <> SuccessStatus then

begin

pvShowErrorDialog(status.StatusNumb, status.StatusMsg);

end;

status.Free;

status := nil;

end;

end;

end;

end.

**Модуль ConnectPage.pas**

unit ConnectPage;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, StdCtrls, Buttons, ADSchemaUnit, ADSchemaTypes, ErrorPage;

type

TConnectForm = class(TForm)

EditHost: TEdit;

EditUsername: TEdit;

EditPassword: TEdit;

LabelHost: TLabel;

LabelUsername: TLabel;

LabelPassword: TLabel;

ConnectBtn: TButton;

ErrorLabel: TLabel;

GroupBox1: TGroupBox;

GroupBox2: TGroupBox;

procedure ConnectBtnClick(Sender: TObject);

procedure ErrorLabelClick(Sender: TObject);

procedure FormClose(Sender: TObject; var Action: TCloseAction);

private

{ Private declarations }

schema : ADSchema;

status : ADSchemaStatus;

procedure pvShowErrorDialog(errorNumb: integer; errorMsg: string);

public

{ Public declarations }

property GetSchema : ADSchema

read schema;

end;

var

ConnectForm: TConnectForm;

implementation

{$R \*.dfm}

//Error handling

procedure TConnectForm.pvShowErrorDialog(errorNumb: integer; errorMsg: string);

var

errorForm : TErrorForm;

begin

errorForm := TErrorForm.Create(Application);

errorForm.SetErrorInfo(errorNumb, errorMsg);

try

errorForm.ShowModal;

finally

errorForm.Free;

end;

end;

procedure TConnectForm.ConnectBtnClick(Sender: TObject);

var

hostName, userName, password : string;

portNumber : integer;

entry : ADEntry;

begin

hostName := EditHost.Text;

userName := EditUsername.Text;

password := EditPassword.Text;

portNumber := 389;

schema := ADSchema.Create(hostName, userName, password, portNumber, status);

if (status.StatusType <> SuccessStatus) then

begin

ErrorLabel.Visible := true

end

else

begin

status.Free;

status := nil;

//check is user in schema admin group

entry := schema.GetEntry('account', ClassEntry, ['cn'], status);

if status.StatusType <> SuccessStatus then

ErrorLabel.Visible := true

else ModalResult := mrOk;

end;

end;

procedure TConnectForm.ErrorLabelClick(Sender: TObject);

begin

pvShowErrorDialog(status.StatusNumb, status.StatusMsg);

end;

procedure TConnectForm.FormClose(Sender: TObject; var Action: TCloseAction);

begin

if status <> nil then

begin

status.free;

status := nil;

end;

end;

end.

**Модуль AddPage.pas**

unit AddPage;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, ComCtrls, StdCtrls, Buttons, ExtCtrls, ADSchemaUnit, ADSchemaTypes,

ErrorPage, SelectPage;

type

OptionClass = class

comboBox : TComboBox;

edit : TEdit;

deleteBtn : TSpeedButton;

onLoadEditText : string;

public

constructor Create(cbx: TComboBox; ed: TEdit; dbtn: TSpeedButton; onLoadText : string);

end;

TAddForm = class(TForm)

PageControl1: TPageControl;

TabSheet1: TTabSheet;

LabelName: TLabel;

LabelType: TLabel;

EditName: TEdit;

ComboBoxType: TComboBox;

GroupBox1: TGroupBox;

OIDLabel: TLabel;

SyntaxLabel: TLabel;

oMSyntaxLabel: TLabel;

OIDEdit: TEdit;

SyntaxEdit: TEdit;

oMSyntaxEdit: TEdit;

FlowPanel1: TFlowPanel;

AddOptionButton: TButton;

SaveBtn: TBitBtn;

TabSheet2: TTabSheet;

ScrollBox1: TScrollBox;

GroupBoxMust: TGroupBox;

GroupBoxMay: TGroupBox;

ListBoxMust: TListBox;

ListBoxMay: TListBox;

BtnAddMust: TButton;

BtnDeleteMust: TButton;

BtnAddMay: TButton;

BtnDeleteMay: TButton;

procedure ComboBoxTypeChange(Sender: TObject);

procedure FormCreate(Sender: TObject);

procedure FormClose(Sender: TObject; var Action: TCloseAction);

procedure AddOptionButtonClick(Sender: TObject);

procedure DeleteInAddMode(Sender: TObject);

procedure DeleteInModifyMode(Sender: TObject);

procedure DeleteClick(Sender: TObject);

procedure SaveBtnClick(Sender: TObject);

procedure FormShow(Sender: TObject);

procedure EditOptionalChange(Sender: TObject);

procedure BtnAddMayClick(Sender: TObject);

procedure BtnAddMustClick(Sender: TObject);

procedure BtnDeleteMustClick(Sender: TObject);

procedure BtnDeleteMayClick(Sender: TObject);

private

{ Private declarations }

optionalAttributes : TList;

availableAttributeList : TStringList;

isMayMustChanged : boolean;

procedure pvLoadData();

procedure pvSetDataByType(isAttribute : boolean);

procedure pvSetAvailableAttributes(isAttribute : boolean);

function pvGetIndexOfComboAttribute(attrName : string) : integer;

procedure pvShowErrorDialog(errorNumb: integer; errorMsg: string);

public

{ Public declarations }

schema : ADSchema;

isModify : boolean;

entryName : string;

end;

var

AddForm: TAddForm;

implementation

{$R \*.dfm}

//Error handling

procedure TAddForm.pvShowErrorDialog(errorNumb: integer; errorMsg: string);

var

errorForm : TErrorForm;

begin

errorForm := TErrorForm.Create(Application);

errorForm.SetErrorInfo(errorNumb, errorMsg);

try

errorForm.ShowModal;

finally

errorForm.Free;

end;

end;

procedure TAddForm.SaveBtnClick(Sender: TObject);

var

status : ADSchemaStatus;

newEntry : ADEntry;

entryName : string;

i : integer;

temp : OptionClass;

tempiAttribute : integer;

begin

if schema = nil then

Exit;

entryName := EditName.Text;

newEntry := ADEntry.Create(entryName);

if not isModify then

// ADDING

begin

newEntry.AddAttribute('cn', [entryName]);

if ComboBoxType.ItemIndex = 0 then

begin

newEntry.AddAttribute('objectClass', ['attributeSchema']);

newEntry.AddAttribute('attributeID', [OIDEdit.Text]);

newEntry.AddAttribute('attributeSyntax', [SyntaxEdit.Text]);

newEntry.AddAttribute('oMSyntax', [oMSyntaxEdit.Text]);

end

else

begin

newEntry.AddAttribute('objectClass', ['classSchema']);

newEntry.AddAttribute('governsID', [OIDEdit.Text]);

end;

for i := 0 to optionalAttributes.Count - 1 do

begin

temp := OptionClass(optionalAttributes[i]);

//add optional attributes

newEntry.AddAttribute(temp.comboBox.Text, [temp.edit.Text]);

//TODO: if there are attributes with equal name then add them to value

end;

if isMayMustChanged then

begin

if ListBoxMay.Items.Count > 0 then

begin

tempiAttribute := newEntry.AddAttribute('mayContain');

for i := 0 to ListBoxMay.Items.Count - 1 do

newEntry.Attributes[tempiAttribute].AddValue(ListBoxMay.Items[i]);

end;

if ListBoxMust.Items.Count > 0 then

begin

tempiAttribute := newEntry.AddAttribute('mustContain');

for i := 0 to ListBoxMust.Items.Count - 1 do

newEntry.Attributes[tempiAttribute].AddValue(ListBoxMust.Items[i]);

end;

end;

status := schema.AddEntry(newEntry);

end

else

//MODIFYING don't touch not changed attributes

begin

for i := 0 to optionalAttributes.Count - 1 do

begin

temp := OptionClass(optionalAttributes[i]);

//add optional attributes

if temp.edit.Font.Color = clRed then

if temp.edit.Text <> temp.onLoadEditText then

newEntry.AddAttribute(temp.comboBox.Text, [temp.edit.Text]);

//TODO: if there are attributes with equal name then add them to value

end;

if isMayMustChanged then

begin

if ListBoxMay.Items.Count > 0 then

begin

tempiAttribute := newEntry.AddAttribute('mayContain');

for i := 0 to ListBoxMay.Items.Count - 1 do

newEntry.Attributes[tempiAttribute].AddValue(ListBoxMay.Items[i]);

end;

if ListBoxMust.Items.Count > 0 then

begin

tempiAttribute := newEntry.AddAttribute('mustContain');

for i := 0 to ListBoxMust.Items.Count - 1 do

newEntry.Attributes[tempiAttribute].AddValue(ListBoxMust.Items[i]);

end;

end;

status := schema.ModifyEntryAttributes(newEntry);

end;

if status.StatusType <> SuccessStatus then

begin

pvShowErrorDialog(status.StatusNumb, status.StatusMsg);

status.Free;

if newEntry <> nil then

newEntry.Destroy;

Exit;

end;

status.Free;

if newEntry <> nil then

newEntry.Destroy;

ModalResult := mrOk;

end;

procedure TAddForm.EditOptionalChange(Sender: TObject);

begin

(Sender as TEdit).Font.Color := clRed;

end;

procedure TAddForm.AddOptionButtonClick(Sender: TObject);

var

combobx : TComboBox;

editbx : TEdit;

deletebx : TSpeedButton;

i : integer;

begin

combobx := TComboBox.Create(self);

combobx.Parent := FlowPanel1;

if availableAttributeList <> nil then

begin

if availableAttributeList.Count > 0 then

begin

for i := 0 to availableAttributeList.Count - 1 do

combobx.Items.Add(availableAttributeList[i]);

combobx.ItemIndex := 0;

end;

end;

editbx := TEdit.Create(self);

editbx.Parent := FlowPanel1;

editbx.OnChange := EditOptionalChange;

deletebx := TSpeedButton.Create(self);

//set image

//deletebx.Glyph.LoadFromFile(GetCurrentDir + '/Icons/delete2322.bmp');

deletebx.Glyph.LoadFromFile(GetCurrentDir + '/Icons/delete2322.bmp');

//set click event handler

deletebx.OnClick := DeleteClick;

deletebx.Parent := FlowPanel1;

optionalAttributes.Add(OptionClass.Create(combobx, editbx, deletebx, 'hehe! Try to fit this :D'));

end;

procedure TAddForm.BtnAddMayClick(Sender: TObject);

var

selectForm : TSelectForm;

begin

selectForm := TSelectForm.Create(Application);

selectForm.schema := schema;

try

if selectForm.ShowModal = mrOk then

begin

ListBoxMay.Items.Add(selectForm.selectedValue);

isMayMustChanged := true;

end;

finally

selectForm.Free;

end;

end;

procedure TAddForm.BtnAddMustClick(Sender: TObject);

var

selectForm : TSelectForm;

begin

selectForm := TSelectForm.Create(Application);

selectForm.schema := schema;

try

if selectForm.ShowModal = mrOk then

begin

ListBoxMust.Items.Add(selectForm.selectedValue);

isMayMustChanged := true;

end;

finally

selectForm.Free;

end;

end;

procedure TAddForm.BtnDeleteMayClick(Sender: TObject);

begin

if ListBoxMay.ItemIndex <> -1 then

begin

ListBoxMay.Items.Delete(ListBoxMay.ItemIndex);

isMayMustChanged := true;

end;

end;

procedure TAddForm.BtnDeleteMustClick(Sender: TObject);

begin

if ListBoxMust.ItemIndex <> -1 then

begin

ListBoxMust.Items.Delete(ListBoxMust.ItemIndex);

isMayMustChanged := true;

end;

end;

procedure TAddForm.ComboBoxTypeChange(Sender: TObject);

begin

if ComboBoxType.ItemIndex = 1 then

begin

pvSetDataByType(false);

pvSetAvailableAttributes(false);

end

else

begin

pvSetDataByType(true);

pvSetAvailableAttributes(true);

end;

end;

procedure TAddForm.DeleteClick(Sender: TObject);

begin

if schema = nil then

Exit;

if isModify then

DeleteInModifyMode(Sender)

else

DeleteInAddMode(Sender);

end;

procedure TAddForm.DeleteInAddMode(Sender: TObject);

var

i : integer;

temp : OptionClass;

begin

//remove from form

for i := 0 to optionalAttributes.Count - 1 do

begin

if OptionClass(optionalAttributes[i]).deleteBtn = Sender then

begin

temp := OptionClass(optionalAttributes[i]);

temp.comboBox.Free;

temp.edit.free;

temp.deleteBtn.Free;

temp.Free;

optionalAttributes.Delete(i);

Break;

end;

end;

end;

procedure TAddForm.DeleteInModifyMode(Sender: TObject);

var

i : integer;

temp : OptionClass;

attrName, attrValue, entryName : string;

status : ADSchemaStatus;

begin

//remove from form

for i := 0 to optionalAttributes.Count - 1 do

begin

if OptionClass(optionalAttributes[i]).deleteBtn = Sender then

begin

temp := OptionClass(optionalAttributes[i]);

attrName := temp.comboBox.Text;

attrValue := temp.edit.Text;

entryName := EditName.Text;

//call api

status := schema.DeleteEntryAttributes(entryName, [attrName]);

if status.StatusNumb <> 16 then

if status.StatusType <> SuccessStatus then

begin

pvShowErrorDialog(status.StatusNumb, status.StatusMsg);

status.free;

Break;

end;

status.free;

temp.comboBox.Free;

temp.edit.free;

temp.deleteBtn.Free;

temp.Free;

optionalAttributes.Delete(i);

Break;

end;

end;

end;

procedure TAddForm.FormClose(Sender: TObject; var Action: TCloseAction);

var

i : integer;

begin

if optionalAttributes.Count > 0 then

for i := 0 to optionalAttributes.Count - 1 do

OptionClass(optionalAttributes[i]).Free;

optionalAttributes.Free;

availableAttributeList.Free;

end;

procedure TAddForm.FormCreate(Sender: TObject);

begin

availableAttributeList := TStringList.Create;

optionalAttributes := TList.Create;

end;

procedure TAddForm.FormShow(Sender: TObject);

begin

isMayMustChanged := false;

pvSetDataByType(true);

pvSetAvailableAttributes(true);

if isModify then

begin

Caption := 'Modify';

pvLoadData;

EditName.Enabled := false;

ComboBoxType.Enabled := false;

OIDEdit.Enabled := false;

SyntaxEdit.Enabled := false;

oMSyntaxEdit.Enabled := false;

BtnAddMust.Visible := false;

BtnDeleteMust.Visible := false;

end;

end;

procedure TAddForm.pvLoadData();

var

entries : ADEntryList;

entry : ADEntry;

status : ADSchemaStatus;

temp : ADAttribute;

iTemp : integer;

iAttribute : integer;

lAttributesBox : TComboBox;

lAttrValueEdit : TEdit;

lAttrDelete : TSpeedButton;

i : integer;

begin

if schema = nil then

Exit;

//Load modify Data entry

entries := schema.GetEntries('(cn=' + entryName + ')', [], status);

if status.StatusType <> SuccessStatus then

begin

pvShowErrorDialog(status.StatusNumb, status.StatusMsg);

status.free;

if entries <> nil then

entries.Destroy;

Exit;

end;

status.free;

if entries = nil then

begin

pvShowErrorDialog(999, 'Cant load the entry data');

Exit;

end;

if entries.EntriesCount > 0 then

entry := entries.Items[0]

else

begin

pvShowErrorDialog(999, 'Cant load the entry data');

entries.Destroy;

Exit;

end;

EditName.Text := entry.Name;

temp := entry.GetAttributeByName('objectClass');

if temp <> nil then

begin

if temp.ValuesCount > 0 then

begin

iTemp := temp.SearchValue('attributeSchema');

if iTemp <> -1 then

begin

ComboBoxType.ItemIndex := 0;

pvSetDataByType(true);

pvSetAvailableAttributes(true);

temp := entry.GetAttributeByName('attributeID');

if temp <> nil then

begin

if temp.ValuesCount > 0 then

OIDEdit.Text := temp.Values[0];

entry.DeleteAttribute('attributeID');

end;

temp := entry.GetAttributeByName('attributeSyntax');

if temp <> nil then

begin

if temp.ValuesCount > 0 then

SyntaxEdit.Text := temp.Values[0];

entry.DeleteAttribute('attributeSyntax');

end;

temp := entry.GetAttributeByName('oMSyntax');

if temp <> nil then

begin

if temp.ValuesCount > 0 then

oMSyntaxEdit.Text := temp.Values[0];

entry.DeleteAttribute('oMSyntax');

end;

end

else

begin

ComboBoxType.ItemIndex := 1;

pvSetDataByType(false);

pvSetAvailableAttributes(false);

temp := entry.GetAttributeByName('governsID');

if temp <> nil then

begin

if temp.ValuesCount > 0 then

OIDEdit.Text := temp.Values[0];

entry.DeleteAttribute('governsID');

end;

end;

end;

entry.DeleteAttribute('objectClass');

end;

//other attributes

for iAttribute := 0 to entry.AttributesCount - 1 do

begin

temp := entry.Attributes[iAttribute];

if ((temp.Name = 'mustContain') or (temp.Name = 'systemMustContain')) then

begin

if temp.ValuesCount > 0 then

for i := 0 to temp.ValuesCount - 1 do

ListBoxMust.Items.Add(temp.Values[i]);

end;

if ((temp.Name = 'mayContain') or (temp.Name = 'systemMayContain')) then

begin

if temp.ValuesCount > 0 then

for i := 0 to temp.ValuesCount - 1 do

ListBoxMay.Items.Add(temp.Values[i]);

end;

lAttributesBox := TComboBox.Create(self);

lAttributesBox.Parent := FlowPanel1;

if availableAttributeList <> nil then

begin

if availableAttributeList.Count > 0 then

begin

for i := 0 to availableAttributeList.Count - 1 do

lAttributesBox.Items.Add(availableAttributeList[i]);

iTemp := pvGetIndexOfComboAttribute(temp.Name);

if iTemp <> -1 then

lAttributesBox.ItemIndex := 0

else lAttributesBox.Text := temp.Name;

end;

end;

lAttrValueEdit := TEdit.Create(self);

lAttrValueEdit.Parent := FlowPanel1;

//TODO: if attribute is multiValued

if temp.ValuesCount > 0 then

lAttrValueEdit.Text := temp.Values[0];

lAttrValueEdit.OnChange := EditOptionalChange;

lAttrDelete := TSpeedButton.Create(self);

//set image

lAttrDelete.Glyph.LoadFromFile(GetCurrentDir + '/Icons/delete2322.bmp');

//set click event handler

lAttrDelete.OnClick := DeleteClick;

lAttrDelete.Parent := FlowPanel1;

optionalAttributes.Add(OptionClass.Create(lAttributesBox, lAttrValueEdit, lAttrDelete, lAttrValueEdit.Text));

end;

end;

function TAddForm.pvGetIndexOfComboAttribute(attrName : string) : integer;

var

i : integer;

begin

result := -1;

if availableAttributeList.Count > 0 then

for i := 0 to availableAttributeList.Count - 1 do

if availableAttributeList[i] = attrName then

result := i;

end;

procedure TAddForm.pvSetDataByType(isAttribute : boolean);

begin

if isAttribute then

begin

OIDLabel.Caption := 'attributeID';

SyntaxLabel.Visible := true;

SyntaxEdit.Visible := true;

oMSyntaxLabel.Visible := true;

oMSyntaxEdit.Visible := true;

TabSheet2.TabVisible := false;

end

else

begin

OIDLabel.Caption := 'governsID';

SyntaxLabel.Visible := false;

SyntaxEdit.Visible := false;

oMSyntaxLabel.Visible := false;

oMSyntaxEdit.Visible := false;

TabSheet2.TabVisible := true;

end;

end;

procedure TAddForm.pvSetAvailableAttributes(isAttribute : boolean);

var

entries : ADEntryList;

status : ADSchemaStatus;

iEntry, iAttribute, iValue : integer;

temp : string;

begin

availableAttributeList.Clear;

if schema = nil then

Exit;

//Load item list

if isAttribute then

begin

entries := schema.GetEntries('(|(cn=top)(cn=attributeSchema))',

['mustContain', 'mayContain', 'systemMustContain', 'systemMayContain'],

status);

end else

begin

entries := schema.GetEntries('(|(cn=top)(cn=classSchema))',

['mustContain', 'mayContain', 'systemMustContain', 'systemMayContain'],

status);

end;

if status.StatusType <> SuccessStatus then

begin

pvShowErrorDialog(status.StatusNumb, status.StatusMsg);

status.free;

entries.destroy;

Exit;

end;

status.free;

if entries = nil then

Exit;

//Set to available attributes except mayContain, mustContain

for iEntry := 0 to entries.EntriesCount - 1 do

begin

for iAttribute := 0 to entries.Items[iEntry].AttributesCount - 1 do

begin

for iValue := 0 to entries.Items[iEntry].Attributes[iAttribute].ValuesCount - 1 do

begin

temp := entries.Items[iEntry].Attributes[iAttribute].Values[iValue];

if ((temp <> 'mustContain')

and (temp <> 'mayContain')

and (temp <> 'systemMustContain')

and (temp <> 'systemMayContain')) then

availableAttributeList.Add(temp);

end;

end;

end;

end;

{ OptionClass }

constructor OptionClass.Create(cbx: TComboBox; ed: TEdit; dbtn: TSpeedButton; onLoadText : string);

begin

comboBox := cbx;

edit := ed;

deleteBtn := dbtn;

onLoadEditText := onLoadText;

end;

end.

**Модуль SelectPage.pas**

unit SelectPage;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, StdCtrls, Buttons, ADSchemaUnit, ADSchemaTypes;

type

TSelectForm = class(TForm)

ListBoxSelect: TListBox;

BtnOK: TBitBtn;

procedure FormShow(Sender: TObject);

procedure BtnOKClick(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

schema : ADSchema;

selectedValue : string;

procedure LoadInOtherThread();

end;

TLoadThread = class(TThread)

protected

procedure Execute; override;

end;

var

SelectForm: TSelectForm;

implementation

{$R \*.dfm}

procedure TLoadThread.Execute;

begin

SelectForm.LoadInOtherThread;

end;

procedure TSelectForm.LoadInOtherThread();

var

entries : ADEntryList;

status : ADSchemaStatus;

i : integer;

begin

if schema <> nil then

begin

entries := schema.GetAll(AttributeEntry, ['cn'], status);

if status.StatusType <> SuccessStatus then

begin

ShowMessage('Unable to load attribute list!');

status.Free;

if entries <> nil then

entries.Destroy;

Exit;

end;

status.Free;

if entries = nil then

Exit;

if entries.EntriesCount = 0 then

begin

entries.Destroy;

Exit;

end;

for i := 0 to entries.EntriesCount - 1 do

begin

ListBoxSelect.Items.Add(entries.Items[i].Name);

end;

entries.Destroy;

end;

end;

procedure TSelectForm.BtnOKClick(Sender: TObject);

begin

if ListBoxSelect.ItemIndex = -1 then

begin

ShowMessage('Please Select the attribute!');

Exit;

end;

selectedValue := ListBoxSelect.Items[ListBoxSelect.ItemIndex];

ModalResult := mrOk;

end;

procedure TSelectForm.FormShow(Sender: TObject);

var

newThread : TLoadThread;

begin

newThread := TLoadThread.Create(true);

newThread.Synchronize(LoadInOtherThread);

end;

end.

**Модуль ErrorPage.pas**

unit ErrorPage;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, StdCtrls, Buttons;

type

TErrorForm = class(TForm)

GroupBox1: TGroupBox;

ErrorMsgLabel: TLabel;

ErrorNumbLabel: TLabel;

BitBtn1: TBitBtn;

procedure BitBtn1Click(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

procedure SetErrorInfo(numb: integer; msg: string);

end;

var

ErrorForm: TErrorForm;

implementation

{$R \*.dfm}

procedure TErrorForm.BitBtn1Click(Sender: TObject);

begin

ModalResult := mrOk;

end;

procedure TErrorForm.SetErrorInfo(numb: integer; msg: string);

begin

ErrorNumbLabel.Caption := ErrorNumbLabel.Caption + IntToStr(numb);

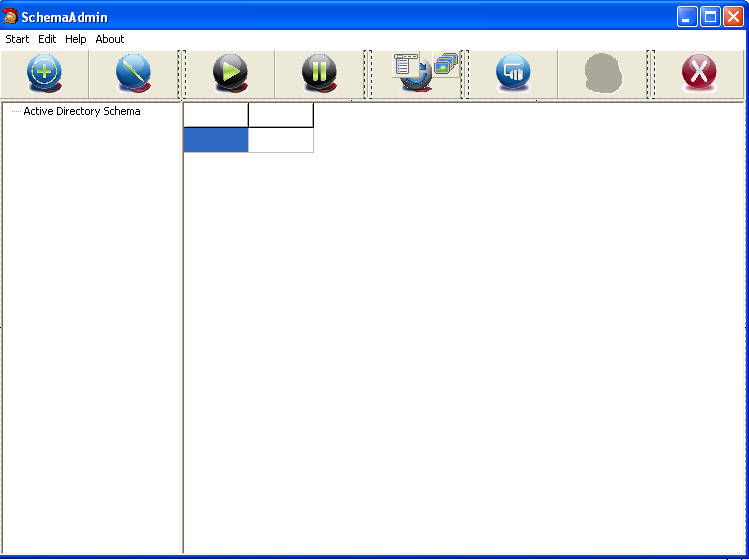
ErrorMsgLabel.Caption := msg;

end;

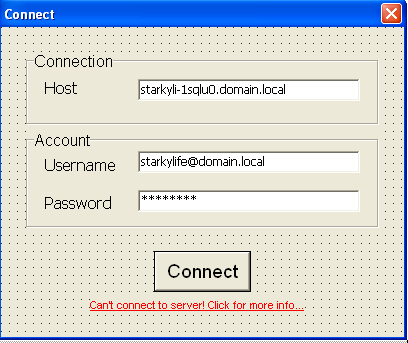
end.

# Приложение 2. Экранные формы программы

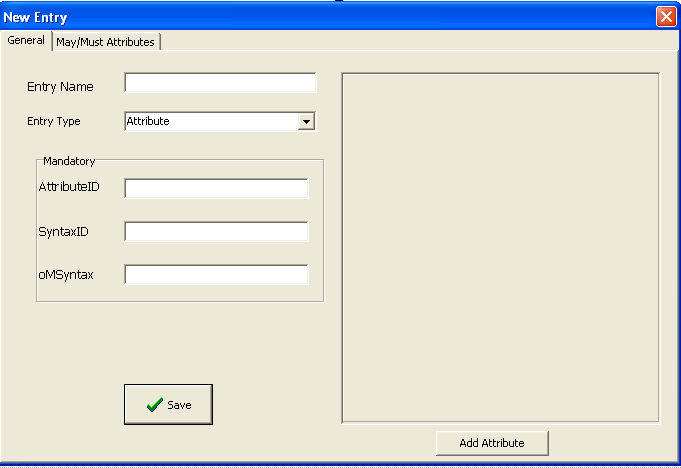
**Форма MainForm**



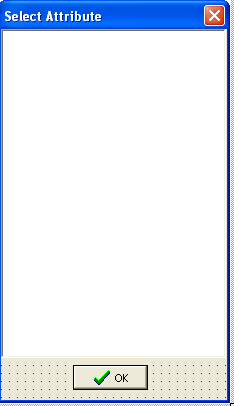
**Форма ConnectForm**

****

**Форма AddForm**

****

**Форма SelectForm**

****

**Форма ErrorForm**

****