

# 52

## Службы каталогов

### **В ЭТОЙ ГЛАВЕ...**

- Архитектура и концепции службы Active Directory
- Инструментальные средства, доступные для администрирования Active Directory
- Способы считывания и изменения данных в Active Directory
- Поиск объектов в Active Directory
- Программное управление учетными записями пользователей и групп
- Использование языка DSML для доступа к Active Directory

Предлагаемый Microsoft продукт Active Directory представляет собой *службу каталогов* (directory service), которая обеспечивает централизованное иерархическое хранилище для пользовательской информации, сетевых ресурсов, разнообразных служб и т.п. Эта служба каталогов может расширяться и предусматривать хранение также и каких-то специальных данных, необходимых для предприятия. Например, в Microsoft Exchange Server и Microsoft Dynamics служба Active Directory довольно интенсивно применяется для хранения общих папок и других элементов.

До выхода Active Directory в Exchange Server для хранения объектов использовалось собственное приватное хранилище. Системному администратору приходилось конфигурировать для каждого человека по две учетных записи: одну в домене Windows NT для предоставления возможности входа в систему и одну — непосредственно в Exchange Directory. Это было необходимо из-за того, что пользователям требовалась дополнительная информация (вроде адреса электронной почты, телефонных номеров и т.д.), а в домен Windows NT такую информацию добавлять было нельзя.

Теперь системному администратору нужно конфигурировать всего лишь одну учетную запись для каждого пользователя в Active Directory, а далее информация для соответствующего объекта user уже может автоматически расширяться в соответствии с требованиями Exchange Server. Эта информация также может расширяться и вручную самим администратором, например, путем добавления в Active Directory списка навыков, что позволит очень легко отыскивать в списке пользователей только тех разработчиков, которые умеют программировать на языке C#, просто выполняя поиск навыка работы на C#.

В настоящей главе демонстрируется, как .NET Framework можно использовать для доступа и манипулирования данными в службе каталогов с помощью классов из таких пространств имен, как System.DirectoryServices, System.DirectoryServices.AccountManagement и System.DirectoryServices.Protocols.



*В этой главе предполагается использование ОС Windows Server 2008 R2 со сконфигурированной службой каталогов Active Directory. Однако это также вполне может быть и Windows Server 2003 с другими службами каталогов.*

После описания архитектуры и приемов программирования в Active Directory далее в этой главе для примера демонстрируется создание простого приложения Windows, позволяющего указывать свойства и фильтр для выполнения поиска объектов user. Как и для других глав, коды всех рассмотренных здесь примеров доступны на прилагаемом компакт-диске.

## Архитектура Active Directory

Прежде чем приступить к программированию с использованием Active Directory, вы должны знать, как эта служба работает, для каких целей предназначена, и какие данные можно в ней хранить.

### Возможности Active Directory

Возможности Active Directory можно кратко описать следующим образом.

- **Иерархическое группирование данных.** Объекты могут храниться в объектах-контейнерах. Вместо того, чтобы поддерживать единственный крупный список объектов, пользователи могут быть сгруппированы в организационные единицы. Конкретная организационная единица может содержать другие организационные единицы, что, в конечном итоге, образует дерево.

- **Репликация с множеством ведущих узлов (multimaster replication).** В Active Directory каждый контроллер домена (domain controller – DC) является ведущим узлом. Благодаря множеству ведущих узлов, изменения могут быть внесены в любой DC. Такая модель является намного более расширяемой, нежели модель с одним ведущим узлом, поскольку изменения можно вносить в различные серверы параллельно. Недостаток такой модели заключается в том, что репликация в этом случае оказывается намного сложнее. Проблемы репликации обсуждаются ниже в этой главе.
- **Гибкая топология репликации.** Это означает поддержку репликации через медленнодействующие каналы передачи данных в глобальных сетях. Вопрос о том, насколько часто данные должны подвергаться репликации, решается администраторами домена во время конфигурирования.
- **Открытые стандарты.** Служба Active Directory поддерживает *открытые стандарты*. Протокол *LDAP* (Lightweight Directory Access Protocol – облегченный протокол службы каталогов) представляет собой один из таких открытых стандартов Интернета, который может применяться для получения доступа к множеству различных служб каталогов, в том числе и к данным Active Directory. Помимо LDAP также еще имеется и отдельный основанный на нем интерфейс LDAP API. Этот интерфейс может применяться для получения доступа к Active Directory с помощью языка C. Другим стандартом, который часто используется в Active Directory, является *Kerberos*, предназначенный для обеспечения аутентификации. Встроенная в Windows Server служба Kerberos может тоже применяться для осуществления аутентификации клиентов UNIX.
- **Интерфейс служб Active Directory (Active Directory Service Interface – ADSI).** ADSI позволяет определять интерфейсы COM для получения доступа к службам каталогов. ADSI делает возможным доступ ко всем функциональным возможностям Active Directory. Классы из пространства имен System.DirectoryServices создают оболочку для объектов ADSI COM и тем самым позволяют делать службы каталогов доступными из приложений .NET.
- **Язык DSML (Directory Service Markup Language – язык разметки для служб каталогов).** Язык DSML представляет собой еще один стандарт для получения доступа к службам каталогов. Он не предусматривает зависимости от платформы и поддерживается группой OASIS.
- **Мелкоструктурная система безопасности.** В рамках Active Directory доступна мелкоструктурная система безопасности. Каждый объект, сохраняемый в Active Directory, может иметь связанный с ним список контроля доступа, который определяет, что можно делать с этим объектом.

Объекты в каталоге подвергаются *строгому контролю типов*, что означает точное определение типа объекта; к объекту не могут быть добавлены никакие не описанные атрибуты. В *схеме* определяются типы объектов, а также части объектов (атрибуты). Атрибуты могут быть обязательными или необязательными.

## Концепции Active Directory

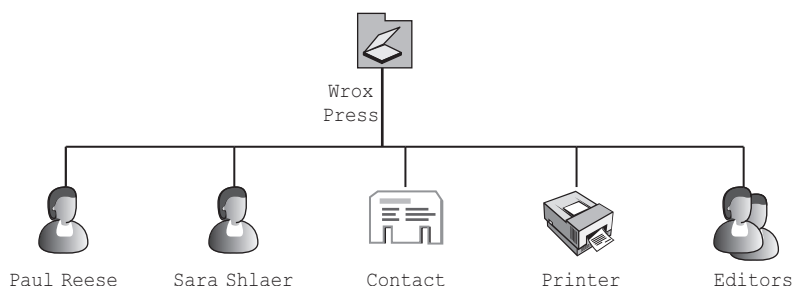
Прежде чем программировать с использованием Active Directory, необходимо ознакомиться с некоторыми терминами и определениями.

### Объекты

В Active Directory хранятся объекты. Любой объект ссылается на нечто конкретное, например, на пользователя, принтер или разделяемый сетевой ресурс. Объекты имеют

обязательные и необязательные атрибуты, которые служат его описанием. Примерами атрибутов объекта user (пользователь) могут служить фамилия и имя пользователя, адрес электронной почты, номер телефона и т.п.

На рис. 52.1 показан объект-контейнер по имени Wrox Press, который содержит другие объекты — два объекта пользователей Paul Reese и Sara Shlaer, объект контактов Contact, объект принтера Printer и объект группы пользователей Editors.



**Рис. 52.1.** Объект-контейнер Wrox Press

## Схема

Каждый объект представляет собой экземпляр какого-то класса, определенного в *схеме*. Схема *определяет типы* и сама хранится среди объектов Active Directory. Нужно четко понимать разницу между схемой classSchema и схемой attributeSchema. В схеме classSchema определяются типы объектов и детализируется, какими обязательными и необязательными атрибутами обладает каждый объект. В схеме attributeSchema определяется внешний вид атрибутов и допустимый для каждого из них синтаксис.

Можно определить пользовательские типы и атрибуты, а затем включить их в схему. Однако не упускайте из виду тот факт, что новый тип схемы нельзя удалить из Active Directory. Можно пометить такую схему как неактивную, и после этого создавать новые объекты этого типа будет нельзя, тем не менее, объекты данного типа могут уже существовать, следовательно, удалять классы или атрибуты, определенные в схеме, невозможно.

Администратор группы пользователей не имеет достаточно прав, чтобы создавать новые элементы схемы, это разрешено только членам группы администраторов уровня предприятия.

## Конфигурация

В дополнение к объектам и определениям классов, которые сохраняются как объекты, конфигурация самой службы Active Directory также хранится в Active Directory. Конфигурация службы Active Directory содержит информацию обо всех сайтах, например, интервал репликации, устанавливаемый системным администратором. Поскольку сама конфигурация хранится в Active Directory, к ней можно получать доступ как ко всем другим объектам Active Directory.

## Домен Active Directory

Домен представляет собой границы безопасности сети Windows. В домене Active Directory объекты хранятся в иерархическом порядке. Сама служба Active Directory состоит из одного или нескольких доменов. На рис. 52.2 показан иерархический порядок объектов, содержащихся в домене, при этом сам домен изображен в виде треугольника. Объекты-контейнеры, такие как Users, Computers и Books, могут хранить в себе другие объекты.

Каждый овал на рисунке представляет объект, а линии между объектами представляют отношения “родительский–дочерний”. Например, объект Books является родителем по отношению к .NET и Java, а ProC#, BegC# и ASP.NET — это дочерние объекты объекта .NET.

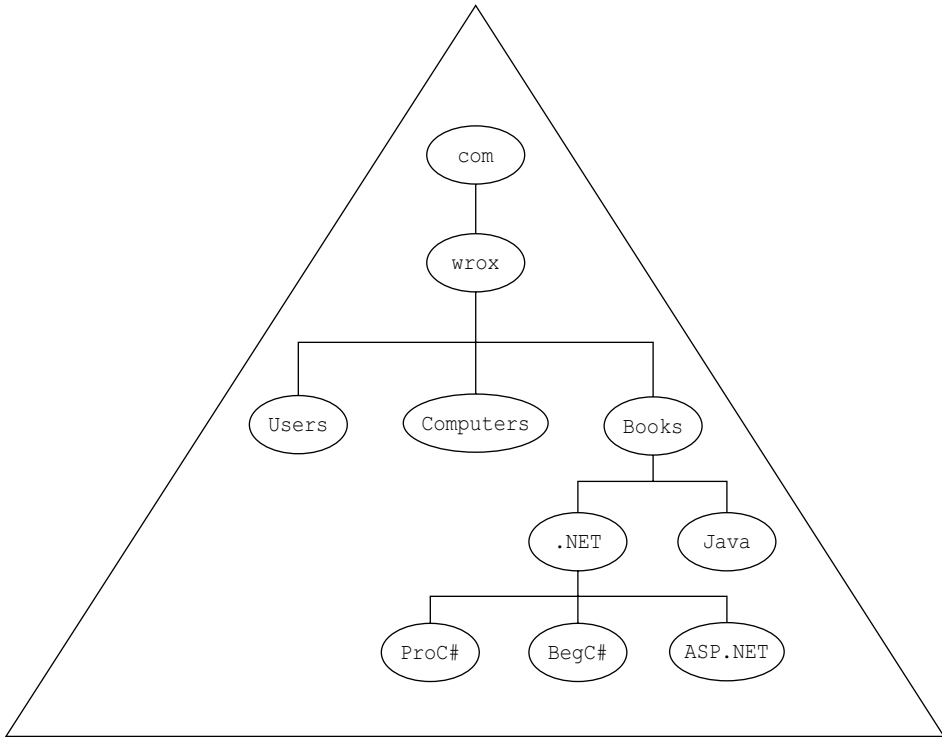


Рис. 52.2. Иерархия объектов в домене

### Контроллер домена

Один домен может иметь несколько контроллеров домена, каждый из которых хранит все объекты домена. Ведущего сервера нет, и все контроллеры домена рассматриваются как эквивалентные; в результате получается модель с множеством ведущих узлов. Объекты реплицируются по серверам, содержащимся в домене.

### Сайт

Сайт (site) представляет собой участок в сети, содержащий, по крайней мере, один контроллер домена. Если на вашем предприятии имеется несколько участков, соединенных между собой медленными линиями связи, вы можете использовать множество сайтов в одном домене. Из соображений резервирования или расширяемости каждый сайт может иметь один или несколько функционирующих контроллеров домена. Репликация между серверами в сайте может выполняться через укороченные интервалы, что связано с быстродействующим сетевым соединением.

Репликация между серверами внутри сайта может быть настроена на более длительные интервалы, что зависит от быстродействия сети. Естественно, интервалы репликации устанавливаются администратором домена.

## **Дерево доменов**

Множество доменов могут связываться друг с другом доверительными отношениями. Такие домены совместно используют *общую схему, общую конфигурацию и глобальный каталог* (глобальный каталог подробно рассматривается позже в главе). Наличие общей схемы и общей конфигурации говорит о том, что эти данные реплицируются между доменами. Деревья доменов совместно используют один и тот же класс и ту же самую схему атрибутов. Репликация самих объектов между доменами не производится.

Домены, связанные описанным способом, образуют дерево доменов. Домены в дереве доменов имеют *непрерывное иерархическое пространство имен*. Это значит, что имя дочернего домена — это имя, под которым дочерний домен добавлялся к родительскому домену. Между доменами устанавливаются доверительные отношения с использованием протокола Kerberos.

Например, пусть имеется корневой домен `wrox.com`, который является *родительским доменом* для *дочерних доменов* `india.wrox.com` и `uk.wrox.com`. Между родительскими и дочерними доменами устанавливаются доверительные отношения, следовательно, учетные записи из одного домена могут аутентифицироваться в другом домене.

## **Лес**

Множество деревьев домена, которые соединены с использованием общей схемы, общей конфигурации и глобального каталога при отсутствии непрерывного пространства имен называется *лесом* (forest). Лес — это множество деревьев домена, и он может применяться в случае, когда компания имеет дочернюю компанию, для которой должно использоваться другое доменное имя. Например, домен `wrox.com` должен быть относительно независимым от домена `wiley.com`, в то же время должна быть обеспечена возможность иметь общее управление, а пользователям домена `wrox.com` должна быть предоставлена возможность доступа к ресурсам домена `wiley.com` и наоборот. При наличии леса можно устанавливать доверительные отношения между множеством деревьев доменов.

## **Глобальный каталог**

Поиск объекта может распространяться на множество доменов. Если вы ищете конкретный объект `user` с определенными атрибутами, то должны просмотреть каждый домен. Например, начиная с домена `wrox.com`, поиск переходит к доменам `uk.wrox.com` и `india.wrox.com`; при наличии медленно действующих каналов связи поиск займет достаточно продолжительное время.

Чтобы ускорить поиск, все объекты копируются в *глобальный каталог* (global catalog — GC). Глобальный каталог реплицируется в каждый домен леса. В каждом домене имеется, по меньшей мере, один сервер, хранящий глобальный каталог. С целью повышения производительности и расширяемости в одном домене можно организовать больше одного сервера глобального каталога. При использовании глобального каталога поиск по всем объектам может выполняться на одном сервере.

Глобальный каталог представляет собой *кэш только для чтения* всех объектов, которые могут использоваться только при поиске, в то время как контроллеры доменов должны применяться для внесения изменений.

В глобальном каталоге сохраняются не все атрибуты объекта. Вы сами можете определить, должен ли атрибут сохраняться вместе с объектом. Решение относительно сохранения атрибута в GC зависит от того, как этот атрибут используется. Если атрибут часто используется при поиске, то запоминание его в глобальном каталоге ускоряет поиск. Помещение фотографии пользователя в GC не принесет какой-либо пользы, поскольку вряд ли вы когда-либо станете искать фотографию. В то же время телефонный номер может оказаться полезной добавкой, достойной запоминания в хранилище. Вы можете также включить индексирование для атрибута, которое еще больше ускорит поиск.

## Репликация

Маловероятно, чтобы вы, как программист, занимались конфигурированием репликации, но поскольку это касается данных, которые хранятся в Active Directory, вы все же должны знать, как эта репликация работает. Служба Active Directory использует архитектуру сервера с множеством ведущих узлов. Изменения вносятся в каждый контроллер домена. *Время задержки репликации* (replication latency) определяет время, по истечении которого изменения вступают в действие.

- Конфигурируемые уведомления об изменениях по умолчанию выполняются внутри сайта каждые 5 минут в случае изменения определенных атрибутов. Контроллер домена, в котором происходит изменение, извещает серверы друг за другом с 30-секундным интервалом, так что четвертый контроллер домена может получить уведомление об изменениях через 7 минут. По умолчанию уведомление об изменениях на сайтах устанавливается равным 180 минут. Межсайтовую и внутрисайтовую репликацию можно сконфигурировать с другими значениями.
- Если изменений не было, на сайте каждые 60 минут выполняется *репликация по расписанию*. Это делается для того, чтобы показать, что уведомления об изменениях не пропущены.
- Для информации, требующей повышенных мер безопасности, например, сведений о блокировке учетной записи, может быть выдано *немедленное уведомление*.

При репликации в контроллеры домена копируются только изменения. Для каждого изменения атрибута сохраняется номер версии (USN — Update Sequence Number (порядковый номер обновления)) и временная метка. Такой порядок способствует разрешению конфликтов, когда изменениям подвергается один и тот же атрибут со стороны различных серверов.

Рассмотрим пример. Атрибут номера мобильного телефона Джона Доу (John Doe) имеет USN, равный 47. Это значение уже реплицировано на все контроллеры домена. Один из системных администраторов меняет номер мобильного телефона. Изменение происходит на сервере DC1; новым значением USN для этого атрибута на сервере DC1 становится 48, в то время как на других контроллерах домена USN все еще имеет значение 47. Абонент, который продолжает считывать этот атрибут, получает старое значение до тех пор, пока не произойдет репликация на все контроллеры доменов.

Возможна, хотя и достаточно редко, ситуация, когда другой администратор изменяет атрибут телефонного номера, и при этом выбирается другой контроллер домена, поскольку этот администратор получил более быстрый ответ с сервера. Значение USN для этого атрибута на сервере DC2 также меняется на 48.

Уведомление об изменении происходит во время интервалов уведомления, поскольку изменился USN атрибута, а последняя репликация произошла, когда USN имел значение 47. Механизм репликации теперь обнаруживает, что оба сервера DC1 и DC2 имеют 48 в качестве значения USN для атрибута телефонного номера. Какой сервер выйдет победителем в этом конфликте, по сути, неважно, но один из этих серверов обязательно станет победителем. Чтобы разрешить этот конфликт, используются временные метки. Поскольку изменение на DC2 произошло позднее, реплицируется значение, хранящееся в контроллере домена DC2.



*При чтении объектов следует помнить о том, что данные не обязательно будут текущими. Срок действия данных зависит от времени задержки репликации. При внесении изменений в объект другой пользователь все еще может читать некоторые старые значения после обновления. Возможно также, что различные изменения могут произойти в один и тот же момент времени.*

## Характеристика данных Active Directory

Служба Active Directory не заменяет реляционную базу данных или реестр, потому возникает вопрос, какие данные в ней хранятся?

- В Active Directory содержатся *иерархические данные*. Можно иметь контейнеры, в которых хранятся другие контейнеры и объекты. Контейнеры сами по себе являются объектами.
- Данные должны использоваться *преимущественно для чтения*. Поскольку репликация выполняется в конкретные временные интервалы, вы не можете быть уверены, что читаете последнюю версию данных. Вы должны сознавать, что в приложениях информация, которую читаете, может оказаться не самой свежей.
- Данные должны представлять для предприятия *глобальный интерес*, так как добавление новых данных в схему вызывает их репликацию на все серверы предприятия. Для типов данных, представляющих интерес только для небольшой группы пользователей, администратор домена предприятия, как правило, не станет устанавливать новые типы схем.
- Сохраняемые данные должны иметь *обоснованный объем* из-за проблем, связанных с репликацией. Данные объемом порядка 100 Кбайт целесообразно хранить в каталоге, если изменения в эти данные вносятся один раз в неделю. В то же время, если данные меняются каждый час, то объем для таких данных выбран слишком большим. Всегда думайте о репликации данных на другие серверы — куда данные передаются, и с какими интервалами. При наличии данных большого объема в Active Directory можно поместить связь, а сами данные хранить в другом месте.

Подводя итог, скажем, что если вы храните данные в Active Directory, то они должны иметь иерархическую организацию, обоснованные объемы и быть значимыми для всего предприятия.

## Указание схемы

Объекты Active Directory поддерживают строгий контроль типов. Схема определяет типы объектов, обязательные и необязательные атрибуты, их синтаксис, а также ограничения, накладываемые на эти атрибуты. Как уже упоминалось ранее, в схемах очень важно отличать объекты со схемой класса и объекты со схемой атрибутов.

Класс представляет собой коллекцию атрибутов. В случае классов поддерживается одностороннее наследование. Как легко видеть на рис. 52.3, класс `user` является производным от класса `organizationalPerson`, `organizationalPerson` — подклассом `person`, а базовым классом является `top`. Схема класса `classSchema`, которая определяет класс, описывает атрибуты с помощью атрибута `systemMayContain`.

На рис. 52.3 показано лишь несколько из значений атрибута `systemMayContain`. С помощью утилиты ADSI Edit можно очень легко просмотреть все эти значения; о ней более подробно будет рассказываться в следующем разделе. В корневом классе `top` видно, что каждый объект может иметь общее имя (`common name` — `cn`) и атрибуты `displayName`, `objectGUID`, `whenChanged` и `whenCreated`. Класс `person` является производным от класса `top` и имеет атрибут `userPassword`. Класс `telephoneNumber.organizationalPerson` является производным от класса `person` и помимо его атрибутов дополнительно имеет еще и такие атрибуты, как `manager`, `department` и `company`. Что касается класса `user`, то у него имеются дополнительные атрибуты, необходимые для осуществления входа в систему.



# Инструменты администрирования Active Directory

Изучение некоторых инструментальных средств для администрирования Active Directory поможет понять принцип действия службы, какие данные в ней хранятся, и что можно сделать программно.

Системному администратору доступно множество инструментальных средств для ввода новых данных, обновления существующих данных и конфигурирования Active Directory.

- Оснастка Active Directory Users and Computers (Пользователи и компьютеры Active Directory) консоли управления MMC (Microsoft Management Console) служит для ввода и обновления данных пользователей.
- Оснастка Active Directory Sites and Services (Сайты и службы Active Directory) консоли управления MMC применяется для конфигурирования сайтов в домене и для репликации между этими сайтами.
- Оснастка Active Directory Domains and Trusts (Домены и доверительные отношения Active Directory) консоли управления MMC используется для установления доверительных отношений между доменами в дереве.
- Инструмент ADSI Edit представляет собой редактор Active Directory, в котором можно просматривать и редактировать каждый объект.



*Чтобы запускать эти инструменты в Windows 7, необходимо установить пакет Windows 7 Remote Administrative Tools (Инструменты удаленного администрирования для Windows 7). Для других версий Windows существуют отдельные загрузки этих инструментов администрирования системы.*

В следующих разделах более подробно рассказывается о функциональных возможностях только таких инструментальных средств, как оснастка Active Directory Users and Computers (Пользователи и компьютеры Active Directory) и утилита ADSI Edit, потому что именно они являются наиболее важными для создания приложений с использованием Active Directory.

## Оснастка Active Directory Users and Computers

Оснастка Active Directory Users and Computers консоли MMC представляет собой инструментальное средство, которое системные администраторы используют для управления пользователями.

Для запуска этой оснастки (рис. 52.4) выберите пункт меню Start⇒Programs⇒Administrative Tools⇒Active Directory Users and Computers (Пуск⇒Программы⇒Администрирование⇒Пользователи и компьютеры Active Directory).

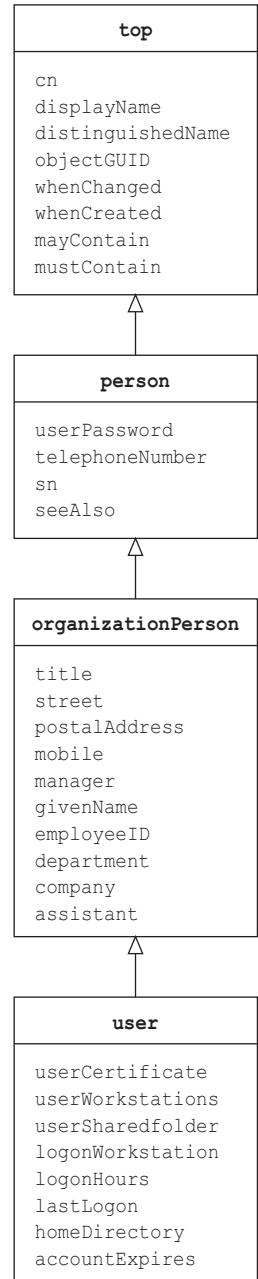
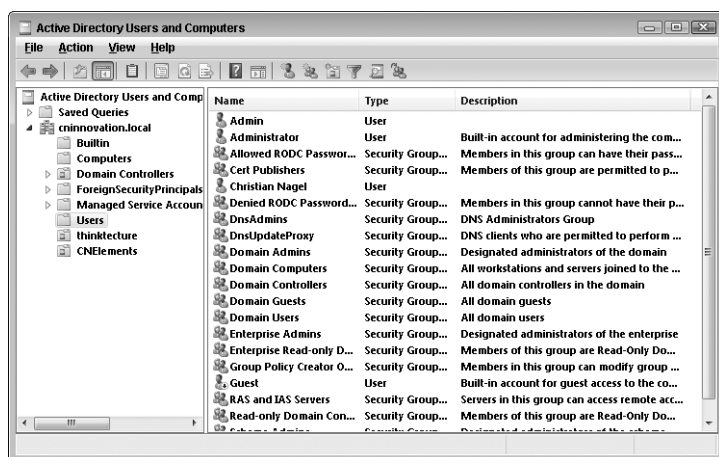


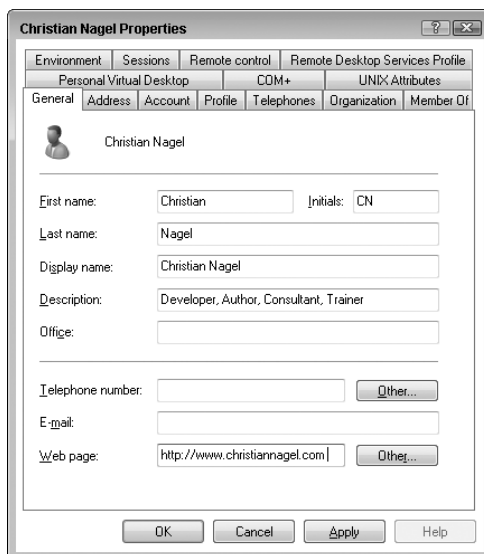
Рис. 52.3. Пример иерархии классов



**Рис. 52.4.** Оснастка Active Directory Users and Computers консоли MMC

С помощью этого инструмента можно добавлять новых пользователей, группы, контакты, организационные единицы, папки совместного использования или компьютеры и модифицировать существующие объекты упомянутого типа. На рис. 52.5 показаны атрибуты, которые могут быть заданы для объекта user: офис, телефонные номера, адреса электронной почты, веб-страницы, информация об организации, адреса, группы и прочие объекты.

Оснастка Active Directory Users and Computers также может применяться для управления крупными предприятиями с миллионами объектов. Отпадает необходимость просмотра списков, содержащих тысячи объектов, поскольку можно установить специальный фильтр для отображения только некоторых объектов. Для поиска соответствующих объектов на предприятии также можно выдавать запросы LDAP (Lightweight Directory Access Protocol – облегченный протокол службы каталогов). Эти возможности рассматриваются далее в главе.



**Рис. 52.5.** Окно свойств для выбранного объекта

## Редактор ADSI Edit

ADSI Edit представляет собой редактор Active Directory. Редактор ADSI Edit предлагает более развитые средства управления, чем оснастка Active Directory Users and Computers, как показано на рис. 52.6; с помощью редактора ADSI Edit можно конфигурировать абсолютно все, при этом можно также просматривать схему и конфигурацию. Этот инструмент не является интуитивно понятным, потому очень легко ввести неправильные данные.

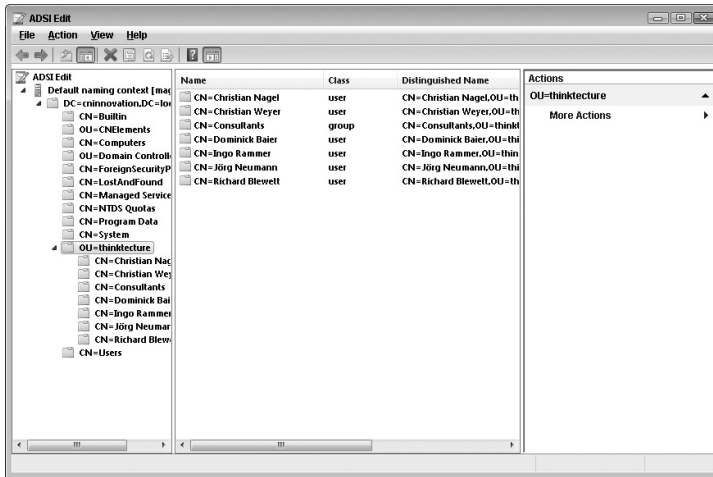


Рис. 52.6. Редактор ADSI Edit

Открыв диалоговое окно Properties (Свойства) объекта, можно просматривать и изменять каждый атрибут объекта в Active Directory. В этом окне отображаются обязательные и необязательные атрибуты со всеми их типами и значениями (рис. 52.7).

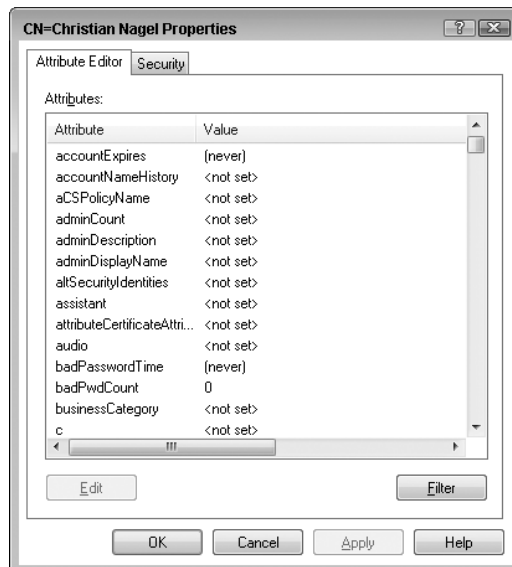


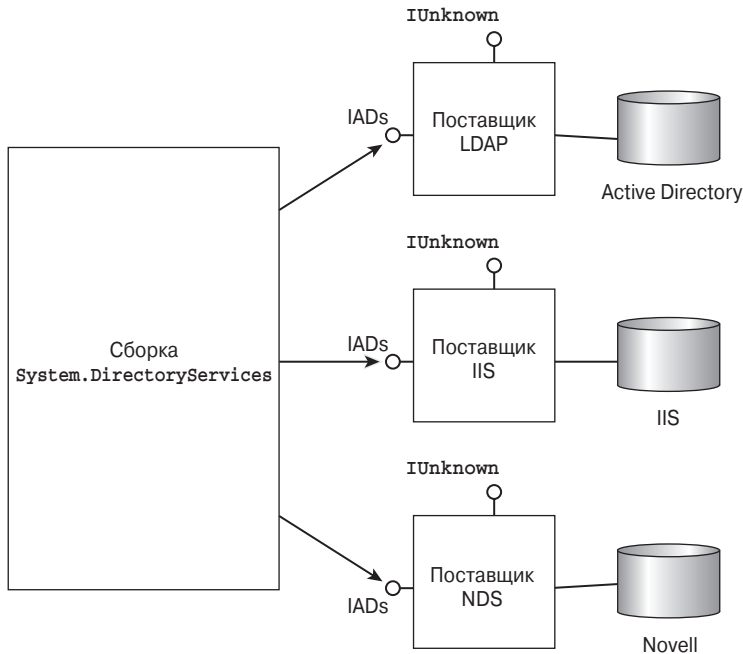
Рис. 52.7. Диалоговое окно Properties

## Программирование для Active Directory

При разработке программ для Active Directory можно пользоваться либо классами пространства имен `System.DirectoryServices`, либо классами пространства имен `System.DirectoryServices.Protocols`. В пространстве имен `System.DirectoryServices` находятся классы, упаковывающие COM-объекты *Active Directory Service Interface* (ADSI) в оболочки для получения доступа к Active Directory.

ADSI представляет собой программный интерфейс для служб каталогов. Он определяет интерфейсы COM, которые реализуются поставщиками ADSI. Это означает, что клиент может использовать разные службы каталогов с помощью одних и тех программных интерфейсов. Классы .NET Framework из пространства имен `System.DirectoryServices` основаны на применении ADSI.

На рис. 52.8 показаны некоторые поставщики ADSI (а именно — LDAP, IIS и NDS), реализующие такие интерфейсы COM, как `IADs` и `IUnknown`, а также применяющая их сборка `System.DirectoryServices`.



**Рис. 52.8.** Поставщики ADSI

Классы из пространства имен `System.DirectoryServices.Protocols` основаны на применении предназначенного для Windows языка DSML (Directory Services Markup Language — язык разметки для служб каталогов). С помощью этого языка группа OASIS ([www.oasis-open.org/committees/dsml](http://www.oasis-open.org/committees/dsml)) создает различные стандартизованные интерфейсы веб-служб.

Чтобы использовать классы из пространства имен `System.DirectoryServices`, необходимо сослаться на сборку `System.DirectoryServices`. С помощью классов этой сборки можно запрашивать объекты, просматривать и обновлять свойства, выполнять поиск объектов и перемещать объекты в другие объекты-контейнеры. Все функциональные возможности этих классов будут демонстрироваться позже в этом разделе на примере простого консольного приложения C#.

В частности, в этом разделе будут рассматриваться следующие вопросы:

- классы из пространства имен `System.DirectoryServices`;
- процесс установления соединения с Active Directory (привязка);
- извлечение записей каталога, создание новых объектов, а также обновление существующих записей;
- выполнение поиска в Active Directory.

## Классы пространства имен `System.DirectoryServices`

В табл. 52.1 перечислены основные классы пространства имен `System.DirectoryServices`.

**Таблица 52.1. Основные классы пространства имен `System.DirectoryServices`**

Класс	Описание
<code>DirectoryEntry</code>	Этот класс является основным в пространстве имен <code>System.DirectoryServices</code> . Объект этого класса представляет объект в хранилище Active Directory. Этот класс используется для привязки объекта, а также для просмотра и обновления свойств. Свойства объекта представляются в виде коллекции <code>PropertyCollection</code> . Каждый элемент в <code>PropertyCollection</code> обладает собственной коллекцией <code>PropertyValueCollection</code> .
<code>DirectoryEntries</code>	<code>DirectoryEntries</code> представляет собой коллекцию объектов <code>DirectoryEntry</code> . Свойство <code>Children</code> объекта <code>DirectoryEntry</code> возвращает список объектов в коллекции <code>DirectoryEntries</code> .
<code>DirectorySearcher</code>	Этот класс является главным классом, который применяется для выполнения поиска объектов с определенными атрибутами. Для определения параметров поиска может использоваться класс <code>SortOption</code> и перечисления <code>SearchScope</code> , <code>SortDirection</code> и <code>ReferralChasingOption</code> . Результаты поиска возвращаются в виде объекта <code>SearchResult</code> или <code>SearchResultCollection</code> , но могут еще также возвращаться и в виде объекта <code>ResultPropertyCollection</code> или <code>ResultPropertyValueCollection</code> .

## Привязка к службам каталогов

Для извлечения значений того или иного объекта из Active Directory требуется установка соединения со службой Active Directory. Процесс установления такого соединения называется *привязкой* (binding). Путь привязки может выглядеть следующим образом:

```
LDAP://dc01.thinktecture.com/OU=Development, DC=thinktecture, DC=Com
```

В этом пути могут указываться такие элементы:

- *протокол*, задающий используемого поставщика;
- *имя сервера* контроллера домена;
- *номер порта* серверного процесса;
- *отличительное имя*, идентифицирующее объект, к которому требуется получить доступ;

- *имя пользователя и пароль*, если пользователь, которому разрешено получать доступ в Active Directory, отличается от текущего зарегистрированного пользователя;
- *тип аутентификации*, если требуется шифрование.

В следующих подразделах все эти опции рассматриваются более подробно.

## Протокол

В первой части пути привязки указывается поставщик ADSI. Поставщики ADSI реализуются как COM-серверы и отображаются в системном реестре в виде ключа `progID` в узле `HKEY_CLASSES_ROOT`. Поставщики, доступные в Windows 7, перечислены в табл. 52.2.

**Таблица 52.2. Поставщики ADSI, доступные в Windows Vista**

Поставщик	Описание
LDAP	Сервер LDAP, подобный каталогу Exchange и серверам Active Directory, начиная с Windows 2000.
GC	GC используется для доступа к глобальному каталогу в Active Directory. Он может применяться для быстрых запросов.
IIS	С помощью поставщика ADSI для IIS можно создавать новые веб-сайты и администрировать их в каталоге IIS.
NDS	Этот программный идентификатор ( <code>progID</code> ) используется для обмена данными со службами каталогов Novell Directory Services.

## Имя сервера

*Имя сервера* в пути привязки следует сразу за протоколом. Имя сервера является необязательным параметром, если вы зарегистрированы в домене Active Directory. При отсутствии имени сервера выполняется *привязка без сервера* (`serverless binding`); это означает, что Windows Server 2008 пытается получить “наилучший” контроллер домена, ассоциированный с пользователем, который осуществляет привязку. Если внутри сайта нет сервера, будет использован первый найденный контроллер домена.

Привязка без сервера может иметь следующий вид:

```
LDAP://OU=Sales, DC=Thinktecture, DC=Local
```

## Номер порта

После имени сервера можно указать *номер порта* серверного процесса с использованием синтаксиса `:xxx`. Номер порта по умолчанию для сервера LDAP равен 389: `LDAP://dc01.sentinel.net:389`. Сервер обмена данными (Exchange Server) использует тот же номер порта, что и сервер LDAP. Если Exchange Server установлен в той же самой системе, например, как контроллер домена Active Directory, можно сконфигурировать другой порт

## Отличительное имя

Четвертый фрагмент пути, который понадобится определить — это *отличительное имя* (`distinguished name` — DN). Отличительное имя представляет собой уникальное имя, идентифицирующее объект, к которому необходимо получить доступ. При работе с Active Directory для определения имени объекта можно использовать синтаксис LDAP, в основе которого лежит протокол X.500.

Ниже показан пример отличительного имени:

```
CN=Christian Nagel, OU=Consultants, DC=thinktecture, DC=local
```

Это отличительное имя определяет общее имя (common name — CN) Christian Nagel в организационной единице (organizational unit — OU) под названием Consultants в контроллере домена (domain controller — DC) с именем thinktecture домена thinktecture.local. В правой части выражения указан корневой объект домена. Это имя должно соответствовать иерархии дерева объектов.

Спецификацию протокола LDAP, определяющую строковое представление отличительного имени, можно посмотреть в документе RFC 2253 по адресу [www.ietf.org/rfc/rfc2253.txt](http://www.ietf.org/rfc/rfc2253.txt).

### Относительное отличительное имя

*Относительное отличительное имя* (relative distinguished name — RDN) используется для ссылки на объекты, содержащиеся в объекте-контейнере. При наличии RDN-имени спецификации OU и DC не требуются, поскольку для этой цели общего имени достаточно. CN=Christian Nagel — это относительное отличительное имя внутри организационной единицы. Относительное отличительное имя может применяться, если уже имеется ссылка на объект-контейнер и необходимо получить доступ к дочерним объектам.

### Контекст именования по умолчанию

Если отличительное имя в пути не указано, процесс привязки будет осуществляться в соответствии с установленным для использования по умолчанию контекстом именования. Считывать информацию об установленном для использования по умолчанию контексте именования можно с помощью rootDSE. В LDAP 3.0 объект rootDSE определен как корень дерева каталогов на сервере каталогов. Например:

```
LDAP://rootDSE
```


или

```
LDAP://servername/rootDSE
```

За счет перечисления всех свойств rootDSE можно получить информацию о контексте defaultNamingContext, который будет применяться в случаях отсутствия в пути указанного отличительного имени. schemaNamingContext и configurationNamingContext указывают на имена, которые должны использоваться для получения доступа к схеме и конфигурации в хранилище Active Directory.

Для извлечения всех свойств rootDSE применяется следующий код:

```

 try
{
    using (var de = new DirectoryEntry())
    {
        de.Path = "LDAP://magellan/rootDSE";
        de.Username = @"cn\innovation\christian";
        de.Password = "Pa$$w0rd";
        PropertyCollection props = de.Properties;
        foreach (string prop in props.PropertyNames)
        {
            PropertyValueCollection values = props [prop];
            foreach (string val in values)
            {
                Console.WriteLine("{0}: ", prop);
                Console.WriteLine(val);
            }
        }
    }
}

```

```
catch (COMException ex)
{
    Console.WriteLine(ex.Message);
}
```

Фрагмент кода *DirectoryServicesSamples\Program.cs*



*Для запуска этого кода на вашей машине потребуется изменить путь к объекту доступа, включая имя сервера.*

Выполнение этого кода приведет к отображению информации об установленном для использования по умолчанию контексте именования (defaultNamingContext DC=cninnovation, DC=local), о контексте, который может применяться для получения доступа к схеме (CN=Schema, CN=Configuration, DC=cninnovation, DC=local), и о контексте именования конфигурации (CN=Configuration, DC=cninnovation, DC=local), как показано ниже:

```
currentTime: 20090925131508.0Z
subschemaSubentry: CN=Aggregate,CN=Schema,CN=Configuration,DC=cninnovation,
DC=local
dsServiceName: CN=NTDS Settings,CN=MAGELLAN,CN=Servers,
CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=cninnovation,DC=local
namingContexts: DC=cninnovation,DC=local
namingContexts: CN=Configuration,DC=cninnovation,DC=local
namingContexts: CN=Schema,CN=Configuration,DC=cninnovation,DC=local
namingContexts: DC=DomainDnsZones,DC=cninnovation,DC=local
namingContexts: DC=ForestDnsZones,DC=cninnovation,DC=local
defaultNamingContext: DC=cninnovation,DC=local
schemaNamingContext: CN=Schema,CN=Configuration,DC=cninnovation,DC=local
configurationNamingContext: CN=Configuration,DC=cninnovation,DC=local
rootDomainNamingContext: DC=cninnovation,DC=local
supportedControl: 1.2.840.113556.1.4.319
supportedControl: 1.2.840.113556.1.4.801
```

## Идентификатор объекта

У каждого объекта имеется свой *глобально уникальный идентификатор* (globally unique identifier — GUID). Этот GUID-идентификатор (как вы уже, возможно, знаете из разработки для COM) представляет собой уникальное 128-битное число. С его помощью можно осуществлять привязку к конкретному объекту. Это гарантирует попадание на один и тот же объект даже в тех случаях, если объект был перемещен в другой контейнер. Идентификатор GUID генерируется при создании объекта и после этого всегда остается неизменным.

Строковое представление GUID-идентификатора можно извлекать с помощью `DirectoryEntry.NativeGuid` и затем использовать для привязки к объекту.

Ниже для примера показано имя пути привязки без участия сервера, с осуществлением привязки к конкретному объекту, представленному идентификатором GUID:

```
LDAP://<GUID=14abbd652aae1a47abc60782dcfc78ea>
```

## Имя пользователя

Если необходимо получить доступ к каталогу от имени какого-то другого пользователя из текущего процесса (у которого может не быть прав на доступ к Active Directory), для процесса привязки должны обязательно явным образом указываться *учетные данные этого пользователя* (user credentials). В Active Directory поддерживается множество способов для указания имени пользователя.



## Регистрация нижнего уровня

При низкоуровневом входе имя пользователя может указываться вместе с именем домена более ранней, чем Windows 2000, версии:

```
домен\имя_пользователя
```

## Отличительное имя

Пользователь может также быть задан с помощью отличительного имени объекта user, например:

```
CN=Administrator, CN=Users, DC=thinktecture, DC=local
```

## Главное пользовательское имя

*Главное пользовательское имя* (user principal name — UPN) объекта определяется с помощью атрибута `userPrincipalName`. Системный администратор указывает его вместе с регистрационной информацией на вкладке Account (Учетная запись) диалогового окна свойств пользователя, доступного из оснастки Active Directory Users and Computers консоли MMC. Обратите внимание, что это не адрес электронной почты пользователя.

Это имя также идентифицирует пользователя уникальным образом и может применяться для регистрации, например:

```
Nagel@thinktecture.local
```

## Аутентификация

Для обеспечения безопасной аутентификации с применением шифрования может также указываться *тип аутентификации*. Это подразумевает использование свойства `AuthenticationType` класса `DirectoryEntry`, которому в качестве значения может быть присвоено одно из значений перечисления `AuthenticationTypes`. Поскольку это перечисление помечено атрибутом `[Flags]`, может быть задано множество значений. Одни из возможных значений позволяют указывать, как должны шифроваться передаваемые данные, значение `ReadonlyServer` — что разрешен только доступ для чтения, а значение — что требуется безопасная аутентификация.

## Привязка с помощью класса DirectoryEntry

Класс `System.DirectoryServices.DirectoryEntry` можно применять для указания всей информации, касающейся привязки. Можно использовать стандартный конструктор и отдельно определять информацию привязки с помощью свойств `Path` (путь), `Username` (имя пользователя), `Password` (пароль) и `AuthenticationType` (тип аутентификации), а можно просто передавать все эти параметры прямо в конструкторе:

```
DirectoryEntry de = new DirectoryEntry();
de.Path = "LDAP://platinum/DC=thinktecture, DC=local";
de.Username = "nagel@thinktecture.local";
de.Password = "password";

// Использовать учетные данные текущего пользователя
DirectoryEntry de2 = new DirectoryEntry("LDAP://DC=thinktecture, DC=local");
```

Успешное создание объекта `DirectoryEntry` вовсе не будет означать, что привязка тоже прошла успешно. Во избежание излишнего сетевого трафика, привязка будет происходить при первом чтении какого-нибудь свойства. При первом получении доступа к объекту становится ясно, существует ли требуемый объект и являются ли указанные учетные данные пользователя правильными.

## Извлечение записей каталога

Теперь, когда уже известно, как задавать атрибуты привязки к объекту в Active Directory, можно переходить к рассмотрению процесса чтения атрибутов объекта. В следующем примере демонстрируется чтение свойств объектов пользователя.

В классе `DirectoryEntry` имеется несколько свойств для получения информации об объекте, а именно — `Name`, `Guid` и `SchemaClassName`. При первом получении доступа к свойству объекта `DirectoryEntry` происходит привязка, в результате чего все его данные помещаются в кэш лежащего в основе объекта ADSI. (Более подробно об этом рассказывается далее в главе.) После этого чтение всех остальных свойств производится уже из упомянутого кэша и, следовательно, необходимость в обмене данными с сервером с целью получения данных из этого же объекта отпадает.

В показанном ниже примере доступ осуществляется к объекту `user`, который имеет общее имя `Christian Nagel` и относится к организационной единице `thinktecture`:

```
using (DirectoryEntry de = new DirectoryEntry())
{
    de.Path = "LDAP://magellan/CN=Christian Nagel," +
             "OU=thinktecture, DC=cninnovation, DC=local";

    Console.WriteLine("Имя: {0}", de.Name);
    Console.WriteLine("GUID: {0}", de.Guid);
    Console.WriteLine("Тип: {0}", de.SchemaClassName);
    Console.WriteLine();
    // ...
}
```

Фрагмент кода *DirectoryServicesSamples\Program.cs*



*Чтобы иметь возможность выполнить этот код на своей машине, измените соответствующим образом путь к объекту, включая имя сервера.*

В объекте Active Directory содержится гораздо больше информации, причем то, какая именно информация доступна, зависит от типа объекта; свойство `Properties` возвращает коллекцию `PropertyCollection`. Каждое из свойств в этой коллекции само по себе тоже является коллекцией, поскольку одно свойство может иметь несколько значений; например, объект `user` может содержать несколько телефонных номеров. В данном случае просмотр всех значений осуществляется с помощью внутреннего цикла `foreach`. Коллекция, возвращаемая свойством `properties[name]`, представляет собой массив `object`. Значения атрибутов могут иметь вид строк, чисел и других типов. Здесь для отображения этих значений применяется метод `ToString()`.

```
Console.WriteLine("Свойства: ");
PropertyCollection properties = de.Properties;
foreach (string name in properties.PropertyNames)
{
    foreach (object o in properties [name])
    {
        Console.WriteLine("{0}: {1}", name, o.ToString());
    }
}
```

Ниже приведены результирующие выходные данные этого кода, в которых можно увидеть все атрибуты указанного объекта `user`. Некоторые из свойств, вроде свойства `otherTelephone`, имеют по несколько значений. В частности, конкретно данное свойство позволяет определять множество телефонных номеров. В некоторых свойствах просто отображается тип объекта — `System.__ComObject`, например, `lastLogoff`,

lastLogon и nTSecurityDescriptor. Для извлечения значений таких атрибутов нужно использовать интерфейсы ADSI COM прямо из классов пространства имен System.DirectoryServices.

```
Имя: CN=Christian Nagel
GUID: 0238fd5c-7e67-48bc-985f-c2f1ccf0f86c
Тип: user
Свойства:
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: user
cn: Christian Nagel
sn: Nagel
givenName: Christian
distinguishedName: CN=Christian Nagel,OU=thinktecture,DC=cninnovation,DC=local
instanceType: 4
whenCreated: 9/25/2009 12:42:05 PM
whenChanged: 9/25/2009 12:42:05 PM
displayName: Christian Nagel
uSNCreated: System.__ComObject
uSNChanged: System.__ComObject
name: Christian Nagel
objectGUID: System.Byte[]
userAccountControl: 66048
badPwdCount: 0
codePage: 0
countryCode: 0
badPasswordTime: System.__ComObject
lastLogoff: System.__ComObject
lastLogon: System.__ComObject
pwdLastSet: System.__ComObject
primaryGroupID: 513
objectSid: System.Byte[]
accountExpires: System.__ComObject
logonCount: 0
sAMAccountName: christian.nagel
sAMAccountType: 805306368
userPrincipalName: christian.nagel@cninnovation.local
objectCategory: CN=Person,CN=Schema,CN=Configuration,DC=cninnovation,DC=local
dSCorePropagationData: 1/1/1601 12:00:00 AM
nTSecurityDescriptor: System.__ComObject
```

С помощью объекта DirectoryEntry.Properties можно получать доступ ко всем свойствам. Если известно имя свойства, доступ к его значениям можно производить непосредственно по имени:

```
foreach (string homePage in de.Properties ["wwwHomePage"])
    Console.WriteLine("Домашняя страница: " + homePage);
```

## Коллекции объектов

Объекты хранятся в Active Directory иерархическим образом. Объекты-контейнеры содержат дочерние объекты. Получить список этих дочерних объектов можно с помощью свойства Children класса DirectoryEntry, а информацию о родительских контейнерах, в которых они содержатся — с помощью свойства Parent.

У объекта user не бывает дочерних объектов, поэтому в следующем примере используется организационная единица. Объекты, не являющиеся контейнерами, для свойства Children возвращают пустую коллекцию объектов. В примере предполагается, что необходимо извлечь все объекты user из организационной единицы thinktecture домена

cninnovation.local. Свойство `Children` возвращает коллекцию `DirectoryEntries` с объектами `DirectoryEntry`. Далее производится проход в цикле по всем этим объектам `DirectoryEntry` для вывода на консоль имен всех дочерних объектов.

```
using (var de = new DirectoryEntry())
{
    de.Path = "LDAP://magellan/OU=thinktecture, DC=cninnovation, DC=local";

    Console.WriteLine("Дочерние объекты в {0}", + de.Name);
    foreach (DirectoryEntry obj in de.Children)
    {
        Console.WriteLine(obj.Name);
    }
}
```

---

Фрагмент кода *DirectoryServicesSamples\Program.cs*

---

Выполнение этого кода приведет к отображению общих имен объектов:

```
Дочерние объекты в OU=thinktecture
OU=Admin
CN=Buddhike de Silva
CN=Christian Nagel
CN=Christian Weyer
CN=Consultants
CN=demos
CN=Dominick Baier
CN=Ingo Rammer
CN=Neno Loye
```

В примере отображаются все объекты в организационной единице: `users`, `contacts`, `printers`, `shares` и т.д. Если должны отображаться только объекты определенного типа, можно воспользоваться свойством `SchemaFilter` класса `DirectoryEntries`. Это свойство возвращает коллекцию `SchemaNameCollection`, которая позволяет применять метод `Add()` и указывать отображаемые типы объектов. Ниже показано, как отобразить только объекты `user`, потому в эту коллекцию добавляется тип `user`.

```
using (var de = new DirectoryEntry())
{
    de.Path = "LDAP://magellan/OU=thinktecture, DC=cninnovation, DC=local";

    Console.WriteLine("Дочерние объекты в {0}", + de.Name);
    de.Children.SchemaFilter.Add("user");
    foreach (DirectoryEntry obj in de.Children)
    {
        Console.WriteLine(obj.Name);
    }
}
```

В результате выполнения этого кода на консоль выводятся только объекты `user` в организационной единице:

```
Дочерние объекты в OU=thinktecture
CN=Christian Nagel
CN=Christian Weyer
CN=Dominick Baier
CN=Ingo Rammer
CN=Jörg Neumann
CN=Richard Blewett
```

## Кэш

Для сокращения операций передачи данных по сети ADSI применяет кэш, запоминая свойства объектов. Как было показано ранее, получение доступа к серверу не происходит при создании объекта `DirectoryEntry`; вместо этого оно осуществляется при первом чтении какого-то значения из хранилища `Active Directory` и сопровождается записью в кэш также и значений всех остальных свойств, тем самым устраняя необходимость в повторном подключении к серверу при получении доступа к следующему свойству.

Внесение каких-либо изменений в объект влияет только на находящуюся в кэше версию, и установка свойств не приводит к генерации сетевого трафика. Для сбрасывания содержимого кэша на диск и передачи измененных данных на сервер необходимо использовать метод `DirectoryEntry.CommitChanges()`. Для извлечения новых записанных данных из хранилища `Active Directory` можно применять метод `DirectoryEntry.RefreshCache()`, который производит чтение свойств. Разумеется, в случае изменения каких-нибудь свойств без применения метода `CommitChanges()` и вызова метода `RefreshCache()` все изменения будут утеряны, поскольку будет просто выполняться повторное чтение значений из службы каталогов с помощью `RefreshCache()`.

Кэш свойств можно отключить, установив для свойства `DirectoryEntry.UsePropertyCache` значение `false`. Однако если только не осуществляется отладка кода, этого лучше не делать из-за дополнительных сеансов обмена данными с сервером, которые в таком случае будут генерироваться.

## Создание новых объектов

Если нужно создать новые объекты `Active Directory`, такие как пользователи, компьютеры, принтеры, контакты и т.п., это можно сделать программно с помощью класса `DirectoryEntries`.

Чтобы добавить новые объекты в каталог, сначала необходимо привязаться к объекту-контейнеру, такому как организационная единица, в который будут включаться новые объекты. В следующем примере используется объект-контейнер с отличительным именем `CN=Users, DC=cninnovation, DC=local`:

```
var de = new DirectoryEntry();
de.Path = "LDAP://magellan/CN=Users, DC=cninnovation, DC=local";
```

Фрагмент кода *DirectoryServicesSamples\Program.cs*

Извлечь объект `DirectoryEntries` можно с помощью свойства `Children` класса `DirectoryEntry`:

```
DirectoryEntries users = de.Children;
```

Класс `DirectoryEntries` предлагает методы для добавления, удаления и поиска объектов в коллекции. В рассматриваемом здесь примере создается новый объект `user`. Метод `Add()` требует указания объекта и имени типа. Имена типов можно извлечь с помощью редактора `ADSI Edit`.

```
DirectoryEntry user = users.Add("CN=John Doe", "user");
```

Сейчас объект имеет стандартные свойства со стандартными значениями. Назначить ему конкретные свойства с определенными значениями можно путем добавления этих свойств в коллекцию `Properties` с помощью метода `Add()`. Разумеется, все свойства должны существовать в схеме объекта `user`. Если указанное свойство не существует, генерируется исключение `COMException` со следующим сообщением: `The specified directory service attribute or value doesn't exist` (Указанного атрибута или значения в службе каталогов не существует).

```
user.Properties["company"].Add("Some Company");
user.Properties["department"].Add("Sales");
user.Properties["employeeID"].Add("4711");
user.Properties["samAccountName"].Add("JDoe");
user.Properties["userPrincipalName"].Add("JDoe@cninnovation.local");
user.Properties["givenName"].Add("John");
user.Properties["sn"].Add("Doe");
user.Properties["userPassword"].Add("someSecret");
```

И, наконец, чтобы записать эти данные в Active Directory, потребуется сбросить содержимое кэша на диск:

```
user.CommitChanges();
```

## Обновление записей в каталоге

Обновление объектов Active Directory производится так же легко, как их чтение. Приступить к изменению значений объекта можно сразу же после его чтения.

Для удаления всех значений какого-то одного свойства можно использовать метод `PropertyValueCollection.Clear()`, для присваивания свойству новых значений — метод `Add()`, а для удаления из коллекции свойств только конкретных значений — методы `Remove()` и `RemoveAt()`.

Чтобы изменить значение, необходимо просто установить новое значение. В следующем примере демонстрируется установка нового значения для номера мобильного телефона с использованием индексатора коллекции `PropertyValueCollection`. В случае применения индексатора значение может изменяться только при условии, что оно существует. Из-за этого с помощью `DirectoryEntry.Properties.Contains()` должна предприниматься проверка доступности атрибута.

```
using (var de = new DirectoryEntry())
{
    de.Path = "LDAP://magellan/CN=Christian Nagel, " +
              "OU=thinkecture, DC=cninnovation, DC=local";
    if (de.Properties.Contains("mobile"))
    {
        de.Properties["mobile"][0] = "+43 (664) 3434343434";
    }
    else
    {
        de.Properties["mobile"].Add("+43 (664) 3434343434");
    }
    de.CommitChanges();
}
```

В части `else` используется метод `PropertyValueCollection.Add()` для добавления свойства, представляющего номер мобильного телефона, если такового еще не существует. В случае применения метода `Add()` с существующими свойствами, конечный результат будет зависеть от типа добавляемого свойства (однозначное или многозначное). Вызов метода `Add()` с однозначным свойством, которое уже существует, приводит к генерации исключения `COMException` со следующим сообщением: `A constraint violation occurred` (Произошло нарушение условия ограничения). Однако вызов метода `Add()` с многозначным свойством проходит успешно и завершается добавлением к свойству еще одного значения.

Свойство `mobile` объекта `user` определено как однозначное, так что дополнительные мобильные номера в него добавляться не могут. Однако пользователь может располагать более чем одним мобильным номером. Для дополнительных мобильных номеров доступно свойство `otherMobile`. Это свойство имеет многозначный тип и позволяет сохранять

множество телефонных номеров и, следовательно, вызывать метод `Add()` множество раз. Обратите внимание, что все значения в этом многозначном свойстве проверяются на предмет уникальности. В случае повторного добавления второго телефонного номера в тот же самый объект `user` генерируется исключение `COMException` со следующим сообщением: `The specified directory service attribute or value already exist` (Указанный атрибут или значение уже существует в службе каталогов).



*После создания новых или обновления существующих объектов в каталоге не забудьте вызвать метод `DirectoryEntry.CommitChanges()`. В противном случае обновляться будут только данные в кэше без передачи изменений в службу каталогов.*

## Получение доступа к собственным объектам ADSI

Часто гораздо проще вызвать методы предопределенных интерфейсов ADSI, чем выполнять поиск имен свойств объектов. Некоторые интерфейсы ADSI также поддерживают методы, которые не могут использоваться прямо из класса `DirectoryEntry`. Одним из примеров таких интерфейсов является `IADsServiceOperations`, который поддерживает методы для запуска и остановки служб Windows. (Службы Windows подробно рассматриваются в главе 25.)

Классы из пространства имен `System.DirectoryServices`, как упоминалось ранее, используют базовые объекты ADSI COM. Класс `DirectoryEntry` позволяет вызывать методы базовых объектов напрямую с применением метода `Invoke()`.

В первом параметре метода `Invoke()` должно быть указано имя метода, который подлжит вызову в объекте ADSI, а во втором — ключевое слово `params` с любым количеством дополнительных аргументов, которые могут передаваться методу ADSI:

```
public object Invoke(string methodName, params object [] args);
```

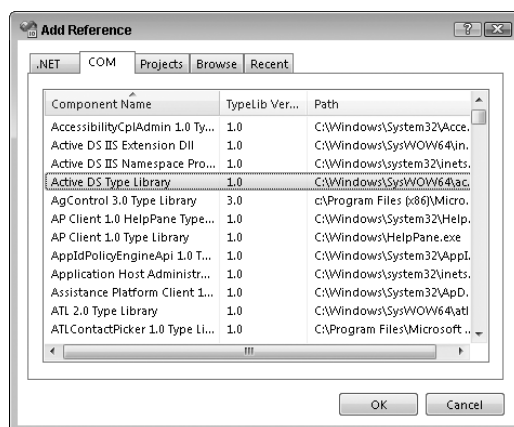
Информацию о методах, которые могут вызываться с помощью метода `Invoke()`, можно найти в документации по ADSI. Каждый объект в домене поддерживает методы интерфейса `IADs`. Созданный в предыдущих примерах объект `user` также поддерживает методы интерфейса `IADsUser`.

В следующем примере показано изменение пароля для этого созданного ранее объекта `user` с помощью метода `IADsUser.SetPassword()`:

```
using (var de = new DirectoryEntry())
{
    de.Path = "LDAP://magellan/CN=John Doe, " +
              "CN=Users, DC=cninnovation, DC=local";
    de.Invoke("SetPassword", "anotherSecret");
    de.CommitChanges();
}
```

Вместо метода `Invoke()` также можно использовать непосредственно сам базовый объект ADSI. Для этого выберите в меню **Project (Проект)** пункт **Add Reference (Добавить ссылку)** и добавьте ссылку на библиотеку типов **Active DS Type Library** (рис. 52.9). Выполнение этого действия приводит к созданию класса-оболочки, из которого потом можно получать доступ к базовым объектам ADSI в пространстве имен `ActiveDs`.

Доступ к встроенному объекту может осуществляться с помощью свойства `NativeObject` класса `DirectoryEntry`. В следующем примере объект `de` является объектом типа `user`, а это значит, что он может быть приведен к типу `ActiveDs`. Метод `IADsUser.SetPassword()` представляет собой метод, документированный в интерфейсе `IADsUser`, а это значит, что его можно вызвать вместо метода `Invoke()`.



**Рис. 52.9.** Добавление ссылки на библиотеку типов *Active DS Type Library*

Установка для свойства `AccountDisabled` интерфейса `IADsUser` значения `false` позволяет активизировать учетную запись. Как и в предыдущих примерах, напоследок все изменения записываются в службу каталогов вызовом метода `CommitChanges()` объекта `DirectoryEntry`:

```
ActiveDs.IADsUser user = (ActiveDs.IADsUser)de.NativeObject;
user.SetPassword("someSecret");
user.AccountDisabled = false;
de.CommitChanges();
```



*Начиная с версии .NET 3.5, снизилась необходимость в вызове собственных объектов .NET-класса `DirectoryEntry`. Для управления пользователями можно применять новые классы в пространстве имен `System.DirectoryServices.AccountManagement`. Эти классы более подробно рассматриваются позже в главе.*

## Выполнение поиска в Active Directory

Поскольку Active Directory является хранилищем данных, которое оптимизировано главным образом под доступ только для чтения, в нем довольно часто требуется выполнять поиск значений. Для поиска в Active Directory в .NET Framework предусмотрен специальный класс `DirectorySearcher`.



*Класс `DirectorySearcher` может применяться только с поставщиком LDAP; с другими поставщиками, такими как NDS или IIS, он работать не будет.*

В конструкторе класса `DirectorySearcher` можно определять четыре основных параметра для поиска: корень для начала поиска, фильтр, свойства, которые должны быть загружены, и область поиска. Кроме того, также можно использовать конструктор по умолчанию и задавать параметры поиска с помощью свойств.

### Параметр `SearchRoot`

Параметр `SearchRoot` позволяет указывать, откуда должен начинаться поиск. По умолчанию он ссылается на корень используемого в текущий момент домена. Задается он с помощью свойства `Path` объекта `DirectoryEntry`.



## Параметр фильтра

Параметр фильтра позволяет указывать значения, среди которых должен осуществляться поиск совпадений. Он представляет собой заключаемую в круглые скобки строку.

В выражениях фильтра допускается применять операции отношения, вроде `<=`, `=` и `>=`. Выражение `(objectClass=contact)` означает, что необходимо выполнить поиск всех объектов типа `contact`, а выражение `(lastName>=Nagel)` — что требуется выполнить поиск в алфавитном порядке всех объектов, у которых свойство `lastName` равно или больше `Nagel`.

Выражения фильтра также могут комбинироваться с помощью префиксных операций `&` и `|`. Например, выражение `(&(objectClass=user)(description=Auth*))` означает необходимость поиска всех объектов типа `user`, у которых описание свойств начинается со строки `Auth`. Поскольку операции `&` и `|` размещаются в начале выражений, с помощью единственной префиксной операции допустимо комбинировать более двух выражений.

По умолчанию выражение фильтра выглядит как `(objectClass=*)` и означает, что при поиске действительными должны считаться все объекты.



*Определение синтаксиса фильтра содержится в документе RFC 2254 “The String Representation of LDAP Search Filters” (“Строковое представление фильтров поиска LDAP”), который доступен по адресу [www.ietf.org/rfc/rfc2254.txt](http://www.ietf.org/rfc/rfc2254.txt).*

## Параметр `PropertiesToLoad`

Параметр `PropertiesToLoad` позволяет определять коллекцию `StringCollection` и указывать в ней только интересующие свойства. Часто в объектах доступно множество свойств, большинство из которых может не иметь отношения к проводимому поиску. Параметр `PropertiesToLoad` позволяет указывать только те свойства, которые должны загружаться в кэш. Если никаких свойств не указано, по умолчанию возвращаются свойства пути и имени объекта.

## Параметр `SearchScope`

Параметр `SearchScope` представляет собой перечисление, определяющее глубину поиска.

- Значение `SearchScope.Base` указывает, что поиск должен проводиться только в атрибутах объекта, с которого был начат поиск, следовательно, подразумевает обнаружение максимум одного объекта.
- Значение `SearchScope.OneLevel` указывает, что поиск должен продолжаться в коллекции дочерних объектов базового объекта. Поиск в самом базовом объекте не производится.
- Значение `SearchScope.Subtree` указывает, что поиск должен осуществляться по всему дереву объектов.

Значением по умолчанию для свойства `SearchScope` является `SearchScope.Subtree`.

## Область поиска

Поиск конкретных объектов в службе каталогов может охватывать сразу несколько доменов. Для ограничения области поиска конкретным количеством объектов или зачисляемым временем предусмотрены дополнительные свойства, которые перечислены в табл. 52.3.

**Таблица 52.3. Дополнительные свойства для ограничения области поиска**

Свойство	Описание
ClientTimeout	Максимальный промежуток времени, в течение которого клиент ждет возврата результатов от сервера. Если сервер не отвечает, никакие записи не возвращаются.
PageSize	В случае применения режима <i>постраничного поиска</i> сервер возвращает не весь результат, а только ряд объектов, определенных с помощью свойства <code>PageSize</code> . Это сокращает количество времени, необходимого для получения клиентом первого ответа, а также объем требуемой памяти. Сервер отправляет клиенту cookie-набор, который тот затем отправляет обратно серверу вместе со следующим запросом на выполнение поиска, благодаря чему поиск может продолжаться с точки, в которой он был завершен в предыдущий раз.
ServerPageTimeLimit	В случае постраничного поиска значение этого свойства указывает, на протяжении какого промежутка времени должен продолжаться поиск того количества объектов, которое было определено в <code>PageSize</code> . Если указанный промежуток времени истекает до обнаружения заданного в <code>PageSize</code> количества объектов, клиенту будут возвращены только те объекты, которые удалось найти до того момента. По умолчанию для данного свойства устанавливается значение <code>-1</code> , означающее, что поиск может выполняться до бесконечности.
SizeLimit	Позволяет указывать максимальное количество объектов, которое должно возвращаться в результате поиска. В случае установки для него значения, превышающего граничное значение сервера (равное <code>1000</code> ), будет использоваться значение сервера.
ServerTimeLimit	Позволяет указывать максимальный промежуток времени, на протяжении которого сервер должен выполнять поиск объектов. По истечении этого промежутка времени все обнаруженные до объекты возвращаются клиенту. По умолчанию значение этого свойства равно <code>120</code> секунд и устанавливать для него более высокое значение нельзя.
ReferralChasing	Поиск может охватывать несколько доменов. Если задать в <code>SearchRoot</code> в качестве корня родительский домен или вообще ничего, то поиск может быть продолжен на дочерние домены. Данное свойство позволяет указывать, должен ли поиск продолжаться на других серверах. Значение <code>ReferralChasingOption.None</code> означает, что поиск не должен быть продолжен на других серверах. Значение <code>ReferralChasingOption.Subordinate</code> определяет, что поиск должен охватывать и дочерние домены. Если поиск начинается с <code>DC=Wrox</code> , <code>DC=com</code> , сервер может вернуть результирующий набор и перенаправление на <code>DC=France</code> , <code>DC=Wrox</code> , <code>DC=COM</code> , а клиент сможет продолжить поиск в этом поддомене. Значение <code>ReferralChasingOption.External</code> указывает, что сервер может перенаправлять клиента на отдельный сервер, не относящийся к данному поддомену. Это значение по умолчанию. Если указано значению <code>ReferralChasingOption.All</code> , возвращаться будут как внешние, так и подчиненные перенаправления.
Tombstone	В случае установки для свойства <code>Tombstone</code> значения <code>true</code> возвращаться будут также и все удаленные объекты, которые соответствуют параметрам поиска.
VirtualListView	Если после поиска ожидается получение результатов большого объема, с помощью этого свойства можно определить подмножество, которое должно быть возвращено. Подмножество определяется с помощью класса <code>DirectoryVirtualListView</code> .

В приведенном ниже примере производится поиск в организационной единице `thinktexture` всех объектов `user`, которые имеют в свойстве `description` значение `Author`.

Первым делом осуществляется привязка к организационной единице `thinktexture`. Именно отсюда должен начинаться поиск. Далее создается объект `DirectorySearcher` и устанавливается значение для свойства `SearchRoot`. Фильтр определяется как `(&(objectClass=user)(description=Auth*))`, чтобы поиск охватывал все объекты типа `user`, описание которых начинается с `Auth`. Для свойства `SearchScope` устанавливается значение `Subtree`, чтобы поиск выполнялся также и в дочерних организационных единицах внутри `thinktexture`.

```

using (var de = new DirectoryEntry("LDAP://OU=thinktexture, DC=cninnovation,
    DC=local"))
using (var searcher = new DirectorySearcher())
{
    searcher.SearchRoot = de;
    searcher.Filter = "&(objectClass=user)(description=Auth*)";
    searcher.SearchScope = SearchScope.Subtree;

```

Фрагмент кода *DirectoryServicesSamples\Program.cs*

В результате поиска должны возвращаться свойства `name`, `description`, `givenName` и `WWWHomePage`:

```

searcher.PropertiesToLoad.Add("name");
searcher.PropertiesToLoad.Add("description");
searcher.PropertiesToLoad.Add("givenName");
searcher.PropertiesToLoad.Add("WWWHomePage");

```

Теперь все готово к выполнению поиска. Однако результат должен быть также отсортирован. `DirectorySearcher` имеет свойство `Sort`, в котором можно задать опцию `SortOption`. Первый аргумент в конструкторе класса `SortOption` отвечает за свойство, которое будет использоваться для сортировки, а второй — за порядок сортировки. Перечисление `SortDirection` поддерживает значения `Ascending` (по возрастанию) и `Descending` (по убыванию).

Для запуска поиска можно использовать метод `FindOne()`, отыскивающий первый объект, или метод `FindAll()`. Метод `FindOne()` возвращает объект `SearchResult`, а метод `FindAll()` — коллекцию `SearchResultCollection`. В следующем коде должны возвращаться все авторы, поэтому применяется метод `FindAll()`:

```

searcher.Sort = new SortOption("givenName", SortDirection.Ascending);
SearchResultCollection results = searcher.FindAll();

```

В цикле `foreach` производится доступ ко всем объектам `SearchResult` в коллекции `SearchResultCollection`. Пот этом `SearchResult` представляет одиночный объект в кэше поиска. Свойство `Properties` возвращает коллекцию `ResultPropertyCollection`, доступ ко всем свойствам и значениям которой осуществляется с помощью имени свойства и индексатора.

```

SearchResultCollection results = searcher.FindAll();
foreach (SearchResult result in results)
{
    ResultPropertyCollection props = result.Properties;
    foreach (string propName in props.PropertyNames)
    {
        Console.WriteLine("{0}: ", propName);
        Console.WriteLine(props[propName][0]);
    }
    Console.WriteLine();
}
}

```

После поиска можно также извлечь объект целиком: `SearchResult` имеет метод `GetDirectoryEntry()`, который возвращает соответствующую запись `DirectoryEntry` обнаруженного объекта.

Выполнение этого кода приведет к выводу списка всех ассоциированных с `thinktecture` объектов с выбранными свойствами:

```
givenname: Christian
adspath: LDAP://magellan/CN=Christian Weyer,OU=thinktecture,DC=cninnovation,
DC=local
description: Author
name: Christian Weyer
givenname: Christian
adspath: LDAP://magellan/CN=Christian Nagel,OU=thinktecture,DC=cninnovation,
DC=local
description: Author
name: Christian Nagel
givenname: Dominick
adspath: LDAP://magellan/CN=Dominick Baier,OU=thinktecture,DC=cninnovation,
DC=local
description: Author
name: Dominick Baier
givenname: Ingo
adspath: LDAP://magellan/CN=Ingo Rammer,OU=thinktecture,DC=cninnovation,
DC=local
description: Author
name: Ingo Rammer
givenname: Jörg
adspath: LDAP://magellan/CN=Jörg Neumann,OU=thinktecture,DC=cninnovation,
DC=local
description: Author
name: Jörg Neumann
```

## Поиск пользовательских объектов

В этом разделе будет построено приложение WPF под названием `UserSearch`. Это приложение является гибким, поскольку предусматривает возможность указания для получения доступа к Active Directory конкретного контроллера домена, имени пользователя и пароля; в противном случае применяются данные пользователя текущего процесса. В этом приложении производится доступ к схеме службы Active Directory для извлечения свойств объекта `user`. Пользователь может вводить строку фильтра для выполнения поиска всех объектов `user` в домене, а также указывать, какие именно свойства объектов `user` должны отобразиться на экране.

## Пользовательский интерфейс

Пользовательский интерфейс реализован в виде пронумерованных шагов, описывающих использование приложения (рис. 52.10).

1. Сначала заполняются поля `Username` (Имя пользователя), `Password` (Пароль) и `Domain Controller` (Контроллер домена). Вся эта информация не является обязательной. Если не указан контроллер домена, соединение работает без привязки к какому-либо серверу. Если имя пользователя отсутствует, принимается контекст безопасности текущего пользователя.
2. С помощью кнопки `Load Properties` (Загрузить свойства) можно динамически загрузить все имена свойств объекта `user` в окно списка `listBoxProperties`.
3. Теперь можно выбирать свойства, которые должны быть отображены на экране. В качестве режима `SelectionMode` в окне списка устанавливается `MultiSimple`.

4. На этом шаге можно задать фильтр для ограничения поиска. Значение по умолчанию, устанавливаемое в этом поле, определяет поиск всех объектов `user:` (`objectClass=user`).
5. Сейчас можно начать поиск, щелкнув на кнопке Search (Поиск).

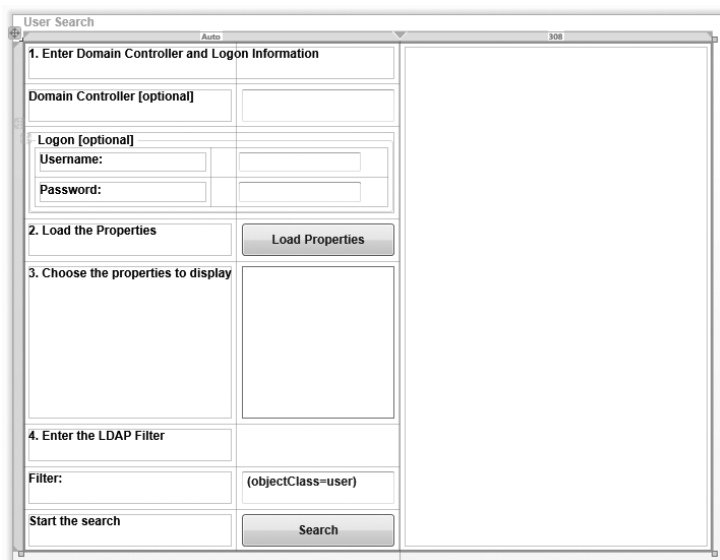


Рис. 52.10. Интерфейс пользователя приложения поиска

## Извлечение контекста именования схемы

В этом приложении есть только два метода обработки: один для кнопки Load Properties, предназначенной для загрузки свойств, и второй для запуска поиска в домене. Прежде всего, из схемы динамически читаются свойства класса `user`, чтобы отобразить их в пользовательском интерфейсе.

В методе обработчика кнопки `buttonLoadProperties_Click()` функция `SetLogonInformation()` читает имя пользователя, пароль и имя хоста из текстовых полей и запоминает их в элементах класса. Далее метод `SetNamingContext()` устанавливает LDAP-имя схемы и LDAP-имя контекста по умолчанию. LDAP-имя схемы используется в вызове `SetUserProperties()` для установки значений свойств в окне списка.

```

❏ private void OnLoadProperties(object sender, RoutedEventArgs e)
{
    try
    {
        SetLogonInformation();
        SetNamingContext();
        SetUserProperties(schemaNamingContext);
    }
    catch (Exception ex)
    {
        // Сообщить о необходимости проверки введенных значений
        MessageBox.Show(String.Format("check your input! {0}", ex.Message));
    }
}

```

Фрагмент кода `UserSearch\MainWindow.xaml.cs`

Во вспомогательном методе `SetNamingContext()` для извлечения свойств сервера используется корень дерева каталогов. В данном случае интерес представляют значения только двух следующих свойств: `schemaNamingContext` и `defaultNamingContext`.

```
private void SetNamingContext()
{
    using (DirectoryEntry de = new DirectoryEntry())
    {
        string path = "LDAP://" + hostname + "rootDSE";
        de.Username = username;
        de.Password = password;
        de.Path = path;
        schemaNamingContext = de.Properties["schemaNamingContext"][0].ToString();
        defaultNamingContext = de.Properties["defaultNamingContext"][0].ToString();
    }
}
```

## Извлечение имен свойств пользовательского класса

Для доступа к схеме имеется LDAP-имя, которым можно воспользоваться для получения доступа к каталогу и чтения свойств. Интерес представляют свойства не только класса `user`, но и таких его базовых классов, как `Organizational-Person`, `Person` и `Top`. В рассматриваемом примере имена этих базовых классов жестко закодированы, но в принципе их можно было бы читать динамически с помощью атрибута `subClassOf`.

Метод `GetSchemaProperties()` возвращает `IEnumerable<string>` с именами всех свойств конкретного типа объектов. Имена всех этих свойств добавляются в окно списка.

```
private void SetUserProperties(string schemaNamingContext)
{
    var properties = from p in GetSchemaProperties(schemaNamingContext,
                                                    "User").Concat(
                                                    GetSchemaProperties(schemaNamingContext,
                                                                    "Organizational-Person")).Concat(
                                                    GetSchemaProperties(schemaNamingContext, "Top"))
    orderby p
    select p;
    listBoxProperties.DataContext = properties;
}
```

Фрагмент кода `UserSearch\MainWindow.xaml.cs`

В методе `GetSchemaProperties()` снова производится доступ в Active Directory. На этот раз вместо `rootDSE` используется LDAP-имя обнаруженной ранее схемы. В свойстве `systemMayContain` содержится коллекция всех атрибутов, которые являются допустимыми в классе `objectType`.


```
private IEnumerable<string> GetSchemaProperties(string
                                                schemaNamingContext, string objectType)
{
    IEnumerable<string> data;
    using (var de = new DirectoryEntry())
    {
        de.Username = username;
        de.Password = password;
        de.Path = String.Format("LDAP://{0}CN={1},{2}", hostname,
                                objectType, schemaNamingContext);
        PropertyValueCollection values = de.Properties["systemMayContain"];
        data = from s in values.Cast<string>()
            orderby s
            select s;
    }
    return data;
}
```

На этом выполнение второго шага в рассматриваемом приложении завершено. Имена всех свойств объектов `user` теперь содержатся в элементе управления `ListBox`.

## Поиск пользовательских объектов

Обработчик события кнопки поиска вызывает только вспомогательный метод `FillResult()`:

```

 private void OnSearch(object sender, System.EventArgs e)
{
    try
    {
        FillResult();
    }
    catch (Exception ex)
    {
        // Сообщить о необходимости проверки введенных значений
        MessageBox.Show(String.Format("check your input! {0}", ex.Message));
    }
}

```

Фрагмент кода *UserSearch\MainWindow.xaml.cs*

Внутри `FillResult()` выполняется обычный поиск по всему домену Active Directory, как уже показывалось раньше. Свойству `SearchScope` присваивается значение `Subtree`, свойству `Filter` — получаемая из объекта `TextBox` строка, а свойствам, подлежащим загрузке в кэш — значения, которые пользователь выбрал в окне списка. Свойство `PropertiesToLoad` объекта `DirectorySearcher` является свойством типа `StringCollection`, при котором загружаемые свойства могут добавляться с помощью метода `AddRange()`, требующего указания массива строк. Загружаемые свойства читаются из `ListBox` по имени `listBoxProperties` с помощью свойства `SelectedItems`. После установки свойств объекта `DirectorySearcher` по всем свойствам производится поиск с помощью вызова метода `SearchAll()`. Результат поиска помещается в коллекцию `SearchResultCollection` и используется для генерирования итоговой информации, которая затем помещается в текстовое поле `textBoxResults`.

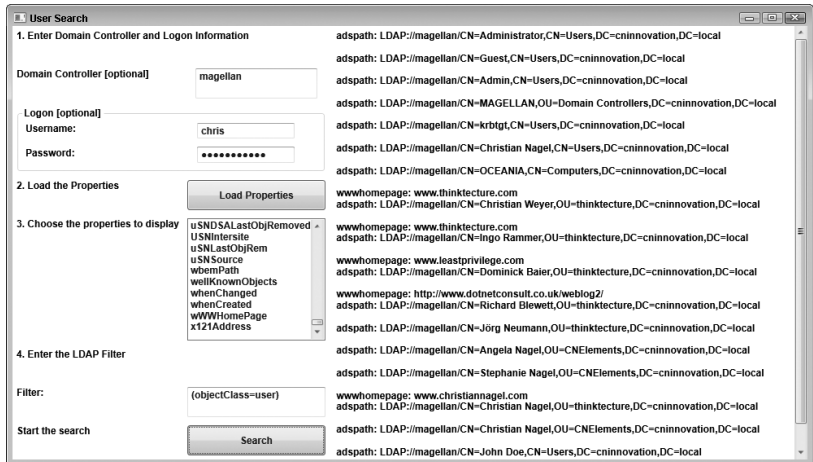
```

private void FillResult()
{
    using (var root = new DirectoryEntry())
    {
        root.Username = username;
        root.Password = password;
        root.Path = String.Format("LDAP://{0}/{1}",
                                hostname, defaultNamingContext);
        using (var searcher = new DirectorySearcher())
        {
            searcher.SearchRoot = root;
            searcher.SearchScope = SearchScope.Subtree;
            searcher.Filter = textFilter.Text;
            searcher.PropertiesToLoad.AddRange(
                listBoxProperties.SelectedItems.Cast < string > ().ToArray());
            SearchResultCollection results = searcher.FindAll();
            var summary = new StringBuilder();
            foreach (SearchResult result in results)
            {
                foreach (string propName in result.Properties.PropertyNames)
                {
                    foreach (object p in result.Properties[propName])
                    {
                        summary.AppendFormat(" {0}: {1}", propName, p);
                    }
                }
            }
        }
    }
}

```

```
        summary.AppendLine();
    }
    summary.AppendLine();
}
textResult.Text = summary.ToString();
}
```

Запуск этого приложения приведет к получению списка всех объектов, соответствующих заданному фильтру (рис. 52.11).



*Рис. 52.11. Результат фильтрации объектов*

## Управление учетными записями

До выхода версии .NET 3.5 создавать и изменять учетные записи пользователей и групп было довольно трудно. Один из способов предусматривал использование классов из пространства имен `System.DirectoryServices` или строго типизированных встроенных интерфейсов COM. Начиная с версии .NET 3.5, стала доступной новая сборка `System.DirectoryServices.AccountManagement`, которая позволяет абстрагироваться от классов `System.DirectoryServices`, предлагая специальные методы и свойства для поиска, изменения, создания и обновления учетных записей пользователей и групп.

В табл. 52.4 описаны классы, доступные в этой сборке.

**Таблица 52.4. Классы, доступные в сборке `System.DirectoryServices.AccountManagement`**

Класс	Описание
<code>PrincipalContext</code>	С помощью класса <code>PrincipalContext</code> конфигурируется контекст управления учетными записями. Здесь можно указывать то, должен ли использоваться домен Active Directory, учетные записи из локальной системы или каталог приложений, путем установ-ки для перечисления <code>ContextType</code> , соответственно, значения <code>Domain</code> , <code>Machine</code> или <code>ApplicationDirectory</code> . В зависи-мости от типа контекста, можно также определять имя домена и указывать имя пользователя и пароль, которые должны использо-ваться для доступа.



Класс	Описание
<code>Principal</code>	Класс <code>Principal</code> является базовым классом всех принципалов. С помощью статического метода <code>FindByIdentity()</code> можно извлекать объект идентификационных данных <code>Principal</code> , наличие которого открывает доступ к различным свойствам, таким как имя, описание, отличительное имя и тип объекта из схемы. Если нужно больше возможностей для управления принципалом, чем доступно при использовании свойств и методов этого класса, можно применять метод <code>GetUnderlyingType()</code> , который возвращает лежащий в основе объект <code>DirectoryEntry</code> .
<code>AuthenticablePrincipal</code>	Класс <code>AuthenticablePrincipal</code> является производным от класса <code>Principal</code> и представляет собой базовый класс для всех принципалов, которые могут аутентифицироваться. Он поддерживает несколько статических методов для поиска принципалов, например, по времени входа в систему или блокировки учетной записи, по попыткам ввода неправильного пароля и по времени установки пароля. С помощью методов экземпляра можно изменять пароль и снимать блокировку с учетной записи.
<code>UserPrincipal</code> <code>ComputerPrincipal</code>	Классы <code>UserPrincipal</code> и <code>ComputerPrincipal</code> унаследованы от базового класса <code>AuthenticablePrincipal</code> и потому имеют все те же, что и он, свойства и методы. Объект <code>UserPrincipal</code> отображается на учетную запись пользователя, а объект <code>ComputerPrincipal</code> — на учетную запись компьютера. <code>UserPrincipal</code> имеет много свойств для извлечения и установки информации о пользователе, например, <code>EmployeeId</code> , <code>EmailAddress</code> , <code>GivenName</code> и <code>VoiceTelephoneNumber</code> .
<code>GroupPrincipal</code>	Группы не могут быть аутентифицированы, и именно поэтому класс <code>GroupPrincipal</code> унаследован прямо от <code>Principal</code> . В классе <code>GroupPrincipal</code> можно извлекать информацию о членах группы с помощью свойства <code>Members</code> и метода <code>GetMembers()</code> .
<code>PrincipalCollection</code>	Класс <code>PrincipalCollection</code> содержит группу объектов <code>Principal</code> ; например, свойство <code>Members</code> из класса <code>GroupPrincipal</code> возвращает объект <code>PrincipalCollection</code> .
<code>PrincipalSearcher</code>	Класс <code>PrincipalSearcher</code> является абстрактной версией класса <code>DirectorySearcher</code> и специально предназначен для управления учетными записями. В случае применения класса <code>PrincipalSearcher</code> владеть синтаксисом составления запросов LDAP вовсе не обязательно, потому что он генерируется автоматически.
<code>PrincipalSearchResult&lt;T&gt;</code>	<code>PrincipalSearchResult&lt;T&gt;</code> возвращается методами поиска из классов <code>PrincipalSearcher</code> и <code>Principal</code> .

В следующих разделах приводятся некоторые сценарии, в которых можно использовать классы из пространства имен `System.DirectoryServices.AccountManagement`.

## Отображение информации о пользователе

Статическое свойство `Current` класса `UserPrincipal` возвращает объект `UserPrincipal` с информацией о текущем вошедшем пользователе:

```

using (var user = UserPrincipal.Current)
{
    Console.WriteLine("Сервер контекста: {0}", user.Context.ConnectedServer);
    Console.WriteLine(user.Description);
    Console.WriteLine(user.DisplayName);
    Console.WriteLine(user.EmailAddress);
    Console.WriteLine(user.GivenName);
    Console.WriteLine("{0:d}", user.LastLogon);
    Console.WriteLine(user.ScriptPath);
}

```

---

Фрагмент кода *AccountManagementSamples\Program.cs*

---

Запуск этого приложения приведет к отображению информации о пользователе, как показано ниже:

```

Сервер контекста: Magellan.cninnovation.local
Developer, Author, Trainer, Consultant
Christian Nagel
Christian@ChristianNagel.com
Christian
2009/09/25

```

## Создание пользователя

Для создания нового пользователя применяется класс `UserPrincipal`. Сначала требуется с помощью `PrincipalContext` определить, где должен быть создан пользователь. Свойство `ContextType` устанавливается в значение перечисления `Domain`, `Machine` или `Application`, в зависимости от того, должна использоваться служба каталогов, локальные учетные записи компьютера или каталог приложений. Если у текущего пользователя нет прав на добавление учетных записей в Active Directory, с помощью класса `PrincipalContext` можно также указать имя пользователя и пароль, которые должны применяться для доступа к серверу.

Далее можно создать экземпляр `UserPrincipal`, передавая контекст принципа и устанавливая все требуемые свойства. В показанном ниже коде устанавливаются свойства `GivenName` и `EmailAddress`. И, наконец, последнее, что нужно сделать — это вызвать метод `Save()` объекта `UserPrincipal` для записи нового пользователя в хранилище.

```

using (var context = new PrincipalContext(ContextType.Domain, "cninnovation"))
using (var user = new UserPrincipal(context, "Tom", "P@ssw0rd", true)
{
    GivenName = "Tom",
    EmailAddress = "test@test.com"
})
{
    user.Save();
}

```

## Сброс пароля

Для сброса пароля существующего пользователя можно использовать метод `SetPassword()` соответствующего объекта `UserPrincipal`:

```

using (var context = new PrincipalContext(ContextType.Domain, "cninnovation"))
using (var user = UserPrincipal.FindByIdentity(context, IdentityType.Name, "Tom"))
{
    user.SetPassword("Pa$$w0rd");
    user.Save();
}

```

---

Фрагмент кода *AccountManagementSamples\Program.cs*

---

Пользователь, выполняющий этот код, должен иметь права на сброс пароля. Чтобы просто поменять старый пароль на новый, можно воспользоваться методом `ChangePassword()`.

## Создание группы

Новая группа создается практически таким же образом, что и новый пользователь. Здесь просто вместо класса `UserPrincipal` используется класс `GroupPrincipal`. Как и при создании нового пользователя, при создании новой группы должны быть установлены свойства и вызван метод `Save()`.

```

using (var ctx = new PrincipalContext(ContextType.Domain, "cninnovation"))
using (var group = new GroupPrincipal(ctx)
    {
        Description = "Sample group",
        DisplayName = "Wrox Authors",
        Name = "WroxAuthors"
    })
{
    group.Save();
}
```

Фрагмент кода *AccountManagementSamples\Program.cs*

## Добавление пользователя в группу

Для добавления пользователя в группу необходимо воспользоваться классом `GroupPrincipal` и добавить в его свойство `Members` нужный объект `UserPrincipal`. Для извлечения информации о существующем пользователе и группе вызывается статический метод `FindByIdentity()`.

```

using (var context = new PrincipalContext(ContextType.Domain))
using (var group = GroupPrincipal.FindByIdentity(
    context, IdentityType.Name, "WroxAuthors"))
using (var user = UserPrincipal.FindByIdentity(
    context, IdentityType.Name, "Stephanie Nagel"))
{
    group.Members.Add(user);
    group.Save();
}
```

Фрагмент кода *AccountManagementSamples\Program.cs*

## Поиск пользователей

Статические методы объекта `UserPrincipal` позволяют находить пользователей на основе заранее определенных критериев. В приведенном здесь примере демонстрируется поиск пользователей, которые не изменяли свой пароль на протяжении последних 30 дней, с помощью метода `FindPasswordSetTime()`. Этот метод возвращает коллекцию `PrincipalSearchResult<UserPrincipal>`, которая проходит в цикле для отображения имени пользователя, времени последнего входа в систему и времени сброса пароля.

```

using (var context = new PrincipalContext(ContextType.Domain, "cninnovation"))
using (var users = UserPrincipal.FindByPasswordSetTime(context,
    DateTime.Today - TimeSpan.FromDays(30), MatchType.LessThan))
{
    foreach (var user in users)
    {
        Console.WriteLine("{0}, последний вход: {1}, " +
            "последнее изменение пароля: {2}", user.Name, user.LastLogon,
```

```

        user.LastPasswordSet);
    }
}

```

Фрагмент кода *AccountManagementSamples\Program.cs*

К числу других предлагаемых классом `UserPrincipal` методов для поиска пользователей относятся `FindByBadPasswordAttempt()`, `FindByExpirationTime()`, `FindByLockoutTime()` и `FindByLogonTime()`.

С помощью класса `PrincipalSearcher` можно достигнуть большей гибкости в поиске пользователей. Этот класс является абстрактной версией класса `DirectorySearcher` и внутренне использует его. В случае применения класса `PrincipalSearcher` свойству `QueryFilter` может быть присвоен любой объект `Principal`.

В приведенном ниже коде свойству `QueryFilter` присваивается объект `UserPrincipal`, имеющий свойства `Surname` и `Enabled`. Благодаря этому, все объекты пользователей со значением `Nag` в свойстве `Surname` и значением `true` в свойстве `Enabled` возвращаются в коллекции `PrincipalSearchResult`, на основе которой класс `PrincipalSearcher` создаст строку запроса LDAP для выполнения поиска.

```

var context = new PrincipalContext(ContextType.Domain);
var userFilter = new UserPrincipal(context);
userFilter.Surname = "Nag*";
userFilter.Enabled = true;
using (var searcher = new PrincipalSearcher())
{
    searcher.QueryFilter = userFilter;
    var searchResult = searcher.FindAll();
    foreach (var user in searchResult)
    {
        Console.WriteLine(user.Name);
    }
}

```

Фрагмент кода *AccountManagementSamples\Program.cs*

## Язык DSML

С помощью пространства имен `System.DirectoryServices.Protocols` доступ к Active Directory можно получать с использованием языка DSML (Directory Services Markup Language — язык разметки для служб каталогов). Язык DSML представляет собой стандарт, который был определен группой OASIS ([www.oasis-open.org](http://www.oasis-open.org)), и позволяет получать доступ к службе каталогов через веб-службу. Возможность доступа к Active Directory посредством DSML предоставляет, как минимум, версия Windows Server 2003 R2.

На рис. 52.12 показан конфигурационный сценарий с DSML. Система, предлагающая службы DSML, получает доступ к Active Directory через LDAP. На клиентской системе для создания SOAP-запросов к службе DSML применяются классы DSML из пространства имен `System.DirectoryServices.Protocols`.

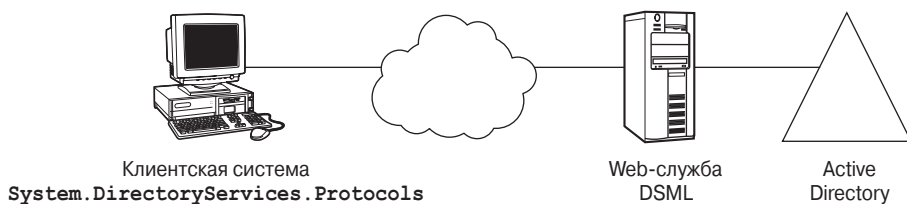


Рис. 52.12. Конфигурационный сценарий с DSML

## Классы в пространстве имен

### **System.DirectoryServices.Protocols**

В табл. 52.5 перечислены основные классы, которые доступны в пространстве имен `System.DirectoryServices.Protocols`.

**Таблица 52.5. Основные классы в пространстве имен `System.DirectoryServices.Protocols`**

Класс	Описание
<code>DirectoryConnection</code>	<code>DirectoryConnection</code> является базовым классом для всех классов соединений и может применяться для определения параметров соединения со службой каталогов. К числу унаследованных от него классов относятся <code>LdapConnection</code> (предназначенный для использования протокола LDAP), <code>DsmlSoapConnection</code> и <code>DsmlSoapHttpConnection</code> . Метод <code>SendRequest</code> позволяет отправлять службе каталогов различные сообщения.
<code>DirectoryRequest</code>	Запрос, который может отправляться службе каталогов, определяется с помощью класса, унаследованного от базового класса <code>DirectoryRequest</code> . В зависимости от типа запроса, для его отправки могут использоваться такие классы, как <code>SearchRequest</code> , <code>AddRequest</code> , <code>DeleteRequest</code> и <code>ModifyRequest</code> .
<code>DirectoryResponse</code>	Результат, который возвращает метод <code>SendRequest</code> , представляет собой экземпляр класса, унаследованного от базового класса <code>DirectoryResponse</code> . Примерами производных классов могут служить <code>SearchResponse</code> , <code>AddResponse</code> , <code>DeleteResponse</code> и <code>ModifyResponse</code> .

## Поиск объектов Active Directory с помощью DSML

В этом разделе приведен пример поиска объектов в службе каталогов. Сначала создается экземпляр объекта `DsmlSoapHttpConnection`, в котором определяется соединение со службой DSML. Это соединение определяется с помощью класса `DsmlDirectoryIdentifier`, содержащего объект `Uri`. При желании вместе с соединением также могут быть заданы учетные данные пользователя.

```

Uri uri = new Uri("http://dsmlserver/dsml");
var identifier = new DsmlDirectoryIdentifier(uri);
var credentials = new NetworkCredential();
credentials.UserName = "cnagel";
credentials.Password = "password";
credentials.Domain = "explorer";
var dsmlConnection = new DsmlSoapHttpConnection(identifier, credentials);

```

Фрагмент кода *DsmlSample\Program.cs*

После определения соединения можно сконфигурировать поисковый запрос. Поисковый запрос состоит из записи каталога, с которой должен начинаться поиск, фильтра LDAP, применяемого при поиске, и значений свойств, которые должны возвращаться после поиска. В примере для фильтра устанавливается значение (`objectClass=user`), чтобы поиск возвращал все объекты пользователей. Для свойства `attributesToReturn` устанавливается значение `null`, чтобы считывались все атрибуты, имеющие значение. `SearchScope` — это перечисление из пространства имен `System.DirectoryServices.Protocols`, которое похоже на перечисление `SearchScope` из пространства имен `System.DirectoryServices` и, подобно ему, позволяет указывать, насколько глубоко должен вы-

полняться поиск. В коде выбирается его значение Subtree, чтобы поиск производился по всему дереву Active Directory.

Фильтр поиска может определяться с помощью строки LDAP или за счет использования XML-документа, содержащегося в классе XmlDocument:

```
string distinguishedName = null;
string ldapFilter = "(objectClass= user)";
string[] attributesToReturn = null; // возвращать все атрибуты
var searchRequest = new SearchRequest(distinguishedName,
    ldapFilter, SearchScope.Subtree, attributesToReturn);
```

После определения поискового запроса с помощью объекта SearchRequest этот запрос пересылается веб-службе вызовом метода SendRequest. Метод SendRequest доступен в классе DsmlSoapHttpConnection и возвращает объект SearchResponse, позволяющий считывать возвращаемые объекты.

Вместо синхронного метода SendRequest могут также вызываться и предлагаемые классом DsmlSoapHttpConnection асинхронные методы BeginSendRequest и EndSendRequest в соответствии с асинхронным шаблоном .NET.



*Асинхронный шаблон подробно рассматривается в главе 20.*

```
SearchResponse searchResponse =
    (SearchResponse) dsmlConnection.SendRequest(searchRequest);
```

Возвращаемые объекты Active Directory могут считываться внутри метода SearchResponse. В SearchResponse.Entries содержится коллекция всех записей, упакованных в оболочку типа SearchResultEntry. Класс SearchResultEntry имеет свойство Attributes, в котором содержатся все атрибуты. Каждый атрибут может считываться с помощью класса DirectoryAttribute.

В приведенном ниже коде отличительное имя каждого объекта выводится на консоль. Затем производится доступ к значениям атрибутов организационной единицы (ou), и имя организационной единицы тоже выводится на консоль. После этого на консоль также выводятся все значения объектов DirectoryAttribute.

```
Console.WriteLine("\r\n\r\n в результате поиска обнаружено {0} элемент(ов):",
    searchResponse.Entries.Count);
foreach (SearchResultEntry entry in searchResponse.Entries)
{
    Console.WriteLine(entry.DistinguishedName);

    // Извлечение конкретного атрибута
    DirectoryAttribute attribute = entry.Attributes ["ou"];
    Console.WriteLine("{0} = {1}", attribute.Name, attribute[0]);

    // Извлечение всех атрибутов
    foreach (DirectoryAttribute attr in entry.Attributes.Values)
    {
        Console.WriteLine("{0}=", attr.Name);

        // Извлечение всех значений атрибутов;
        // типом значения может быть string, byte[] или Uri
        foreach (object value in attr)
        {
            Console.WriteLine("{0} ", value);
        }
        Console.WriteLine();
    }
}
```

Добавление, изменение и удаление объектов выполняется аналогично их поиску и подразумевает использование соответствующих классов.

## Резюме

В этой главе рассматривалась архитектура Active Directory, а точнее — такие важные понятия, как домены, деревья и леса Active Directory. Доступ можно получать к информации по всему предприятию. При написании приложений, имеющих доступ к службам Active Directory, нужно помнить о том, что считываемые данные могут не быть актуальными из-за задержек репликации.

Классы из пространства имен `System.DirectoryServices` обеспечивают простой доступ к службам Active Directory за счет создания оболочки поставщиков ADSI.

Класс `DirectoryEntry` делает возможным чтение и запись объектов прямо в хранилище данных.

Класс `DirectorySearcher` позволяет выполнять сложный поиск и определять фильтры, тайм-ауты, загружаемые свойства и область поиска. Применение глобального каталога может существенно ускорить поиск объектов по всему предприятию, поскольку в нем хранятся доступные только для чтения версии всех объектов леса.

DSML — это еще один API-интерфейс, который позволяет получать доступ к Active Directory через веб-службы.

В пространстве имен `System.DirectoryServices.AccountManagement` предлагаются абстрактные версии классов из пространства имен `System.DirectoryServices`, которые значительно облегчают процессы создания и изменения учетных записей пользователей, групп и компьютеров.